N° 1410

50376
1994
301

20102548
50376
1994
301

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

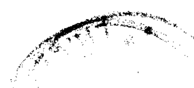
pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE

Spécialité Productique :
Automatique et Informatique Industrielle

par

Pascal REMY
Maître E.E.A.
Mastère E.C.LILLE.



CONCEPTION ET REALISATION D' UN PROCESSEUR POUR LA GENERATION D'UN MODELE BOND-GRAPH A PARTIR DE DIFFERENTES DESCRIPTIONS DE SYSTEMES PHYSIQUES SOUS LE LOGICIEL ARCHER

Soutenue le 22 novembre 1994 devant le jury d'examen :

Mr	P. BORNE	Président
Mr	D. MEIZEL	Rapporteur
Mr	M. LEBRUN	Rapporteur
Mr	J. P. CASSAR	Examinateur
Mr	P. YIM	Examinateur
Mme	G. DAUPHIN-TANGUY	Examinateur et directeur de Thèse
Mr	A. AZMANI	Examinateur et co-directeur

Thèse préparée au sein du L.A.I.L, U.R.A, CNRS D 1440, Ecole Centrale de Lille.

A la mémoire d'Anna Sedziak-Kostek,
Pour Joseph et Irène.

AVANT-PROPOS ET REMERCIEMENTS

Le travail présenté dans ce mémoire a été effectué au Laboratoire d'Automatique et d'Informatique Industrielle de Lille (LAIL) à l'Ecole Centrale de Lille sous la direction scientifique de Madame le Professeur G. DAUPHIN-TANGUY. Je tiens à la remercier chaleureusement pour son accueil, sa patience et pour les nombreux conseils que j'ai bénéficiés tout au long de ce travail.

Je remercie vivement Monsieur le Professeur P. Borne, Directeur Scientifique de l'Ecole Centrale de Lille, pour nous avoir fait l'honneur d'être le président du jury de Thèse.

Je suis très reconnaissant à Monsieur D. MEIZEL, Professeur à l'Université de Technologie de Compiègne et à Monsieur M. LEBRUN, Maître de Conférences et Docteur d'Etat à l'Université de Lyon I pour l'honneur qu'ils me font en acceptant d'examiner ce travail et d'être les rapporteurs de cette thèse.

Je suis honoré de la présence de Monsieur J.P. CASSAR, Maître de Conférences à l'Université des Sciences et Technologie de Lille (USTL) et P. YIM, Maître de Conférences à l'Ecole Centrale de Lille et les remercie d'avoir accepté d'être les examinateurs de mes travaux.

Je tiens à remercier tout particulièrement Monsieur A. AZMANI, Maître de Conférences à l'Ecole Centrale de Lille et co-directeur de cette thèse pour son aide et ses précieux conseils ainsi que les multiples discussions que nous avons entretenues pendant ces quelques années.

Je voudrais adresser une pensée aux membres du LAIL, au personnel de L'EC Lille et spécialement à l'équipe bond-graph (Philippe, David, Janette, Moncef, Laurent et tous les autres...) pour la bonne humeur et le climat de sympathie qu'ils entretiennent au sein du Laboratoire.

Enfin, je remercie très sincèrement Monsieur M. VANGREVENINGE pour la reprographie de ce mémoire.

TABLE DES MATIERES

CHAPITRE I. LA MODELISATION : POSITION DU PROBLEME ET DEFINITIONS

INTRODUCTION GENERALE 1
I.1. Introduction 7
I.2. Notion de modèle 7
I.2.1. Classes de modèles 9
I.2.1.1. Modèles de connaissance 9
I.2.1.2. Modèles de représentation et modèles de conduite 9
I.2.2. Formes de modèles 9
I.2.2.1. Modèles de règles / Systèmes experts 9
I.2.2.2. Modèles fichiers 10
I.2.2.3. Modèles par reconnaissance des limites du processus 10
I.2.2.4. Modèles mathématiques 10
A. Modèles d'états 10
B. Modèles entrées-sorties 10
C. Modèles par traitement du signal 11
D. Modèles par reconnaissance des formes 11
I.2.2.5. Modèles graphiques 11
A. Les schémas fonctionnels 11
B. Les graphes de fluence 12
C. Les Bonds-graphs 12
D. Les réseaux de Pétri et les grafcets 12
I.2.3. Modèles descriptifs 12
I.2.3.1. Modèles homogènes 13
I.2.3.2. Modèles hétérogènes 14
I.3. La triple Modélisation 14
I.3.1. Outils méthodologiques pour la modélisation fonctionnelle 15
I.3.1.1. Flow-model 15
I.3.1.2. SADT 17
I.3.1.3. Le Bond-Graph à Mot (BGM) 19
I.3.1.4. Comparaison des trois méthodes 21
I.3.2. Outils méthodologiques pour la modélisation structurelle 22
I.3.2.1. Graphe biparti 22
I.3.2.2. Propriétés structurelles par la causalité 22
I.3.3. Outils méthodologiques pour la modélisation comportementale 23
I.3.4. Positionnement du bond-graph dans la démarche de modélisation 24
I.4 Le projet ARCHER 25
I.4.1. Logiciels utilisant l'approche bond-graph 25
I.4.2. ARCHER 27
I.5. Conclusion 31

CHAPITRE II. DESCRIPTION DE MODELES SOUS ARCHER

II.1. Introduction 35
II.2. Problématique 36
II.2.1. Aspects topologiques 37
II.2.1.1. Description descendante 37
II.2.1.2. Type de modèles des sous-parties 38
II.2.2. Couplage des blocs bond-graph 40
II.2.2.1. Le port 40
II.2.2.2. Le bloc 41
II.2.3. Cahier des charges du couplage de blocs bond-graphs 42
II.2.3.1. Comment assembler les blocs ? 42
A. Système formé exclusivement de blocs à une entrée et à une sortie 44
B. Système formé en majorité de blocs à plusieurs entrées et sorties 44
II.2.3.2. Comment générer un modèle global ? 46
A. Première étape de la procédure de génération d'un modèle 46
B. Deuxième étape de la procédure 48
II.2.3.3. Quelle convention adopter pour identifier et différencier les éléments des blocs BG ? 49
II.2.4. Conclusion 51
II.3. Description d'un modèle constitué de blocs dipôles 52
II.3.1. Notions de bases 52
II.3.1.1. La notation série et parallèle 52
II.3.1.2. Le point de connexion 53
II.3.2. Blocs et composants élémentaires 55
II.3.2.1. Electriques 55
II.3.2.2. Electroniques 56
II.3.2.3. Mécaniques 56
II.3.2.4. Hydrauliques 58
II.3.2.5. Conclusion 58
II.3.3. Langage pour un modèle homogène 59
II.3.3.1. Electrique (e) 59
II.3.3.2. Mécanique à 1 dimension (m) 60
II.3.3.3. Hydraulique (h) 61
II.3.3.4. Conclusion 62
II.3.4 Génération d'un Bond-graph à partir d'une description texte 63
II.3.4.1. Composants élémentaires et éléments bond-graphs 63
II.3.4.2. Principe de la méthode 65
A. Systèmes électriques 65

B. Systèmes Mécaniques....68
II.3.4.3. Exemples....70
A. Système électrique....70
B. Système mécanique....71
II.3.5. Etude de la description d'un bloc....72
II.3.5.1. A partir d'éléments de bases appartenant au même domaine....72
II.3.5.2. A partir d'éléments de base appartenant à des domaines différents....75
II.4. Bond-graphs et blocs Bond-graphs....76
II.4.1. Langage de description bond-graph sous forme texte....77
II.4.2. Description d'un bloc bond-graph....79
II.5. Description avec Blocs à plusieurs entrées et sorties....80
II.5.1. Couplage....80
II.5.1.1. Avec un bond-graph causal....80
II.5.1.2. Bond-graph acausal....83
II.5.2. Choix d'une syntaxe adaptée pour le couplage....85
II.5.2.1. La partie déclaration des blocs....85
II.5.2.2. La partie description d'un réseau Blocs & Noeuds....86
II.5.2.3. Avantages du langage....88
II.5.2.4. Sélection des ports....88
II.5.2.5. Choix du noeud de couplage....89
II.6. CONCLUSION....92

CHAPITRE III. CONSTRUCTION D'UN COMPILATEUR POUR LES LANGAGES DE DESCRIPTION SOUS ARCHER

III.1. Introduction 95
III.2. Représentations symboliques des données d'ARCHER 96
III.2.1 Outils informatiques 96
III.2.1.1. PROLOG 96
III.2.1.2. TURBO PROLOG 97
III.2.1.3. La liste 99
III.2.1.3 Le graphe en PROLOG 100
III.2.2. Codage bond-graph sous Archer 102
III.2.2.1. Le bond-graph et le graphe 102
III.2.2.1. Bond-graph acausal 104
III.2.2.2. Bond-graph causal 106
III.2.3. Les Limites du langage Prolog 106
III.3. Construction d'un compilateur pour ARCHER 107
III.3.1. Analyseur lexical 109
III.3.1.1. Décomposition d'une ligne 109
III.3.1.2. Traitement du lexème 110
III.3.2. Analyseur syntaxique 113
III.3.2.1. Automate à états finis 115
III.3.2.2. Erreurs syntaxiques 119
III.3.2.3. Automate en prolog 121
III.3.3. Analyse sémantique et génération de code 121
III.3.4. Analyse post-sémantique 123
III.3.4.1. Erreurs topologiques 124
III.3.4.2. Simplification 125
III.3.4.3. Validation du sens physique 128
III.3.4.4. Analyse pré-causale 132
III.3.4.5. Tableau de conversion 134
III.4. Conclusion 137

CHAPITRE IV. LANGAGES DE DESCRIPTION ET GENERATION DE CODE BOND-GRAPH

IV.1. Introduction 141
IV.2. Description homogène d'un système physique 142
IV.2.1. Langage bond-graph 142
IV.2.1.1. Analyseur syntaxique 143
IV.2.1.2. Analyse pré-sémantique 145
IV.2.1.3. Génération de code 147
IV.2.1.4. Exemple d'application 148
IV.2.2. Langage Série & Parallèle 150
IV.2.2.1. Analyseur syntaxique 151
IV.2.2.2. Analyse pré-sémantique 153
IV.2.2.3. Génération de code 153
A. Electrique 153
B. Electronique 155
C. Mécanique 156
IV.2.2.4. Phase de simplification 158
IV.3. Création d'un bloc bond-graph 159
IV.3.1. Bloc en langage bond-graph 160
IV.3.1.1. Analyse syntaxique 160
IV.3.1.2. Analyse pré-sémantique 162
IV.3.1.3. Génération de code 162
IV.3.1.4. Exemple d'un bloc hydro_mécanique 163
IV.3.2. Bloc en Langage série & parallèle 166
IV.3.2.1. Analyse syntaxique 166
IV.3.2.2. Génération de code 166
IV.3.2.3. Exemple d'un bloc électrique 167
IV. 4. Langage Blocs & noeuds 170
IV.4.1. Analyse syntaxique 171
IV.4.1.1. Grammaire du langage 171
IV.4.1.2. Erreurs syntaxiques : 173
IV.4.2. Analyse pré-sémantique 174
IV.4.2.1. Partie déclaration 174
IV.4.2.2. Partie description 176
IV.4.2.3. La Fusion 177
IV.4.2.4. Règles de couplage 178
IV.4.3. Génération d'un code bond-graph unique 182
IV.4.3.1. La Renumérotation 182
IV.4.3.2. Mixage des différents codes 183
IV.4.4. Aspects évolutifs du langage 184
IV.4.4.1. Le pseudo-bloc 184

IV.4.4.2. Bloc en langage blocs & noeuds 185
IV.4.5. Modélisation d'une machine outil 186
IV.5. Bond-graph et bloc mathématique. 192
IV.5.1. Position du problème 192
IV.5.2. Couplage Digraphe 193
IV.5.2.1. Digraphe.. 194
A. A partir d'une équation d'état.. 194
B. A partir d'une fonction de transfert 194
C. A partir d'un bond-graph 195
IV.5.2.2. Codage et langage d'un digraphe 195
A. Le Codage 195
B. Le Langage 196
IV.5.2.3. Bloc digraphe 197
IV.5.2.4. Couplage digraphe et bond-graph.. 199
A. Couplage digraphe + digraphe 199
B. Couplage digraphe + bond-graph 199
C. Couplage bond-graph + digraphe 200
D. Couplage bond-graph avec un noeud signal 201
IV.6.Conclusion 202

CONCLUSION GENERALE.. 207

ANNEXE 1 : LANGAGE ET GRAMMAIRE 211
1. Définitions 211
2. Hiérarchie des grammaires 213
ANNEXE 2 : AUTOMATE EN PROLOG.. 215
ANNEXE 3 : ALGORITHMES DU LANGAGE SERIE PARALLELE 223
ANNEXE 4 : SESSION COMPLETE D'UN COUPLAGE BOND-GRAPH 231
ANNEXE 5 : BIBILIOTHEQUE DE BLOCS PRE-DEFINIS POUR
L'APPLICATION DU CHAPITRE IV 245

REFERENCES BIBLIOGRAPHIQUES 249

INTRODUCTION GENERALE

INTRODUCTION GENERALE

La modélisation représente une grande part des difficultés rencontrées lors de l'étude des systèmes physiques, surtout lorsqu'ils sont composés de sous-ensembles de nature différente (électro-mécanique, hydraulique, mécanique...).

La nécessité de définir des méthodologies et des outils unifiés apparaît primordiale lorsque le rôle du modèle dépasse la seule simulation et intervient dans une démarche de conception intégrée.

Le logiciel ARCHER développé au LAIL est un outil d'aide à la modélisation et à l'analyse, fondé sur l'outil bond-graph. Il permet, à partir du codage informatique du bond-graph, de lui affecter la causalité, de l'analyser (étude des propriétés structurelles), de construire les équations mathématiques qui lui sont associées (sous la forme d'expressions formelles), de dessiner le bond-graph automatiquement à l'écran, et de le simuler par couplage avec des solveurs d'équations comme MATLAB, ASCL ou MATHEMATICA.

Notre travail a consisté dans une première phase, à réfléchir sur les différentes formes de données possibles pour l'introduction du modèle.

Une première forme est un langage utilisateur proche du langage naturel. Il permet, à partir d'une représentation "type réseau" du système, une description texte des liens de connexion et de l'architecture en utilisant les notions d'association "série" et "parallèle". Cette approche se prête bien aux systèmes assez simples appartenant à un domaine physique d'un seul type et particulièrement pour les systèmes électriques ou à représentation topologique analogue.

La deuxième forme est un langage bond-graph qui permet d'entrer sous forme texte le modèle bond-graph. Aucune limitation de complexité n'est à prendre en compte, même si la version actuelle d'ARCHER n'autorise que les éléments 1- et 2-ports.

La troisième forme est un langage de modèle mathématique. Cette forme pose des problèmes particuliers sur la représentation des données (équation d'état, fonction de transfert) pour les rendre compatibles avec celles déjà existantes dans ARCHER.

Un compilateur unique, possédant un analyseur syntaxique et un analyseur sémantique, a été développé sous une forme générale qui s'applique à tout type de données.

Pour simplifier la démarche de modélisation, il est intéressant de disposer d'une bibliothèque de modèles, et donc de pouvoir en extraire des modèles pour les coupler et construire un système complexe.

Ce couplage entre modèles pose des problèmes différents suivant la forme des données des différents modèles. Une démarche structurée, avec des étapes de validation est proposée dans ce mémoire.

Le mémoire se décompose en quatre chapitres :

Dans le **premier chapitre**, nous présentons la modélisation d'un système physique à travers les aspects structurels, fonctionnels et comportementaux qu'il est possible de trouver à travers plusieurs méthodes. Nous orientons notre choix sur la méthodologie bond-graph car elle permet, à travers un jeu de quelques entités génériques, la représentation de nombreux systèmes physiques notamment par l'emploi du bond-graph à mots. Nous présentons la philosophie et les généralités du projet ARCHER.

Le **deuxième chapitre**, traite du problème de la décomposition d'un système physique en sous-systèmes et de leur couplage.

Pour résoudre le problème du couplage d'un point de vue informatique, nous proposons de représenter chaque sous-systèmes par un bloc objet muni de ports d'entrées et de sorties. Des noeuds de connexions permettent le couplage de ces blocs. Un langage permettant la description et le couplage des blocs est défini.

Dans le **troisième chapitre**, nous proposons une structure unique du compilateur pour englober le traitement syntaxique des différents langages à travers l'utilisation d'automates à états finis.

Dans la première partie de ce chapitre, nous rappelons les principes du langage Turbo-Prolog et son utilisation dans la représentation des graphes ainsi que la forme et le codage des données bond-graphs générées lors des différentes phases du compilateur.

Dans la deuxième partie, nous démontrons qu'il est possible, à partir du bond-graph acausal, de trouver d'une part, des méthodes heuristiques afin d'anticiper d'éventuels conflits causaux lors de l'affectation de la causalité et d'effectuer d'autre part, une analyse qualitative sur la propagation de la puissance pour éviter des non sens physiques qui sont le résultat d'une mauvaise description du système originel.

Enfin dans le **quatrième chapitre**, nous formalisons pour chaque type de langage proposé, sa syntaxe et sa grammaire exactes avec les automates syntaxiques respectifs, l'analyse sémantique et la génération de code, à travers de nombreux exemples.

Nous proposons une structure de données pour la création et le stockage des blocs bond-graphs, un dialogue pour gérer la sélection des ports, des règles de couplages lorsqu'il y a incompatibilité des domaines physiques en présence. Nous présentons un exemple significatif afin de montrer l'approche originale du langage blocs et noeuds.

Dans une optique d'extension et de généralisation, nous définissons les perspectives de couplages avec des blocs autres que bond-graph en particulier des blocs à caractère mathématique à travers la représentation par digraphes.

CHAPITRE I

LA MODELISATION : POSITION DU PROBLEME ET DEFINITIONS

LA MODELISATION : POSITION DU PROBLEME ET DEFINITIONS

I.1. Introduction

Le problème étudié dans cette thèse concerne la définition d'un langage pour la description des systèmes physiques.

Dans un premier temps, nous allons rappeler les principales méthodes de modélisation, en insistant sur les méthodes qui permettent de prendre en compte la topologie du système et en donnant quelques exemples de "modèles descriptifs".

Nous présentons les trois formes de modélisation nécessaires pour englober toutes les connaissances du système à savoir : structurelle, fonctionnelle et comportementale ; ainsi que quelques méthodes utilisant ces approches.

Enfin nous présentons, le logiciel ARCHER en nous situant dans son fonctionnement actuel, ainsi que quelques logiciels existant sur le marché et leur situation par rapport à ARCHER.

I.2. Notion de modèle

La modélisation est nécessaire à toute étude d'un système dynamique pour sa compréhension et son amélioration.

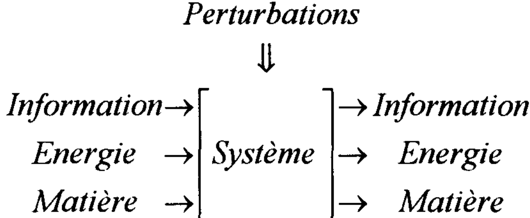
Un modèle est issu de la rencontre d'un système et de son concepteur : cela veut dire qu'il n'y a pas une façon unique de représenter un système ; en effet un modèle ne peut pas être "exact", il est une vision partielle des phénomènes mis en jeu dans la réalité d'un processus, en fonction des buts poursuivis.

Exemple :

Le processus consistant à fondre un sucre sera modélisé par un chimiste comme la transformation du saccharose en caramel (niveau macroscopique), par un physicien comme la cassure des liaisons moléculaires par l'agitation des électrons et des composantes du noyau (niveau microscopique), par un économiste comme la fabrication d'un élément de base nécessaire aux industries alimentaires.

$$\left.\begin{matrix} Chimiste \\ Physicien \\ Economiste \end{matrix}\right\} \rightarrow \left\langle \begin{matrix} Sucre \\ fondu \end{matrix} \right\rangle \rightarrow \left\{ \begin{matrix} Modèle \\ non \\ unique \end{matrix} \right.$$

Ce processus est un système dynamique (évolutif avec le temps) et peut être schématisé par une "boîte" traversée par des flux d'information, d'énergie, et de matière tout en étant soumise à des perturbations ayant l'une des trois formes précitées.



Cet exemple "simpliste" démontre que le choix de la méthode utilisée dépend des objectifs fixés par le concepteur. Ces objectifs peuvent être regroupés dans quatre directions principales : **concevoir, comprendre, prévoir et commander (éventuellement surveiller).**

Mais avant d'atteindre ces buts, se pose le choix d'un modèle qui aidera à la perception et à la compréhension du processus. Voyons comment pourrait se déduire un modèle à partir des différentes méthodes existantes.

L'absence de méthode universelle permettant de construire un modèle respectant fidèlement le système étudié, a fait apparaître un certain nombre de méthodes utilisées selon le contexte et l'objectif recherchés.

Dans cette section, nous présentons succinctement les classes de modèles classiquement utilisées en automatique [BORNE et AL 92].

I.2.1. Classes de modèles

I.2.1.1. Modèles de connaissance

Un modèle de connaissance est un modèle dont les caractéristiques et les équations ont été établies en faisant appel à des modèles plus généraux mettant en oeuvre les lois de la physique, de la chimie, de la biologie, de l'économie... Les paramètres d'un tel modèle ont alors une interprétation physique directe : température, pression, courant, accélération, force... Ils ont beaucoup plus de signification que les modèles de représentation définis ci-dessous et contiennent toutes les informations utiles sur le processus étudié. Par contre, ils sont en général difficiles à déterminer et de mise en oeuvre complexe. Ils se présentent généralement, sous forme d'équations différentielles d'ordre élevé ou d'équations aux dérivées partielles.

I.2.1.2. Modèles de représentation et modèles de conduite

Ces modèles ne permettent pas, le plus souvent, d'interprétation physique des phénomènes étudiés. Ils sont constitués d'un ensemble de relations mathématiques qui vont relier dans un domaine d'évolution donnée, les différentes variables du processus. Les paramètres de tels modèles peuvent n'avoir aucun sens physique particulier connu.

I.2.2. Formes de modèles

I.2.2.1. Modèles de règles / Systèmes experts

Ils correspondent à une description par règles de conduite issues de l'observation du système pendant son fonctionnement.

$$\left.\begin{array}{r}\textbf{si A est actionné}\\ \textbf{et que B est sur ON}\end{array}\right\}\ \textbf{alors C avance}$$

Bien que lourds à manier et limités du point de vue possibilités, ces modèles se prêtent assez bien à une exploitation mettant en oeuvre les techniques de l'intelligence artificielle. On les retrouve essentiellement dans les systèmes experts.

Le système expert est composé classiquement d'une base de connaissances et d'un moteur d'inférences. Facilement modifiable par un spécialiste du domaine traité, la base de connaissance contient généralement deux parties, la base de faits est constituée d'informations provenant des systèmes de détection ou des opérateurs humains [GERTLER 88].

I.2.2.2. Modèles fichiers

Ils sont constitués d'informations, sous forme de tableaux de données, reliant l'évolution des sorties du processus à l'évolution des entrées pour diverses classes d'entrées. Ces modèles constituent le plus souvent le point de départ en vue de l'élaboration de modèles plus évolués.

I.2.2.3. Modèles par reconnaissance des limites du processus

Très répandue en milieu industriel, cette méthode est basée sur l'observation du fonctionnement du système par un opérateur ou un dispositif automatique. Les grandeurs sont comparées en ligne à des seuils min et max bas prédéterminés. Le modèle du système est donc formé de ces inégalités.

I.2.2.4. Modèles mathématiques

A. Modèles d'états

Ils sont caractérisés par un ensemble de variables, en nombre minimum, regroupées dans un vecteur x de R^n appelé **vecteur état**.

La connaissance de x à l'instant t_0 ($x(t_0)$) associée à la connaissance de l'évolution des entrées u sur l'intervalle $\tau \in [\ t_0,\ t\ ]$ permet, à partir du modèle, de prévoir l'évolution de x(t) sur τ à l'aide des équations :

$$\begin{cases} \dot{X} = AX + BU \\ Y = CX + DU \end{cases}$$

B. Modèles entrées-sorties

Dans ce type de représentation, les entrées et sorties du processus sont liées par un ensemble de relations mathématiques (algébriques, équations différentielles, relations récurrentes...) de type égalité, inégalité ou inclusion suivant les contraintes ou saturations éventuelles.

Dans le cas de processus linéaires stationnaires à état continu, les relations entrées-sorties peuvent être définies par des **matrices de transfert** (en s ou en z).

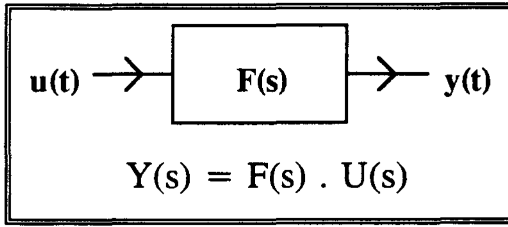


fig 1.1 : Fonction de transfert.

C. Modèles par traitement du signal

Les signaux de sortie se décomposent fréquemment en des composantes basses fréquences de grande amplitude et des composantes hautes fréquences de faible amplitude. Ces dernières peuvent être caractérisées par :

- leur loi de distribution statistique : moyenne, variance, écart type ...

- la densité spectrale et la fonction autocorrélation à l'aide de la transformée rapide de fourrier [ZWINGELSTEIN 79].

D. Modèles par reconnaissance des formes

Parfois, le degré de complexité d'un système est tel que l'utilisation d'un modèle classique est difficilement envisageable. Il est alors nécessaire d'avoir recours à des modèles construits à partir des seules informations fournies par les capteurs et par l'homme.

I.2.2.5. Modèles graphiques

Les modèles graphiques constituent un mode de représentation en général assez aisé à manipuler. La représentation graphique possède des propriétés remarquables pouvant aider à la modélisation et parfois permettre sa validation. Les principaux types de modèles graphiques sont :

A. Les schémas fonctionnels

Ce mode de description, appelé également représentation par schéma-blocs, est souvent utilisé de façon intuitive. Il correspond à une description directe des divers éléments du processus étudié faisant ou non intervenir les diverses relations mathématiques mises en oeuvre.

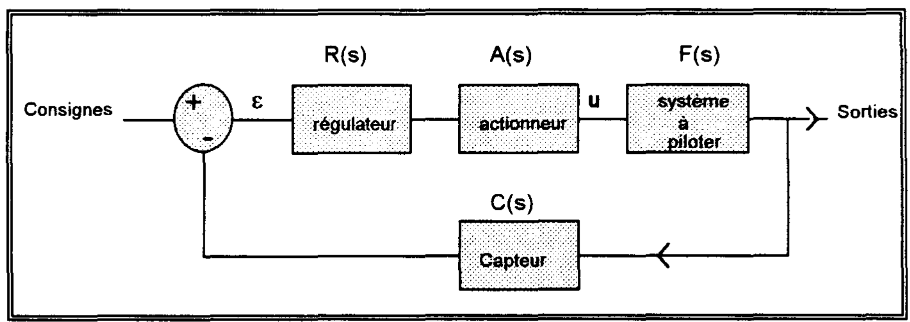


fig 1.2 : Exemple type d'un schéma blocs.

B. Les graphes de fluence

Ils correspondent à une représentation duale des schémas fonctionnels de figure 1.2 et ont pour principal intérêt la modélisation des systèmes linéaires pour lesquels il est possible d'appliquer le théorème de superposition :

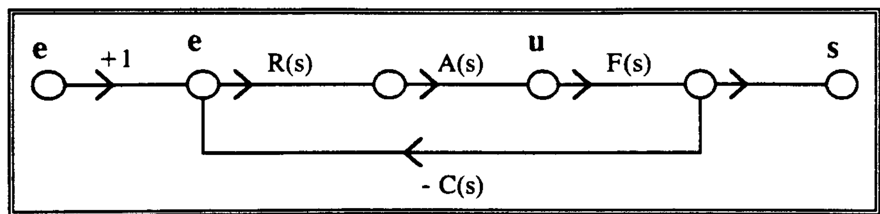


fig 1.3 : Graphe de fluence de la figure 1.2.

C. Les Bonds-graphs

Ils permettent une description des systèmes physiques (mécaniques, électriques, hydrauliques...) très bien adaptée à la modélisation des transferts de puissance, avec un langage unique quel que soit le domaine physique concerné.

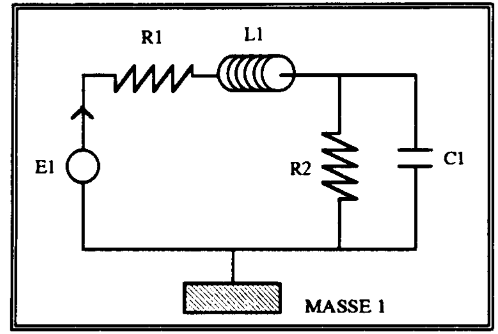


fig 1.4 : Schéma d'un système électrique.

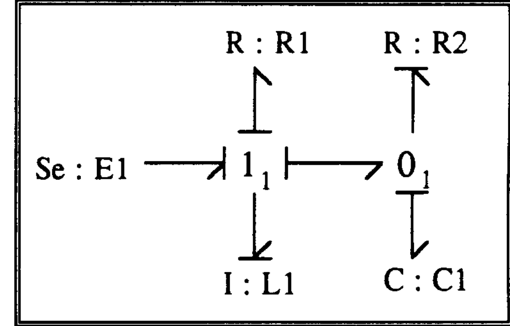


fig 1.5 : Bond-graph de la figure 1.4.

D. Les réseaux de Pétri et les grafcets

Ces représentations sont particulièrement adaptées aux processus à événements discrets dont le nombre d'états est fini.

Cette méthode est principalement utilisée dans la caractérisation des systèmes logiques, des systèmes de fabrication automatisés et des ateliers flexibles.

I.2.3. Modèles descriptifs

Nous allons nous intéresser particulièrement aux systèmes qui, par leur topologie, permettent de mettre en évidence des parties distinctes et les lois physiques qui les réunissent, afin d'obtenir une représentation symbolique du système. Cette opération peut être réitérée aux différentes parties dégagées jusqu'à l'obtention d'une entité irréductible que l'on sait modéliser.

I.2.3.1. Modèles homogènes

Les modèles homogènes sont des systèmes principalement constitués d'éléments appartenant au même domaine physique.

Dans le schéma électrique de la figure 1.6, des parties se distinguent facilement par les éléments de base : résistances (R), capacités (C), Intensités (L), sources de tension et de courant (E et I), et il est aisé de décrire le système en suivant le sens du courant par l'application de la loi des noeuds et des mailles (Kirshoff).

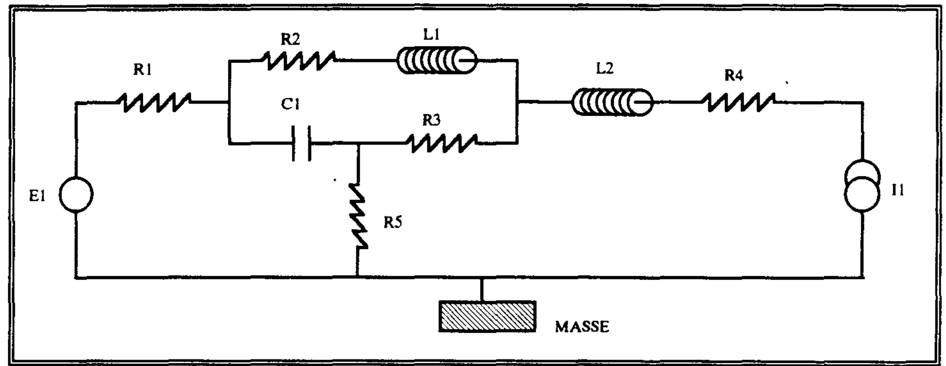


fig 1.6 : Système électrique.

De la même façon, dans le schéma mécanique à une dimension de la figure 1.6, on peut aussi dégager des parties principales : amortisseur, ressort, masse, bâti, plus des informations inhérentes à certains de ces éléments.

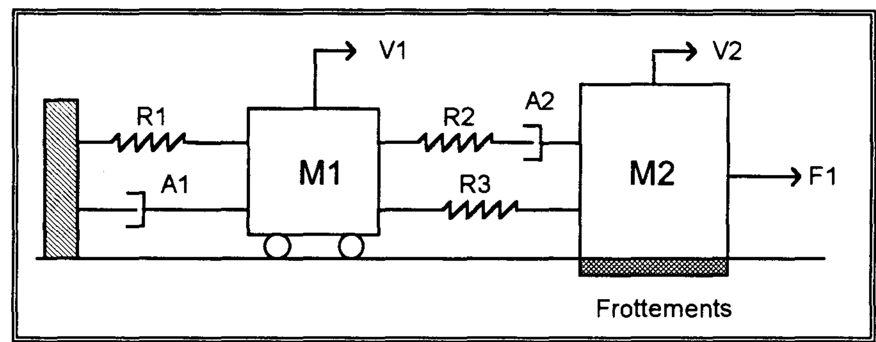


fig 1.7 : Système mécanique à 1 dimension.

Pour ces deux types de schéma physique, qui sont déjà une représentation de la réalité, une représentation "réseau" a été adoptée.

La notion de connexion "série" et "parallèle" apparaît nettement, ce qui permet d'écrire simplement des lois mathématiques associées aux éléments, aux noeuds et aux mailles.

I.2.3.2. Modèles hétérogènes

On appelle un système hétérogène, un système composé de sous-parties appartenant à des domaines physiques différents.

Prenons le système physique de la figure 1.8 composé des domaines électrique, mécanique et hydraulique.

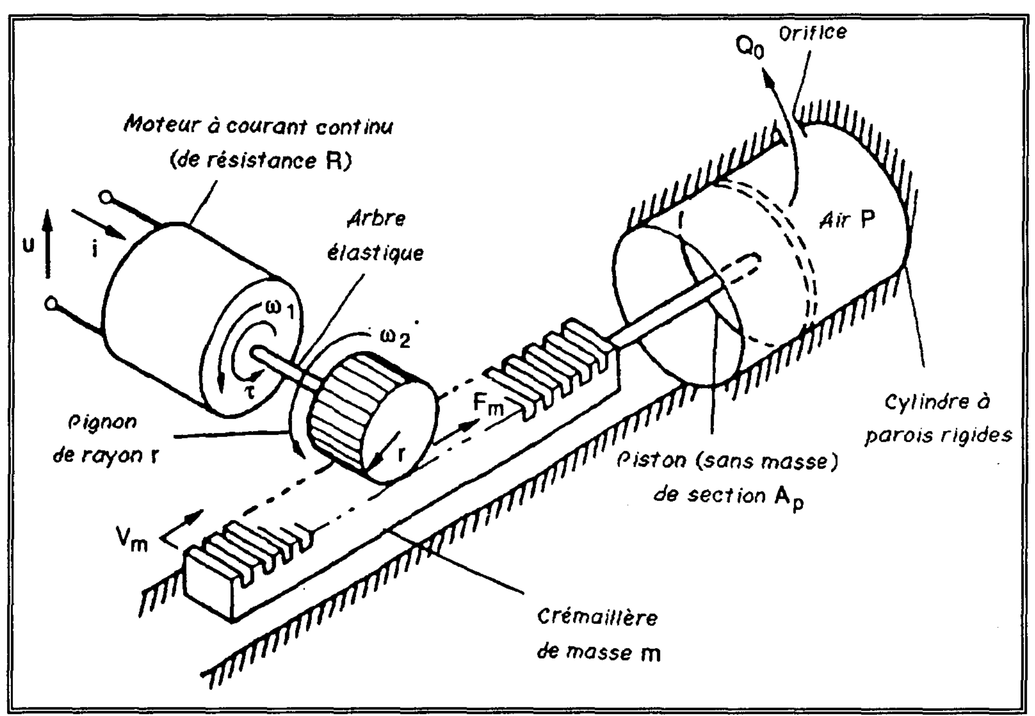


fig 1.8 : Système à crémaillère faisant intervenir plusieurs domaines physiques.

Ici le schéma fait apparaître des sous-systèmes complexes. La notion de connexion "série" et "parallèle" n'a plus de sens, puisque les variables mises en jeu sont de nature différentes.

I.3. La triple Modélisation

Une modélisation très complète d'un système physique comporte trois phases :

- Une première phase, conduisant à la construction d'un modèle "fonctionnel", représentant les activités qu'il réalise à travers des fonctions. Ces activités sont organisées sous forme d'une hiérarchie.

- Une deuxième phase où la définition d'un modèle "structurel" décrit en même temps la structure des couplages dans le système et les ensembles de variables liées.

- Une troisième phase dans laquelle un modèle "comportemental" permet de décrire les trajectoires des variables de sa structure, qu'elles soient **quantitatives** (les relations de contrainte sont alors des équations différentielles ou algébriques, des abaques, etc...) ou **qualitatives** (les relations de contrainte sont alors des règles, des graphes d'état, des processus de Markov, etc...).

Les modèles de commande traditionnellement utilisés par les automaticiens (équations d'états, fonction de transfert) ne couvrent qu'une petite partie du problème de la représentation des processus complexes.

Ils sont généralement des modèles de comportement (ils ne représentent ni l'aspect fonctionnel, ni l'aspect structurel du système) et sont souvent limités à des variables quantitatives. L'approche par logique floue permet cependant d'utiliser des informations qualitatives pour la commande.

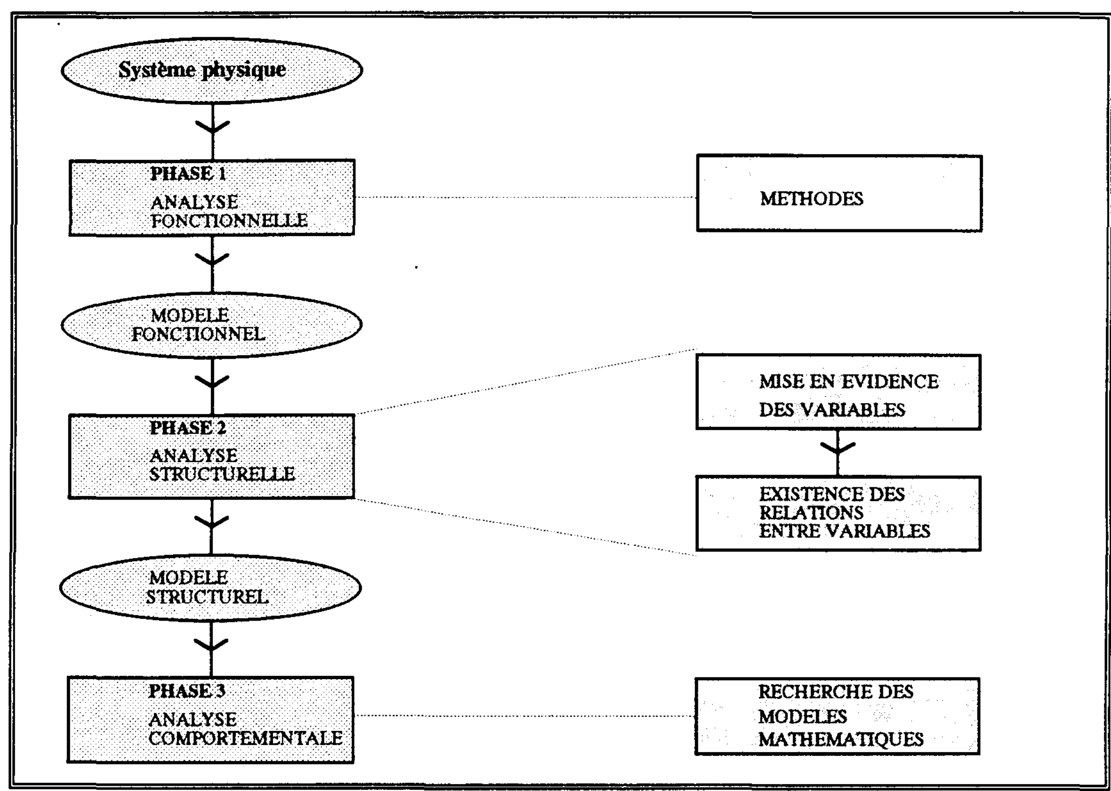


fig 1.9 : Différentes phases d'analyses d'un système physique.

La triple représentation : fonctionnelle, structurelle, comportementale, constitue une véritable base de connaissances relatives aux variables du processus et à leur couplage.

I.3.1. Outils méthodologiques pour la modélisation fonctionnelle

I.3.1.1. Flow-model

La méthode flow-model [LINDT 82] [RASMUSSEN 85] [TABORIN 89] est une démarche qui permet la construction du modèle fonctionnel d'un processus thermo-dynamique. Elle est basée sur l'identification des structures de flot de puissance à différents niveaux d'agrégation physique du système.

Représentant les aspects qualitatifs des fonctions du processus, le graphe résultat de la modélisation décrit la topologie des circuits massiques et énergétiques. Il est constitué à l'aide de noeuds correspondant chacun à une fonction du processus. Les différentes fonctions du flow-model sont décrites dans le tableau figure 1.10.

ENTITES	SYMBOLE		FONCTIONS
	MASSE	ENERGIE	
SOURCE			Propriété du système de se comporter comme un réservoir infini de masse ou d'énergie. Source : réservoir fournissant de la masse ou de l'énergie
PUITS			Puits : réservoir gardant de la masse ou de l'énergie
STOCKAGE			Propriété d'accumulation de la masse ou de l'énergie Le paramètre qui le caractérise indique le niveau de masse ou d'énergie
TRANSPORT			Propriété de transfert de la masse ou de l'énergie entre deux systèmes. Elle est caractérisée par le débit massique ou énergétique
DISTRIBUTION			Propriété de répartition entre les flots entrants et les flots sortants. Les rapports de distribution la caractérisent.
BARRIERE			Elle représente la propriété du système d'empêcher le transfert de matière ou d'énergie entre deux systèmes différents

fig 1.10 : Fonctions de bases du flow-model.

Au cours de ce paragraphe, nous allons présenter l'exemple d'une foreuse hydraulique (figure 1.11) qui sera modélisée par trois méthodes différentes. Le Flow-model, que nous présentons, la méthode SADT et finalement le Bond-graph à mot (BGM).

Une discussion suivra pour justifier le choix de la méthode la plus avantageuse pour le logiciel ARCHER.

Exemple :

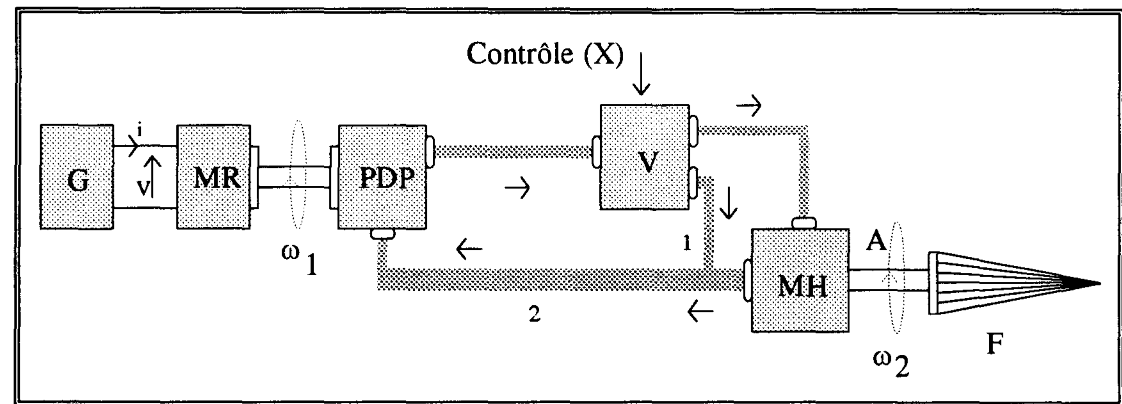


fig 1.11 : Foreuse hydraulique.

La machine de la fig 1.11 fait intervenir les domaines électrique, hydraulique, mécaniques et la topologie du système permet de dégager trois parties distinctes en relation avec ces mêmes domaines.

- Dans la **partie 1**, le moteur rotatif **(MR)**, alimenté par une tension **(u)** et un courant **(i)**, fournit une vitesse de rotation ω_1 pour le fonctionnement d'une pompe à déplacement positif **(PDP)** du circuit hydraulique de la **partie 2**.

- Le moteur hydraulique **(MH)** a une valve **V** en amont pour contrôler la puissance, dans la **partie 3**, délivrée à l'arbre **(A)**, qui par une vitesse de rotation ω_2 fait tourner le foret **(F)**.

- Le fluide résultant du moteur hydraulique retourne à la PDP.

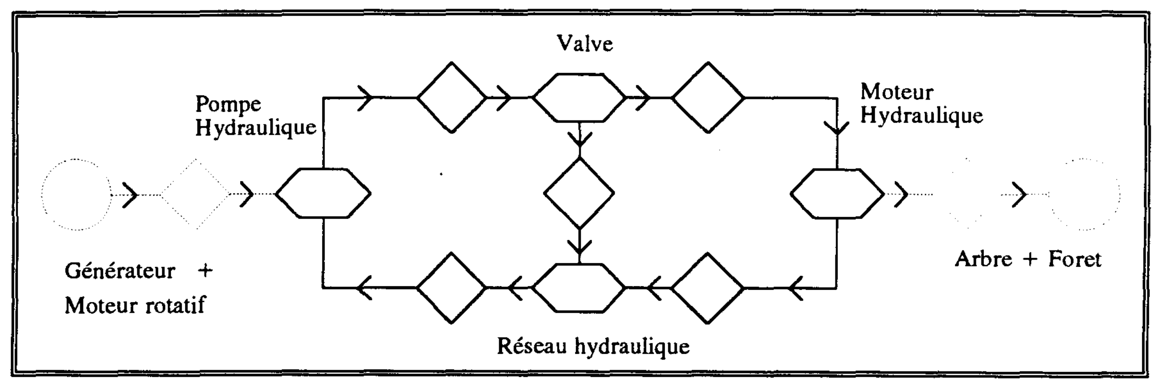


fig 1.12 : Flow-model de la foreuse.

I.3.1.2. SADT

La méthode SADT (Structured Analysis and Design Technique), d'origine américaine fondée par DOUGLAS T. ROSS [ROSS 77], a été introduite et diffusée par MICHEL LISSANDRE [LISSANDRE 86] en Europe vers 1976.

Elle conduit à décrire le système sous forme de fonctions présentées de manière hiérarchisée et structurée. Cette description met ainsi en évidence les parties qui constituent le système, la finalité de chacune, mais aussi les interfaces entre les diverses parties. Le système n'est pas une collection d'éléments indépendants, mais une organisation structurée de ceux-ci sous la forme de boîtes et de flèches constituant des diagrammes codifiés selon une certaine grammaire.

Cette méthode est plutôt utilisée pour la gestion de production dans différentes entreprises (banque, informatique, télécommunication...). On peut néanmoins l'utiliser pour modéliser des systèmes physiques en employant un modèle dérivé de SADT [DECLERCK 91].

Le modèle SADT est composé principalement :

- de diagrammes d'activités ou actigrammes représentant l'ensemble des activités du système
- de diagrammes de données ou datagrammes représentant l'ensemble des données du système.
- d'une liste hiérarchique du système analysé.

Actigramme : Un diagramme d'activités est identifié par un verbe d'action qui

- génère, crée une donnée en sortie
- transforme, modifie, change une donnée d'entrée à partir de directives de contrôle

comme le montre la figure 1.13 .

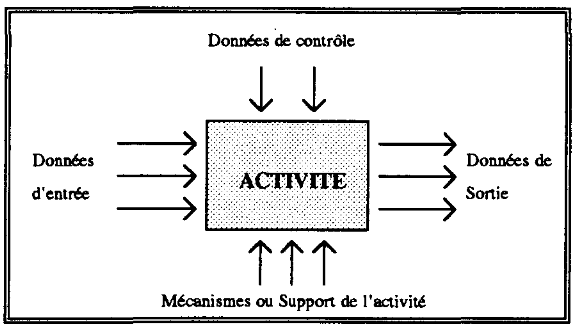


fig 1.13 : Actigramme.

Datagramme : Un diagramme de données crée, à partir d'activités d'entrée (activités génératrices), une activité de sortie (activités utilisatrices) sous le contrôle d'activités de création. Voir figure 1.14 .

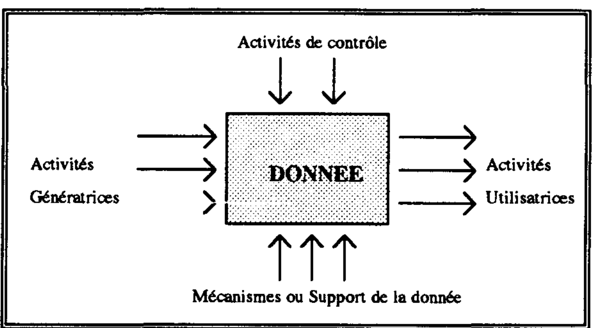


fig 1.14 : Datagramme.

Les deux diagrammes sont en quelque sorte duaux l'un de l'autre. Le choix d'un datagramme ou d'un actigramme dépend de la volonté de mettre en évidence, dans le modèle, les données ou les activités.

Prenons l'exemple de la foreuse figure 1.11 et construisons, en utilisant l'actigramme, son modèle SADT figure 1.15.

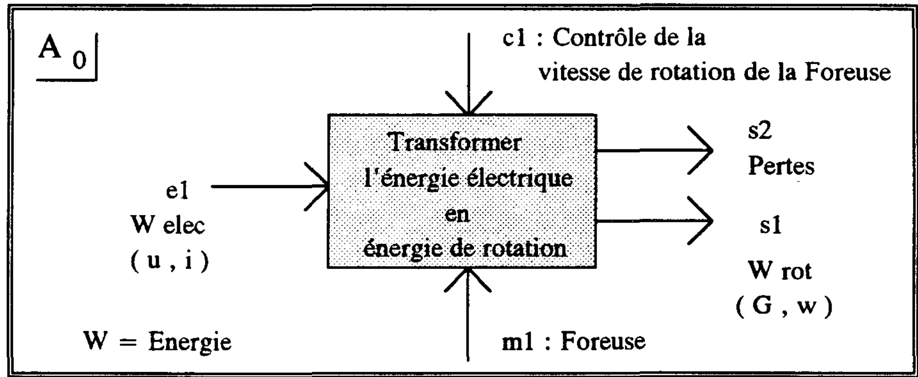


fig 1.15 : Actigramme de la foreuse (niveau 0).

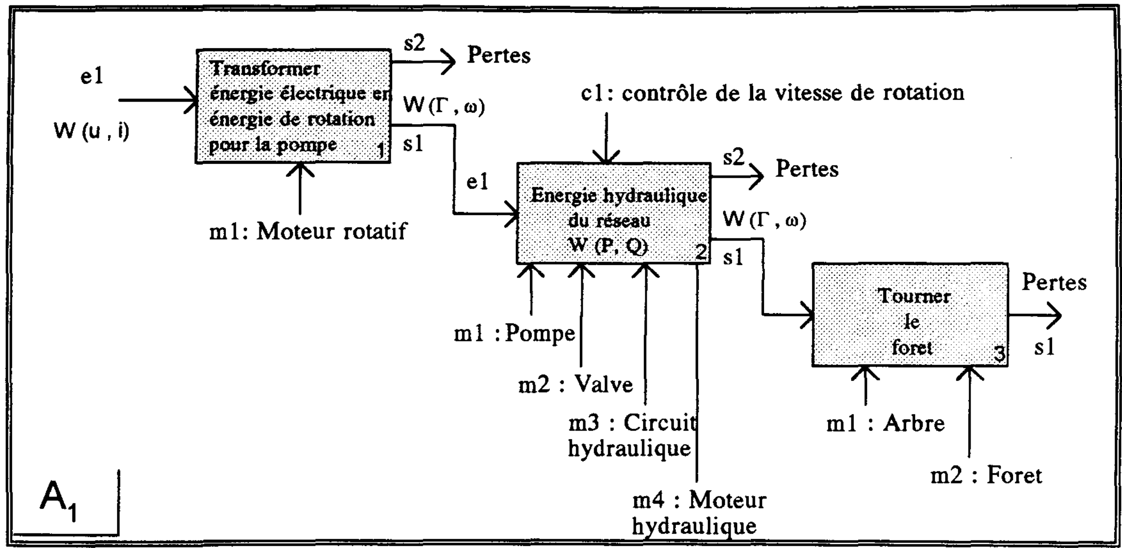


fig 1.16 : Niveau 1.

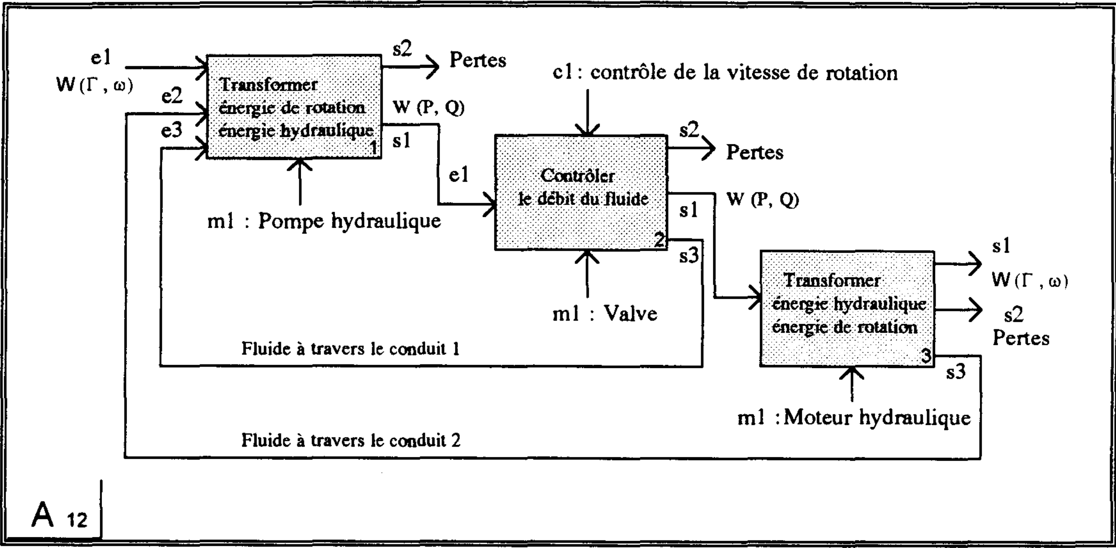


fig 1.17 : Représentation plus détaillée de l'actigramme 2 du niveau 1.

I.3.1.3. Le Bond-Graph à Mot (BGM)

Les Bond-graphs sont basés sur une décomposition du système en composants qui échangent de l'énergie ou de la puissance à travers des connexions appelées ports. Avec cela, il est possible de dessiner le bond-graph du système en représentant, dans un premier temps, les éléments du système par des "mots" et les liens de puissance échangés par de simples lignes orientées suivant le sens de la puissance.

Bien entendu, cette méthode s'adresse avant tout au bond-graphiste, mais un "physicien" peut, sans être spécialiste de la méthode, construire un Bond-graph à mot. Cela est possible en suivant les différentes phases décrites plus loin et par le tableau de la figure 1.18 qui regroupe les variables effort (e) et flux (f) avec la notation usuellement employée en physique. La puissance dans un lien est toujours le produit de 2 variables :

$$P = e \cdot f$$

DOMAINE	SYMBOLE	EFFORT	FLUX
ELECTRIQUE	e	u	i
HYDRAULIQUE	h	P	Q
MECANIQUE ROTATION	mr	Γ	ω
MECANIQUE TRANSLATION	mt	F	V

fig 1.18 : Tableau des principales variables physiques.

La démarche conduisant à l'obtention d'un BGM peut se résumer en trois phases :

Phase 1 : Recherche des différentes puissances mises en jeu dans le système à l'aide du tableau des variables physiques. Cette première investigation servira à caractériser les liens de puissance.

Phase 2 : Caractérisation des sous-ensembles physiques intervenant dans le transfert de puissance. La finesse de la décomposition dépend de la précision recherchée pour le modèle.

Phase 3 : Connexion des sous-systèmes par la recherche des variables physiques qui s'égalisent deux à deux dans le sens de la puissance.

Exemple :

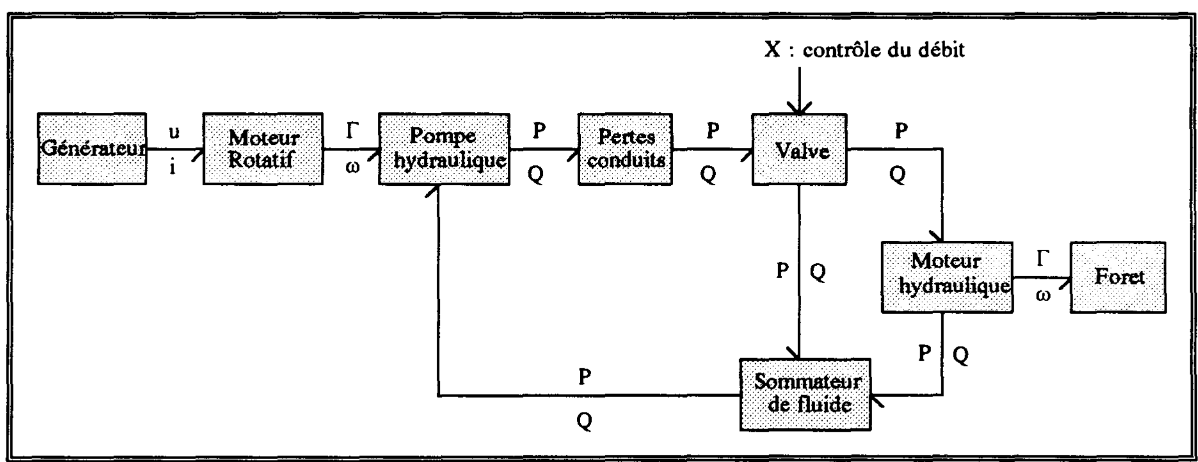


fig 1.19 : Bond-graph à mot de la forreuse hydraulique.

Remarque : pour symboliser les pertes de pression **P** et de débit **Q** dans les conduits du circuit hydraulique, nous avons introduit l'élément "pertes dans les conduits".

I.3.1.4. Comparaison des trois méthodes

Dans la méthode Flow-Model, à partir d'une connaissance précise des processus, on obtient un modèle massique et énergétique utilisant une décomposition topologique du système ainsi qu'une classification des activités du processus. Trop axé sur la physique et la topologie du système, le côté fonctionnel n'a pas été clairement intégré. De plus, un apprentissage des symboles est nécessaire. Cette méthode étant trop restrictive, nous allons nous intéresser aux deux autres méthodes.

La méthode SADT englobe l'aspect informationnel directement dans sa construction. A travers la forme actigramme, les activités peuvent être des transferts de d'énergie (échange, dissipation), mais tout autre phénomène peut-être représenté (création de matière, transport d'informations, activités de contrôle, ...). Une bonne connaissance du processus est demandée (qui est créé, à partir de quoi, selon quel critère, par qui et par quoi).

Le BGM fait apparaître des informations exclusivement sur des transferts de puissance ou d'énergie entre supports physiques. Il repose sur une décomposition du processus physique en un certain nombre de phénomènes physiques, ou de sous-systèmes qui s'échangent de la puissance, et il constitue une représentation de ces échanges. La décomposition du BGM dépend du niveau de décomposition lié aux informations disponibles, et au degré de finesse de la modélisation souhaitée.

A noter que pour le BGM et SADT, les représentations sont identiques pour caractériser ces transferts comme le montre la figure 1.20.

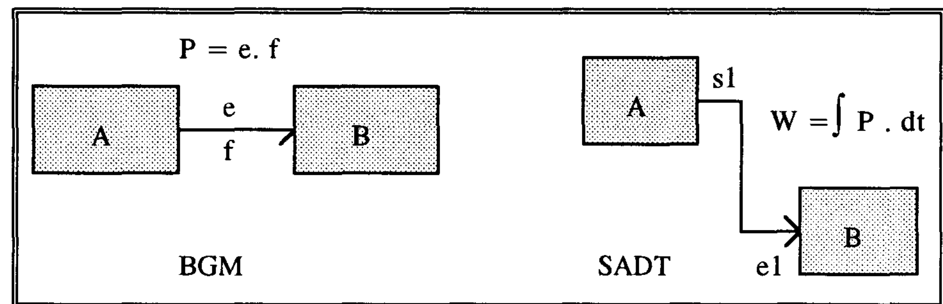


fig 1.20 : Analogie entre le modèle SADT et Bond-graph.

Le BGM présente l'avantage d'être plus synthétique : une représentation SADT donne la possibilité de représenter tout type d'activité dans le système. De plus elle fait apparaître explicitement les variables de contrôle.

Ces deux approches sont complémentaires pour une analyse fonctionnelle (certains aspects sont faits pour l'un ou pour l'autre) et chacune des approches valide l'autre. Les deux représentations constituent donc un bon point de départ pour l'analyse du système.

SADT permet de faire des zooms sur certaines parties du BGM (en décomposant l'action : **A** délivre de la puissance à **B**), et permet une analyse plus fine. On peut ainsi faire un va-et-vient entre BGM et SADT.

I.3.2. Outils méthodologiques pour la modélisation structurelle

I.3.2.1. Graphe biparti [STAROSVIECKI 89]

Le processus est représenté par une structure de blocs interconnectés. A chacun de ces blocs correspond un ensemble de m contraintes : $F = \{f_1, \ldots\ldots\ldots\ldots, f_m\}$
appliquées à un ensemble de n variables : $Y = \{Y_1, \ldots\ldots\ldots, Y_n\}$
où chaque contrainte lie une partie des variables de Y par la relation : $F(Y) = 0$

La structure est décrite par la relation binaire suivante : $S : F.Y \rightarrow \{0,1\}$

$$(f_i, y_j) \rightarrow S(f_i, y_j)$$

tel que $S(f_i,y_j) = 1$ ssi la contrainte f_i est appliquée à la variable y_j, **0** sinon

On obtient ainsi un graphe biparti qui comporte la totalité des variables sur l'axe horizontal et la i-ème ligne de ce graphe est construite à partir de f_i indiquant la dépendance des variables entre elles.

I.3.2.2. Propriétés structurelles par la causalité

La causalité affectée au bond-graph simplifié amène des informations supplémentaires sur les relations entre les éléments constitutifs du modèle bond-graph de ces informations. On en déduit des propriétés structurelles car elles ne dépendent que du type d'éléments qui composent ce système et de la façon dont ils sont interconnectés, mais pas de valeur numérique de leurs paramètres.
Ainsi peut-on, par simple manipulation causale et sans calcul, déterminer :

- le rang structurel de la matrice d'état associé au bond-graph,
- la commandabilité et l'observabilité structurelle,
- le nombre de modes structurellement nuls,
- le type d'actionneurs et de détecteurs (effort ou flux) à mettre autour d'un système, leur nombre minimum et leur positionnement pour que le système soit structurellement commandable et observable ...

Des problèmes numériques peuvent intervenir lorsque le modèle bond-graph présente des éléments dynamiques en causalité dérivée ou des boucles algébriques. La mise en évidence de chemins causaux particuliers permet de détecter à priori ces difficultés.

I.3.3. Outils méthodologiques pour la modélisation comportementale

Le modèle comportemental décrit l'évolution des variables d'état du processus et constitue donc la représentation la plus détaillée du fonctionnement du système. Cela se traduit dans la description par un grand nombre de variables, de fonctions et pour ces dernières une grande diversité de types : qualitatif / quantitatif, linéaire / non-linéaire, statique / dynamique ...

Remarque : les différents modèles comportementaux, définis au paragraphe 1 peuvent être utilisés.

Bond-graph causal

Le modèle bond-graph d'un système physique dynamique se situe comme intermédiaire entre le schéma physique et les modèles mathématiques associés (équation d'état, fonction ou matrice de transfert). Le bond-graph montre non seulement l'architecture du système, mais aussi son organisation causale, par la mise en évidence des relations de cause à effet qui interviennent entre les éléments. Cette propriété est fondamentale car elle permet une approche systématique et organisée pour écrire les relations qui caractérisent l'évolution du système et combiner équations différentielles et algébriques.

Exemple :

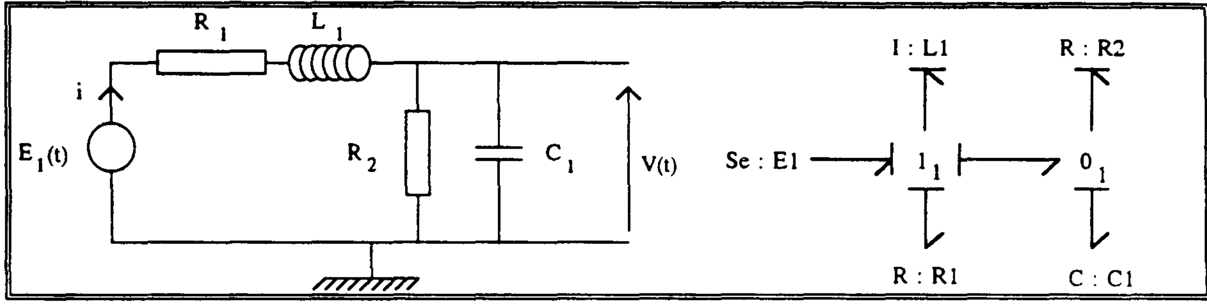


fig 1.21 : Système électrique et son bond-graph associé.

Trois boucles causales sont apparentes sur le bond-graph et chacune d'elle est associé à un gain B :

- entre R_1 et L_1 :

$$R : R_1 \leftarrow\!\!| \; 1_1 \; \rightharpoonup\!| \; I : L_1$$

$$B_1 = - R_1 / L_1 s$$

- entre C_1 et R_2 :

$$C : C_1 \leftarrow\!\!| \; 0_1 \; \rightharpoonup\!| \; R : R_2$$

$$B_2 = - 1 / R_2 C_1 s$$

- entre L_1 et C_1 :

$$I : L_1 \leftarrow 1_1 \rightarrow 0_1 \rightarrow C : C_1$$

$$B_3 = -1 / L_1 C_1 s^2$$

Ces boucles permettent donc de calculer formellement l'équation d'état :

$$\dot{X} = AX + BU \qquad Y = CX + DU$$

à l'aide des variables p et q associées aux éléments I et C par les relations :

$$p = \int e.dt \ , \ q = \int f.dt \ , \ e = \frac{q}{C} \ , \ f = \frac{p}{L}$$

$$\dot{X} = \begin{bmatrix} \dot{p}_I \\ \dot{q}_C \end{bmatrix} = \begin{bmatrix} -\frac{R_1}{L_1} & -\frac{1}{C_1} \\ \frac{1}{L_1} & -\frac{1}{R_2 C} \end{bmatrix} \begin{bmatrix} p_I \\ q_C \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} E \qquad y = V = \begin{bmatrix} O & \frac{1}{C} \end{bmatrix} \begin{bmatrix} p_I \\ q_C \end{bmatrix}$$

Par la règle de Mason, on obtient transmittance entrée / sortie :

$$T(s) = \frac{V(s)}{E(s)} = \frac{\frac{1}{LCs^2}}{1 + \frac{R_1}{Ls} + \frac{1}{R_2 Cs} + \frac{1}{LCs^2} + \frac{R_1}{R_2 LCs^2}}$$

I.3.4. Positionnement du bond-graph dans la démarche de modélisation

Nous avons choisi l'approche bond-graph, car elle ne demande pas généralement l'écriture de lois générales de conservation, même si celles-ci peuvent être mises en évidence. Elle repose essentiellement sur la caractérisation des phénomènes d'échanges de puissance au sein du système. De plus cette démarche possède intrinsèquement les trois formes de modélisation précitées : fonctionnelle, structurelle et comportementale.

Nous synthétisons la démarche par bond-graph en mettant en exergue ces trois formes.

La **première étape** consiste à étudier les fonctions et l'architecture du système, soit l'interconnexion des sous-systèmes, soit le couplage des phénomènes physiques retenus, et à la reproduire graphiquement par des mots nommant les parties mises en évidence et par des flèches caractérisant les échanges de puissance entre ces parties.

On obtient ainsi le bond-graph à mot qui représente l'aspect fonctionnel du système. Ce raisonnement peut être poussé, à chaque partie du système, afin d'obtenir plusieurs niveaux de modélisation. Le dernier niveau est constitué des éléments bond-graph permettant de représenter à travers un langage unique, tous les domaines de la physique.

Dans la **deuxième étape,** on affecte la causalité au bond-graph. Ainsi, les relations de cause à effet, entre les éléments bond-graph et les variables du modèle, peuvent être mises en évidence afin de déduire plusieurs propriétés liées à la structure même du système : aspect structurel .

La **troisième étape** consiste à écrire les lois constitutives des composants à l'aide des chemins causaux qui guident, de façon systématique, l'écriture et l'organisation des équations (aspect comportemental).

Ainsi, est-il possible en suivant la causalité dans le bond-graph, d'obtenir formellement, sans aucun calcul, les équations d'états ou les fonctions de transfert pour la simulation avec des logiciels adaptés.

Conclusion :

Le modèle bond-graph apparaît comme un excellent outil d'aide à l'analyse des systèmes à l'aide de son caractère graphique et sa structure causale. Il fournit directement à l'utilisateur des informations originales, parfois difficiles à obtenir par d'autres voies.

I.4 Le projet ARCHER

Introduit par H.M. PAYNTER [1961], la méthodologie bond-graph a été formalisée par D. KARNOPP et R.C. ROSENBERG [1983], THOMA [75], BREEDVELD [84]. Elle est utilisée aujourd'hui dans le milieu industriel et universitaire.

I.4.1. Logiciels utilisant l'approche bond-graph

De nombreux logiciels utilisant l'approche bond-graph existent actuellement sur le marché ou sont en cours de développement. Pratiquement tous ces logiciels utilisent le modèle bond-graph comme point de départ et leur finalité demeure donc la simulation du système physique étudié.

ARCHER est un pré-processeur qui se situe en amont de l'étape de simulation, il modélise symboliquement des systèmes physiques, les analyse à travers un dialogue avec l'utilisateur, calcule les équations en formelles.

Nous présentons les principaux logiciels utilisant la méthodologie bond-graph.

ENPORT [ROSENBERG 75] :

Ce logiciel est le plus ancien a été développé pour de gros systèmes informatiques par Rosenberg. Le modèle bond-graph est entré sous forme de code. Il affecte les causalités et construit l'équation d'état du système à partir des valeurs numériques des éléments. Il réalise ensuite l'intégration de l'équation et fournit la réponse du système, pour une entrée donnée, sous forme de table de résultats ou de courbe de réponse. Il est commercialisé par ROSENCODE et utilisé notamment par FORD.

TUTSIM :

Il accepte en entrée des bloc-diagrammes (linéaires ou non linéaires), des modèles bond-graphs ou une combinaison des deux et calcule les réponses du modèle à des entrées données. C'est un logiciel convivial et proche des principes de base de la simulation analogique, ce qui en fait un excellent outil pédagogique. L'utilisateur construit le schéma de simulation à partir des équations différentielles issues de l'application des lois de la physique ; on entre le modèle bond-graph correspondant complet (simplifié et avec la causalité) sous forme de code. Il est commercialisé par MEERMAN AUTOMATISERING.

CAMP :

Il est développé pour traduire la représentation du modèle bond-graph d'un système physique en codes FORTRAN utilisables par des logiciels de simulation. Il génère à partir du modèle bond-graph, les équations différentielles du premier ordre (correspondant à l'équation d'état), les variables et crée les fichiers appropriés pour la simulation. Le modèle bond-graph est entré sous CAMP grâce à un code très proche de celui de ENPORT, c'est-à-dire la liste des éléments et numéros du lien auquel ils sont attachés. Il est commercialisé par RAPID DATA.

CAMP G [GRANDA 93] :

C'est un logiciel dérivé de CAMP avec une interface graphique qui permet de décrire le modèle bond-graph. Ce logiciel est développé par José GRANDA.

PROUESSE [BRESCH 86] :

C'est un outil d'étude des systèmes dynamiques. Il se compose de trois modules. Le premier EREL permet d'exploiter la représentation d'état linéaire (simulations temporelles et fréquentielles, calculs des valeurs propres, calculs de la matrice de transfert à partir de la représentation d'état...). Le module SIGE permet de réaliser la simulation des systèmes dynamiques (linéaires ou non linéaires). Le dernier module GRAPHEUR permet de présenter les résultats graphiques de ces simulations.

Trois autres modules sont prévus dans les versions futures. GESIG permettra de générer des signaux d'excitation pour la simulation des systèmes modélisés. EDM sera un éditeur de matrice et EDBG un éditeur de modèles bond-graphs.

CAMAS [BROENINK 90] :

C'est un processeur qui accepte en entrée à la fois des modèles bond-graphs et des relations constitutives. Il est considéré par son auteur comme un langage de construction de modèles bond-graphs.

I.4.2. ARCHER

ARCHER est un logiciel d'aide à la modélisation et à l'analyse des systèmes dynamiques. Il utilise la méthodologie bond-graph, les techniques de l'intelligence artificielle et possède quelques concepts de la programmation orientée objet.

Comme présenté par [MONTBRUN & AL 91], beaucoup de programmes basé sur la technique du bond-graph ont été développés depuis le premier programme ENPORT présenté plus haut. Tous ont le même point de départ, ils nécessitent, de la part de l'utilisateur, la connaissance de la méthodologie bond-graph et de construire par lui-même le modèle bond-graph du système étudié.

ARCHER [DAUPHIN & AL 87], n'est pas seulement un programme de simulation. Ses principaux buts sont la modélisation et l'analyse des propriétés structurelles du système étudié. La phase de modélisation est décomposée en plusieurs étapes.

A partir d'une description du système physique, ARCHER construit le bond-graph associé au modèle, assigne la causalité, dessine automatiquement le bond-graph, génère les modèles mathématiques sous des expressions formelles.

La figure 1.22 présente les différents modules du processeur ARCHER. Un programme de menu a en charge l'environnement d'ARCHER et permet d'obtenir certaines informations concernant l'ensemble des données générées par les modules. L'environnement est en perpétuelle évolution car il suit les progrès en matière de convivialité à travers certains logiciels comme Windows associés à des outils de développement comme Turbo Pascal Windows.

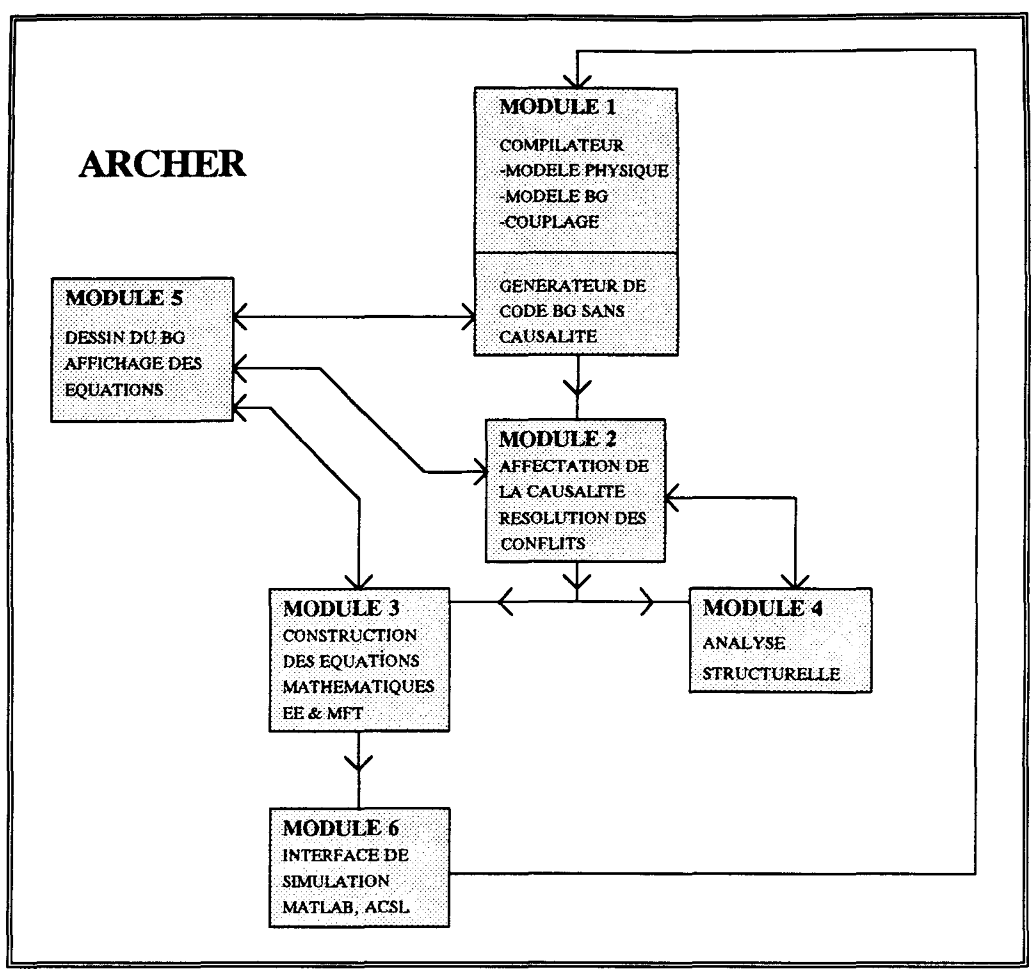


fig 1.22 : Cascade des différents modules utilisés dans ARCHER.

Module 1 : [REMY 92]

Cette étape est importante car elle fournit les données nécessaires au fonctionnement d'ARCHER. Celles-ci peuvent être générées de différentes façons :

cas 1 : L'utilisateur a construit son modèle bond-graph à partir du modèle physique qu'il veut étudier. Dans ce cas il utilise l'éditeur bond-graph et entre son modèle sous la forme d'un texte, avec une syntaxe adaptée..

cas 2 : Le modèle physique est descriptif et homogène, l'utilisateur peut traduire la topologie du système à travers un **L**angage simple utilisant les notions de **S**érie et **P**arallèle (**LSP**) associées à des parcours de circuits issus de la loi des noeuds et des mailles.

cas 3 : Le modèle est complexe et fait intervenir plusieurs domaine de la physique (modèle descriptif hétérogène), les notions de **B**locs et **N**oeuds sont alors utilisées à travers un autre **L**angage (**LBN**) capable de traduire la topologie du système par combinaisons de ces deux entités.

Dans les deux premiers cas, les données générées sont des données bond-graphs (liens-jonctions-éléments). Dans le troisième cas, ARCHER gère les modèles des blocs définis, qu'il s'agisse de bond-graph ou de modèles mathématiques. Le compilateur associé au langage possède la structure classique :

- Un analyseur syntaxique basé sur un langage très souple et proche du naturel.
- Un analyseur sémantique capable de détecter un certain nombres de contradictions ou d'anomalies.
- Un générateur de code bond-graph sans causalité.
- Un gestionnaire d'une base de données formant une bibliothèque de blocs de différents types.

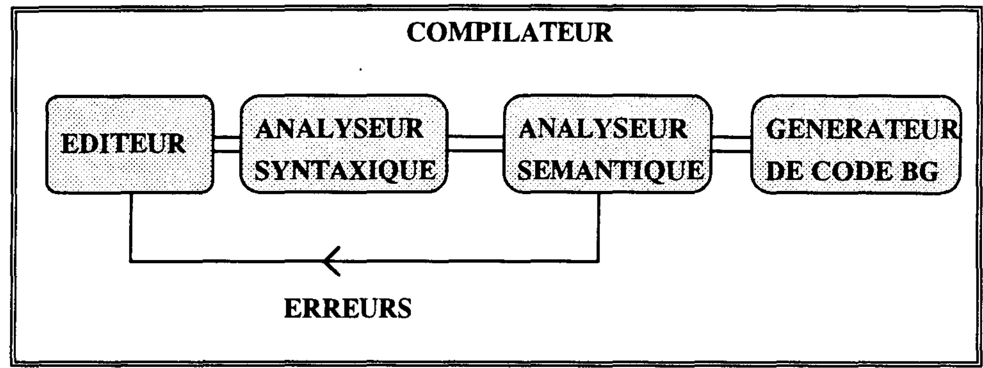


fig 1.23 : Schéma type d'un compilateur.

Module 2 : [AZMANI 90]

Il a pour objet d'affecter la causalité aux éléments bond-graphs à partir d'une représentation symbolique sous forme d'arbres construits par l'intermédiaire de vecteurs d'incidences extraits du modèle bond-graph.

Cette méthode d'affectation de la causalité provoque, en fonction de l'architecture du bond-graph, des conflits causaux qui correspondent à des situations non déterministes. Leur résolution se fait par les techniques de l'intelligence artificielle, couplées à un dialogue avec l'utilisateur lorsque le conflit n'est pas solvable sans une modification topologique du bond-graph (ajout d'éléments capacitifs, inductifs, ou résistifs).

Les données générées par ce module apparaissent sous la forme d'une base de faits nécessaires à toutes les méthodes basées sur la théorie des bonds-graphs.

Module 3 : [AZMANI 90]

A partir du bond-graph causal, on détermine les informations causales nécessaires aux calculs des équations mathématiques. Les entrées et sorties du modèle sont sélectionnées par l'intermédiaire d'une interface utilisateur qui offre la possibilité de choisir certaines grandeurs physiques associées aux éléments.

Par exemple pour un système électrique, nous pouvons choisir parmi la tension ou le courant, mais aussi le flux dans une bobine ou la charge dans un condensateur.

Ces données permettent de déterminer les matrices et fonctions de transfert (M.F.T) dans le cas linéaire et les équations d'états (E.E) du système linéaire (ou partiellement non linéaire) sous forme d'expressions formelles. Cela est permis par un générateur de calcul formel développé spécialement pour ARCHER. (L'utilisation d'un logiciel comme Mathématica aurait demandé un interfaçage lourd qui aurait considérablement ralenti le calcul des équations ; mais l'emploi d'un tel logiciel spécialisé dans le calcul formel est à l'étude).

Module 4 : [AZMANI 91] [SUEUR 90] [RAHMANI 93]

Il utilise la base de connaissance causale à savoir, la causalité dérivée, les boucles algébriques, les boucles causales ... Toutes ces informations contribuent à l'analyse structurelle du système par l'implantation des méthodes d'analyses structurelles telles que la commandabilité ou l'observabilité.

Module 5 : [BOUAYAD 91]

C'est une interface graphique qui intervient à plusieurs étapes d'ARCHER.

- Pour le dessin du bond-graph acausal (Module 1) et causal (Module 2) à partir d'une base de faits. Cette opération de traçage nécessite la conversion du bond-graph en graphe simple qui, après traitement (tris et réécritures) combinés à des contraintes esthétiques, donne le dessin du bond-graph à l'écran. Cette interface sert aussi à visualiser les conflits lors du dialogue utilisateur.

- Pour l'affichage des équations mathématiques (Module 3) par une traduction graphique des représentations formelles des équations d'états et de transferts.

Module 6 :

ARCHER se définit comme un pré-processeur de modélisation et d'analyse de modèle physique. C'est pourquoi nous avons cherché à l'associer à des logiciels de simulation existant dans le commerce comme MATLAB.

Cette interface consiste à traduire les données formelles issues du calcul des équations mathématiques dans le formalisme MATLAB. Par l'intermédiaire d'un dialogue, l'utilisateur initialise les différents paramètres physiques,
MATLAB est ensuite lancé automatiquement [KUBIAK 92].

Une interface avec ACSL et MATHEMATICA, basé sur le même principe, est en cours de développement.

I.5. Conclusion

Nous avons présenté dans ce chapitre différentes méthodes de modélisation. Nous avons opté pour la méthodologie bond-graph qui englobe intrinsèquement les trois formes de modélisation à savoir structurelle, fonctionnelle et comportementale.

La modélisation fonctionnelle et structurelle intervient lors de la construction du bond-graph avec un jeu de quelques fonctions génériques (R, C, I, Se, Sf) à travers une structure de jonction. Les liens comportementaux sont représentés par la causalité qui traduit les relations de cause à effet entre les différentes composantes du bond-graph.

Mais avant de construire le bond-graph il nous fallait une méthode capable de décrire graphiquement un système physique d'une manière structurée.

La notion de bond-graph à mot (BGM) nous a offert cette démarche et tout au long de ce mémoire, nous allons voir comment nous pouvons l'associer à la notion de couplage, afin de l'exploiter d'une manière informatique en l'adaptant à la philosophie d'ARCHER.

La position d'ARCHER vis-à-vis des logiciels que nous avons cités, utilisant l'outil Bond-Graph, est originale pour plusieurs raisons :

- il utilise les techniques de l'Intelligence Artificielle (Turbo Prolog, programmation orientée objet).

- la connaissance de l'outil Bond-Graph n'est pas une obligation puisque son utilisation peut être tout à fait transparente, en particulier pour les systèmes simples.

- il permet le couplage de modèles appartenant à des domaines différents de la physique pouvant être stockés sous forme de blocs (bond-graph ou non) dans une base de données.

- les règles d'expertise, sous forme d'heuristique, ont été introduites dans certains modules, en particulier concernant l'affectation des causalités et le règlement de certains conflits.

- les équations mathématiques du système sont calculées et représentées d'une manière formelle.

- à partir de ces équations, une simulation peut être effectuée par un logiciel spécialisé (MATLAB) et un interfaçage adapté.

- le dessin du modèle bond-graph se construit automatiquement.

CHAPITRE II

DESCRIPTION DE MODELES SOUS ARCHER

DESCRIPTION DE MODELES SOUS ARCHER

II.1. Introduction

Dans ce chapitre, nous présentons le module de couplage d'ARCHER capable de générer un bond-graph à partir de la description d'un modèle physique, composé de différentes parties appartenant à des domaines différents de la physique.

Nous allons progressivement cerner le problème d'un langage texte de description en commençant d'abord, par nous poser quelques questions intuitives sur l'organisation toplogique d'une telle description.

Ensuite, nous introduirons les notions de "sous-parties" par l'application de la réticulation sur le système étudié. La structuration des sous-parties nous amènera aux notions des blocs associés à des ports stigmatisant les échanges de puissances.

Suivant le type de bloc et le nombre de ses ports, différents assemblages seront possibles et chacun sera traité à travers un langage adapté. Cette description utilisera, soit les notions de série et parallèle, soit les notions de blocs et noeuds, à travers un langage (texte ou graphique) associé à un compilateur.

Dans le cas où les blocs sont de type bond-graph, il sera possible de les "fusionner" afin d'obtenir un modèle bond-graph global du modèle initial. En effet, le bond-graph est un outil performant qui permet l'obtention d'un modèle unifié, et cela quel que soit le domaine physique auquel le système appartient. Cela permet donc l'association de sous-systèmes appartenant à des domaines différents de la physique (électrique, mécanique, hydraulique, électronique...).

Cette opération, nommée couplage, sera contrôlée, à plusieurs niveaux par des règles, entre les ports des blocs, pouvant aboutir sur des choix d'associations.

Tout au long du chapitre, nous présentons l'essentiel des principes des méthodes utilisées pour la génération d'un bond-graph à travers quelques exemples.

La résolution informatique, des différents langages proposés, se fera au chapitre III et IV à travers la présentation du complilateur d'ARCHER.

II.2. Problématique

Pour déduire un modèle sous ARCHER, nous avons choisi de concevoir et développer un éditeur texte fiable, avec une syntaxe facilement compréhensible par un nouvel utilisateur. Le langage proposé permet de décrire des modèles physiques simples et des bond-graphs en mode texte à travers un fichier qui pourra être sauvegardé dans une base de données. Cette description est très rapide car elle permet de faire des copies de blocs textes à l'intérieur d'un modèle ou d'aller les chercher dans d'autres modèles. L'aisance d'un tel langage texte rend son utilisation intéressante et souple.

Par la suite, il sera facile de créer un éditeur graphique qui ne fera que traduire les différentes manipulations du dessin dans le langage texte de description. L'utilisateur pourra intervenir à plusieurs niveaux de création et modifier à son gré le modèle texte.

La figure 2.1 synthétise les différentes étapes nécessaires à la description d'un modèle sous ARCHER.

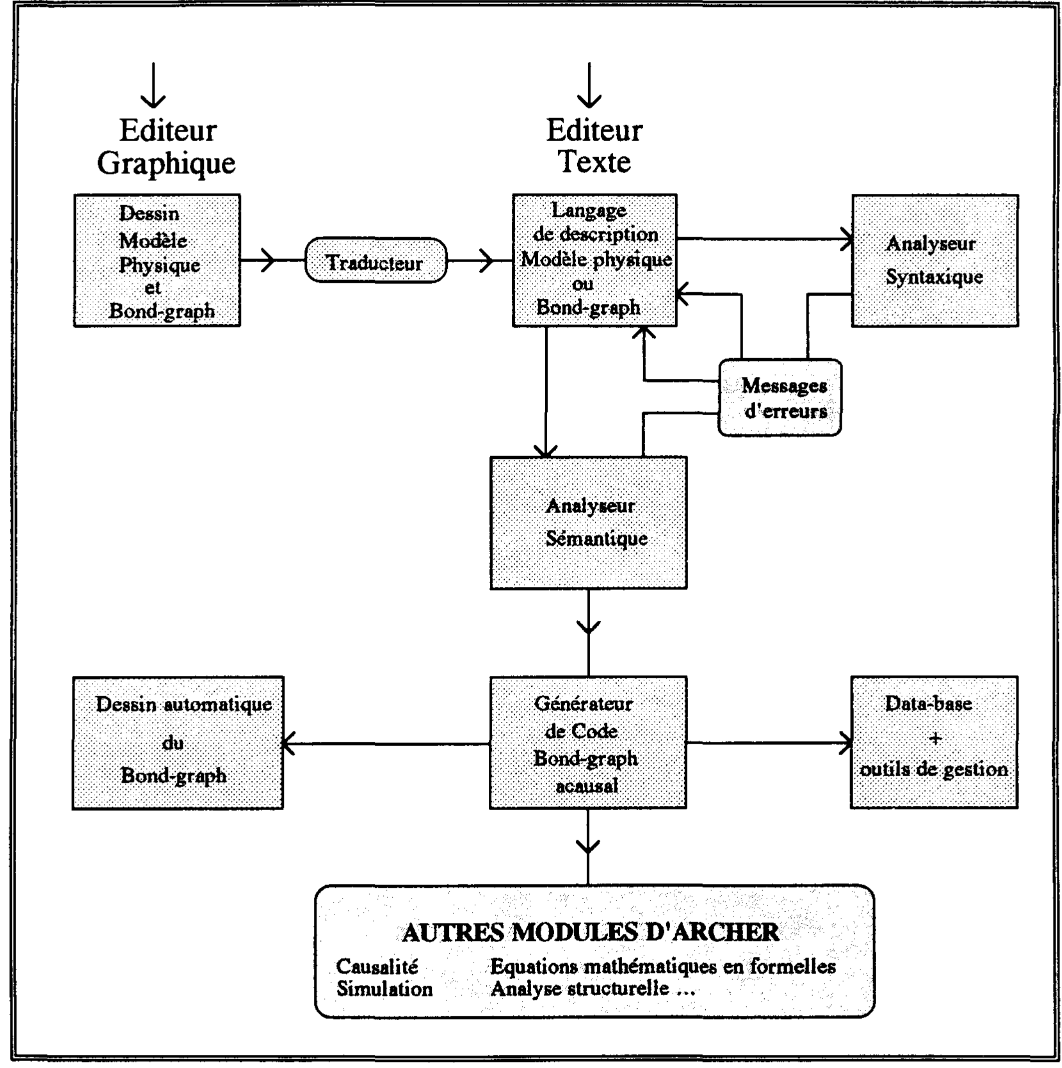


fig 2.1 : Architecture générale d'une description de modèle avec génération de code bond-graph sous ARCHER.

Dans cette structure, on reconnaît les éléments génériques composant un compilateur. Mais avant de détailler son fonctionnement, nous allons définir le problème posé et les difficultés rencontrées, soit :

Trouver un langage topologique pour la description de modèle physique sous un éditeur texte.

II.2.1. Aspects topologiques

Dans la méthode de modélisation utilisée dans cette thèse, le modèle est étudié du point de vue macroscopique suivant la théorie des systèmes physiques.

On sépare les propriétés du système les unes des autres, afin de faire apparaître des sous-systèmes qui sont interconnectés par des connexions où la puissance s'échange. Cette procédure est le concept de réticulation [PAYNTER 61] & [KRON 63].

La réticulation peut être appliquée aux sous-systèmes jusqu'à atteindre des propriétés élémentaires qui ne peuvent être subdivisées.

Lorsque le sous-système est sorti de son environnement (système), nous utiliserons le terme de sous-partie qui donne une autonomie propre au sous-système et prépare la définition du bloc abordé plus loin.

II.2.1.1. Description descendante

Considérons le système S. La mise en évidence des sous-parties dans S peut se faire en identifiant les domaines énergétiques mis en oeuvre dans le système. Plusieurs cas "d'assemblages" graphiques peuvent se présenter ;

- Le système S est composé de sous-parties disposées en série :

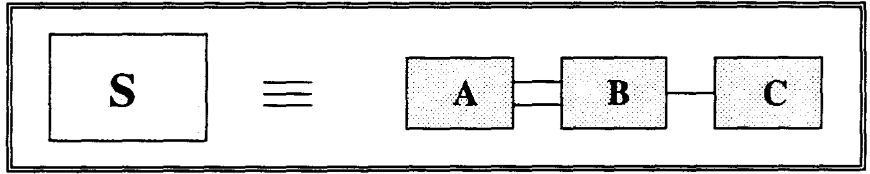


fig 2.2 Les sous-parties A, B et C sont en série.

- Les sous-parties sont en parallèles :

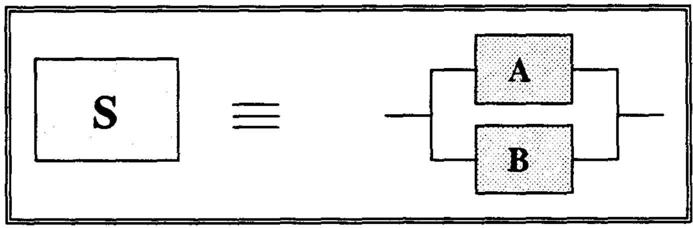


fig 2.3 : Les sous-parties A et B sont en parallèle.

- Le plus souvent, on retrouve une combinaison de ces 2 cas :

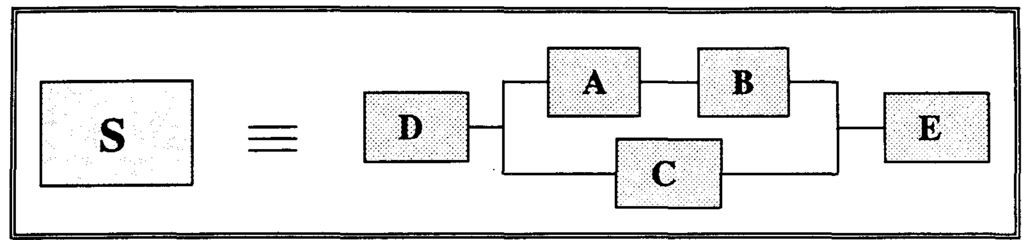


fig 2.4 : La sous-partie C est en parallèle avec A et B en série ; le tout est en série à gauche avec D et à droite avec E.

- Dans le cas général on peut trouver une disposition hétéroclite dans laquelle il peut être difficile de mettre en évidence les sous-parties en série et en parallèle. La description d'un tel schéma consiste à faire état des sous-parties liées entre elles en évitant les redondances inutiles.

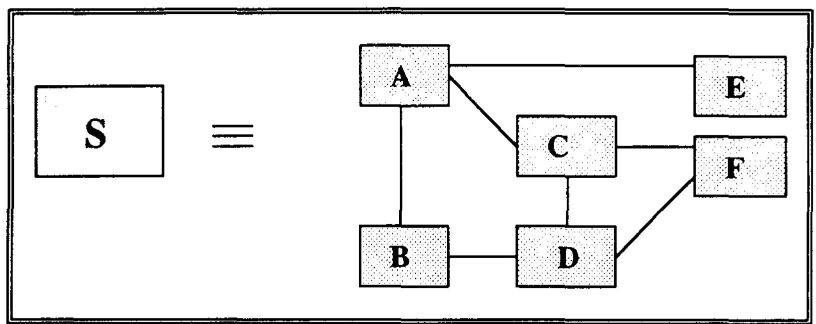


fig 2.5 : A est lié à E, C et B ; C est lié avec F et D , enfin D est lié à F et B.

Après cette brève présentation des différentes structures topologiques pouvant intervenir dans une décomposition fonctionnelle du système, nous allons envisager le contenu des sous-parties.

II.2.1.2. Type de modèles des sous-parties

Dans un système physique, les différentes sous-parties ne sont pas toujours modélisées de la même façon.

En effet, comme nous l'avons vu au chapitre I, les méthodes de modélisation que l'on peut rencontrer conduisent aux types de modèles suivants :

Description Physique (D.PHY),
Fonctions et Matrices de Transfert (F.M.T),
Equations d'Etats (E.E),
Equations Différentielles (E.D),
Graphe de Fluence (G.F),
Bond-Graph (B.G),
Macro d'un Logiciel de Simulation (M.L.S) ...
Modèle Fichier (F),
Ensemble de Règles(E.R), ...

Le problème d'une description mixte est délicat. Cependant, nous verrons que certaines combinaisons sont plus faciles à obtenir que d'autres et se prêtent bien au couplage.

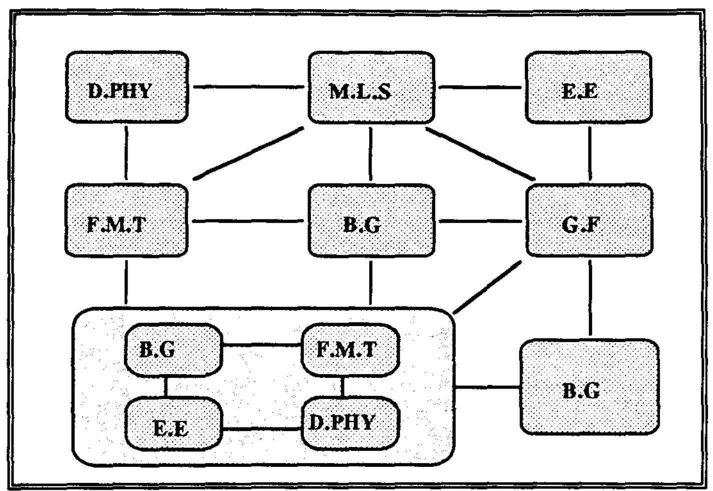


fig 2.6 : Système formé de sous-parties appartenant à différents types de modèles.

Les sous-parties échangent de la puissance par l'intermédiaire de connexions qui peuvent traduire une réalité physique "palpable" (ex : arbre d'un moteur en rotation, générateur de tension ...), ou être l'abstraction d'une fonction physique qui ne peut être caractérisée ponctuellement par un lien physique (ex : frottements sur une masse, pertes dans un conduit hydraulique ...). La notion d'échange de puissance n'apparaît de façon évidente que dans l'approche bond-graph.

Cette puissance est le produit de deux variables : effort (e) et le flux (f). Les connexions sont alors des lieux virtuels ou physiques dans lesquels les variables s'égalisent.

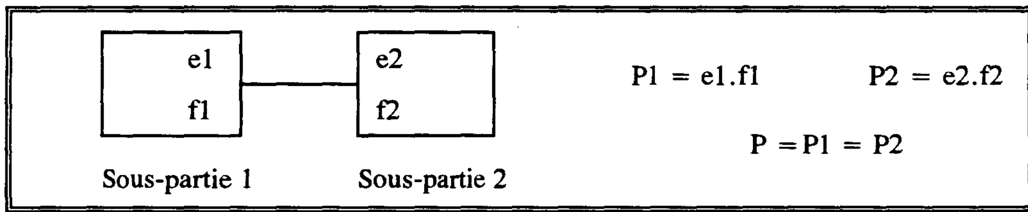


fig 2.7 : Les sous-parties 1 et 2 échangent une puissance P.

On a deux cas de figures pour la connexion :

1. Le couplage est direct, et les variables effort et flux s'égalisent :

$$e1 = e2 \quad \text{et} \quad f1 = f2$$

2. Le couplage se fait par l'intermédiaire d'un transducteur, conservatif de puissance, et donc sans pertes. Les variables effort et flux sont alors liées entre elles par des relations du type :

$e1 = m(.)\, e2$ et $f2 = m(.)\, f1$ (TF ou MTF en bond-graph)

$e1 = r(.)\, f2$ et $e2 = r(.)\, f1$ (GY ou MGY).

Lorsque d'autres types de modèles interviennent (mathématiques, graphiques...) l'échange de puissance se transforme en échanges d'informations (une seule variable est véhiculée)

II.2.2. Couplage des blocs bond-graph

Les différentes sous-parties véhiculent entre elles de la puissance. Il faut donc se définir des points d'attaches, physiques ou virtuelles que nous nommons ports, pour traduire ces échanges.

II.2.2.1. Le port

Un port est en entrée (*in*) ou en sortie (*out*), suivant le sens de la puissance (P = e.f) transmise.(*)

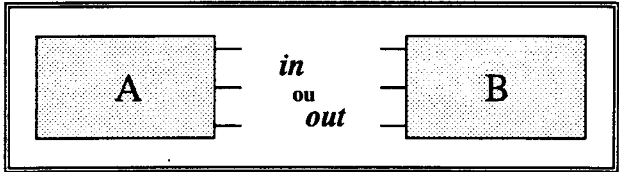


fig 2.8 : L'échange de puissance entre les sous-parties A et B se fait à travers des ports.

Les sous-parties peuvent être modélisées de différentes façons (voir plus haut), mais elles peuvent aussi appartenir à des domaines différents de la physique (électrique, mécanique, hydraulique...).

Lorsque l'on a identifié les sous-parties, les domaines physiques associés et les ports, le problème qui se pose est de choisir la bonne connexion qui servira à les **"Coupler"**.

Dans l'exemple du moteur de la figure 2.9, la liaison se fera par un transducteur pour transformer l'énergie électrique en énergie mécanique, en l'occurrence un gyrateur dont la loi est décrite par la figure 2.10.

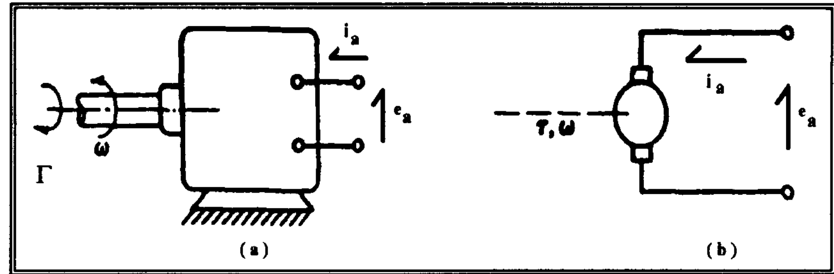


fig 2.9 : La puissance électrique sortant du moteur est transformée en puissance mécanique par un transducteur vers l'arbre de rotation.

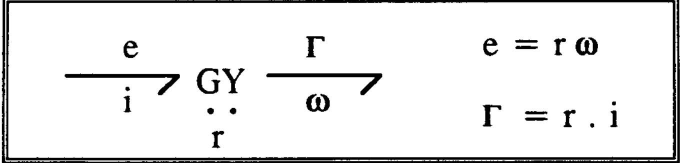


fig 2.10 : Loi du gyrateur.

(*) Ne pas confondre avec la causalité qui traduit le sens dans lequel l'effort et le flux sont connus dans un bond-graph.

Un port est toujours accompagné d'une information sur la nature (électrique, mécanique, hydraulique...) des variables qu'il véhicule.

Pour cela, on utilise le tableau qui regroupe les variables les plus utilisées en physique (figure 1.18). De plus, pour éviter toute ambiguïté, on numérote les ports.

Exemple :

La figure suivante montre une sous partie de **type** bond-graph avec 3 ports de **natures** différentes : électrique (e), hydraulique (h), mécanique (m).

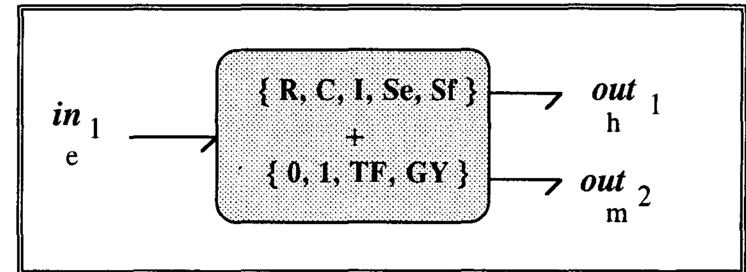


fig 2.11 : Sous-partie appartenant au type bond-graph avec 3 ports :
***in1* de nature électrique (e), *out1* de nature hydraulique (h) et**
***out2* de nature mécanique (m).**

II.2.2.2. Le bloc

Définitions

Un bloc est une sous-partie distincte d'un système, représenté par un type de modèle et qui possède des ports numérotés et identifiés par la nature du domaine physique associé.

Bloc = Sous-partie + Type de modèle + Ports (entrée ou sortie) + Natures physiques.

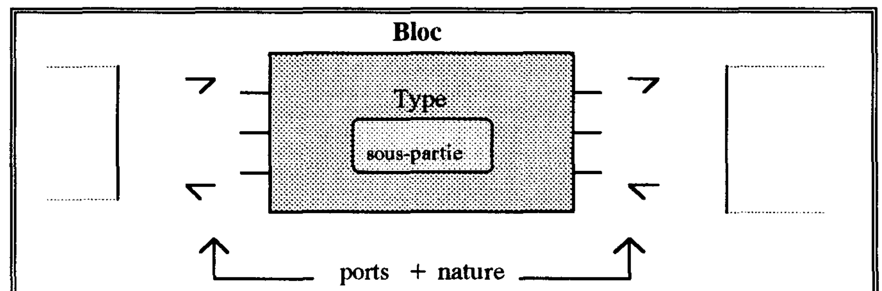


fig 2.12 : Architecture d'un bloc.

Il est possible de réitérer la démarche sur les blocs, jusqu'à obtenir des modèles simples qui peuvent être stockés pour une utilisation ultérieure.

La démarche inverse peut se faire. En effet, on peut se définir une bibliothèque de blocs prédéfinis pour construire un modèle complexe selon la démarche de l'utilisateur.

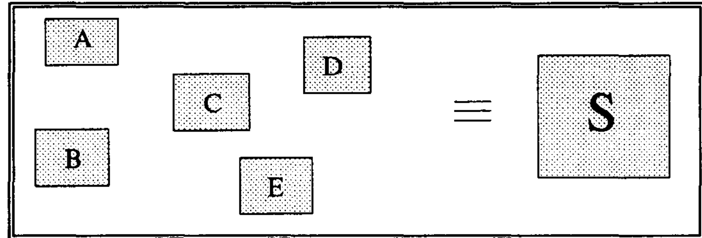


fig 2.13 : Les blocs A, B, C et D sont assemblés pour former un système original.

II.2.3. Cahier des charges du couplage de blocs bond-graphs

Maintenant, nous allons souligner les contraintes et les cas de figures qui pourront être rencontrés à travers un cahier des charges.

II.2.3.1. Comment assembler les blocs ?

Selon le nombre de ports associés aux blocs, deux approches sont possibles.

La première, valable pour les éléments simples, concerne les blocs à une entrée (***in***) et une sortie (***out***). Dans ce cas on a des dipôles (2-ports) qui par association forment un réseau maillé propice à une description série et parallèle comme le montre la figure 2.14.

La seconde, concerne le bloc à plusieurs ports. Dans ce cas les notions de série et de parallèle ne sont plus utilisables. L'idée est de décrire le système autour d'un noeud (N) tout en gardant la structure du réseau formé de blocs (figure 2.15). Cette façon fonctionne a fortiori avec les blocs dipôles. (voir figure 2.16).

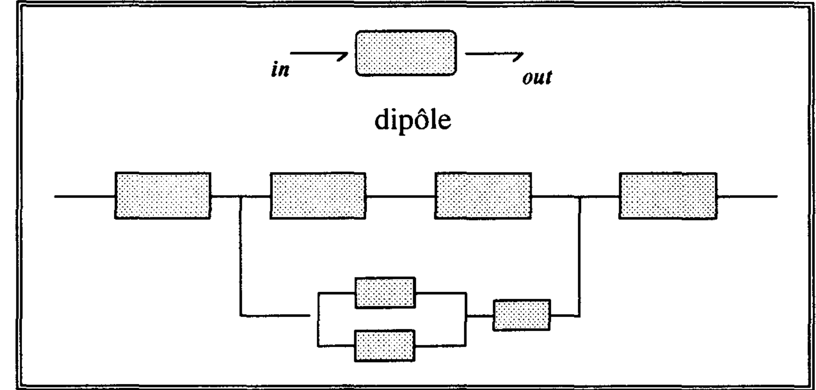


fig 2.14 : Systèmes composés de blocs à une entrée et une sortie avec une description série et parallèle

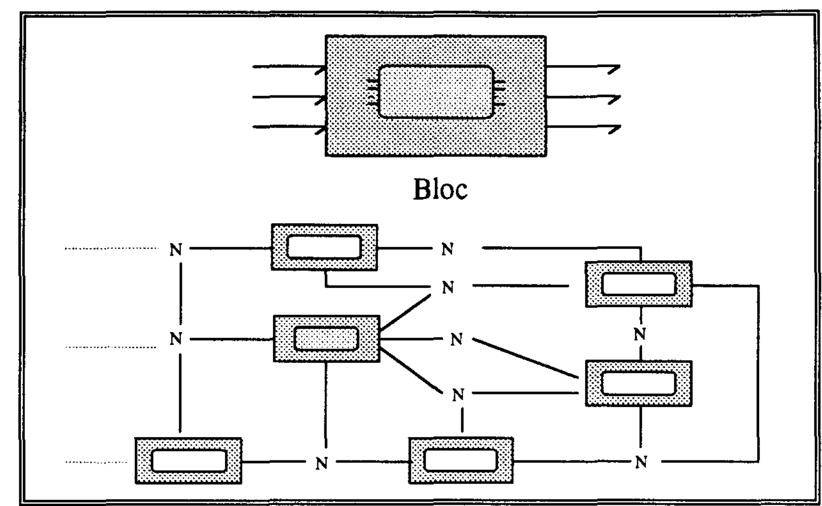


fig 2.15 : Système composé de blocs à plusieurs entrées et sorties, reliés entre eux par des noeuds.

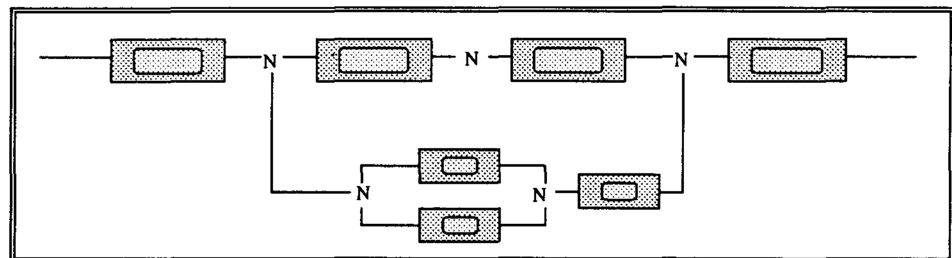


fig 2.16 : Système composé de blocs à une entrée et une sortie avec une description blocs et noeuds.

Le noeud permet une plus grande liberté dans la combinaison des blocs, de plus il permet une simplicité dans la description. Pour les blocs possédant plusieurs entrées et sorties, il faudra donc indiquer quel port (***in*** ou ***out***) sera attaché au noeud. (figure 2.17)

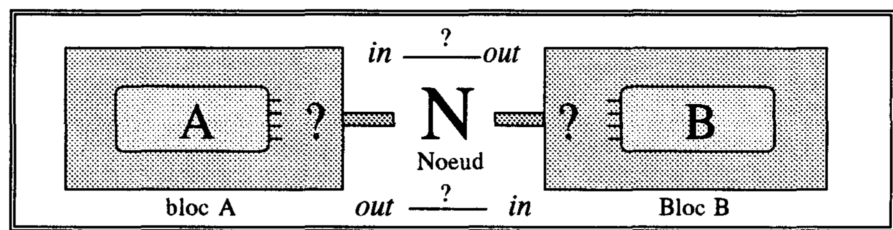


fig 2.17 : Le noeud N relie un des ports des blocs A à son antagoniste du bloc B.

Remarque : Pour l'instant nous faisons abstraction de la signification physique des connexions entre ports (***in***) et (***out***) ; seul l'aspect topologique des blocs nous intéresse pour "prévoir" les contraintes à surmonter d'un point de vue informatique.

D'une manière générale, **l'assemblage** des blocs peut se faire en 2 étapes :

La première étape est une description à partir des blocs et des noeuds du schéma représentant le système étudié (sans indiquer les ports attachés au noeud courant).

La deuxième étape consiste à choisir pour chaque bloc associé à un noeud, le port qui lui sera attaché. Nous développerons cette idée par la suite.

A. Système formé exclusivement de blocs à une entrée et à une sortie

Dans le cas le plus simple, tous les ports sont de même nature et les blocs appartiennent au même domaine physique. Dans ce cas les notions de série et parallèle sont utilisables ainsi qu'une description par point de connexion. Lorsque les ports sont de natures différentes, le noeud semble mieux adapté car il offre un contrôle dans les connexions des ports.

B. Système formé en majorité de blocs à plusieurs entrées et sorties

Quelle que soit la nature des ports, la description se fera presque exclusivement par l'assemblage de blocs et de noeuds, grâce aux contrôles des connexions. Cependant, lorsqu'une structure formée d'un assemblage de blocs dipôles est mise en évidence, il est possible de combiner la description série et parallèle avec la structure formée par les blocs et les noeuds. Nous appelons cette structure un **pseudo-bloc**.(*)

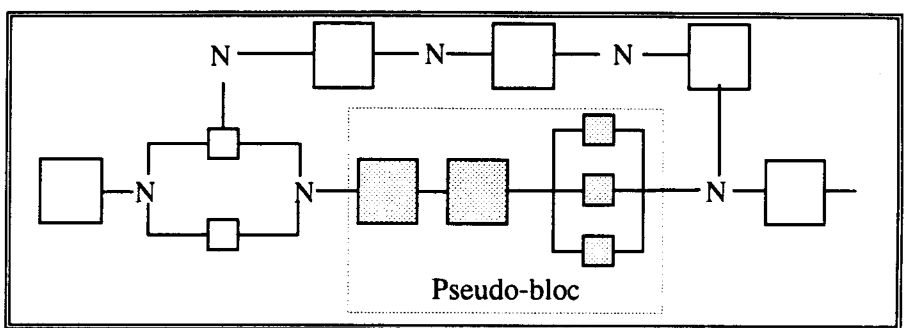


fig 2.18 : Description blocs & noeuds associée à une description série-parallèle.

Remarque : Un bloc avec plus d'un port en entrée ou plus d'un port en sortie n'est pas un dipôle et entre donc dans la deuxième catégorie, celle des blocs à plusieurs entrées et sorties.

La propagation de la puissance entre 2 blocs, se fait suivant un sens défini au préalable. Il est donc normal d'orienter les ports suivant ce sens. Il en découle une **loi d'antagonisme** des ports :

Un port en ***out*** sera toujours relié à un port en ***in*** et réciproquement.

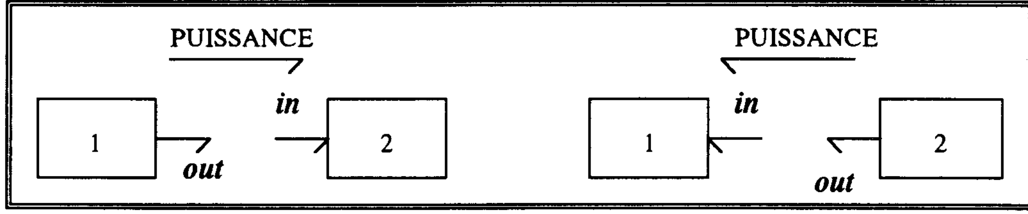


fig 2.19 : Antagonisme des ports.

(*) Voir les pseudo-blocs dans le chapitre IV.

Cas particulier : Plusieurs blocs peuvent être reliés par une sous-partie de type mathématique caractérisant une loi d'échange de puissance suivant la loi des mailles par exemple, comme le montre la figure 2.20 :

Σ des Puissance entrantes = Σ Puissances sortantes.

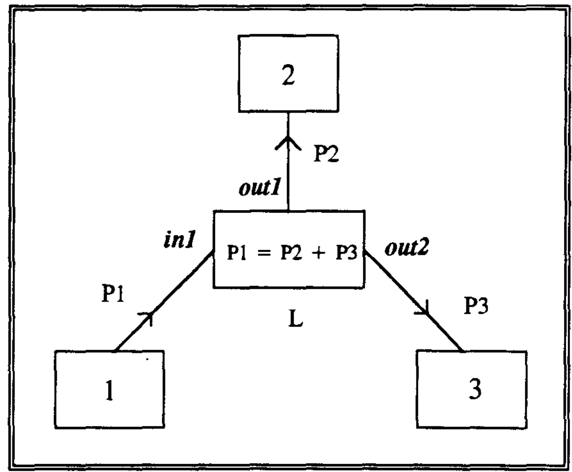


fig 2.20 : La loi L :

$$P_{in1} = P_{out1} + P_{out2}$$

Pour un bloc bond-graph, cette loi peut se traduire, soit par une **jonction 0**, soit par une **jonction 1**. En effet, la puissance est la résultante du produit de l'effort et du flux.

$$e1.f1 = e2.f2 + e3.f3$$

fig 2.21 : Conservation de la puissance selon la relation de la figure 10.

La figure 2.22 montre les deux cas où le bloc est assemblé à au moins 2 autres sous-parties. Les jonctions sont identifiées à des noeuds où la puissance s'échange.

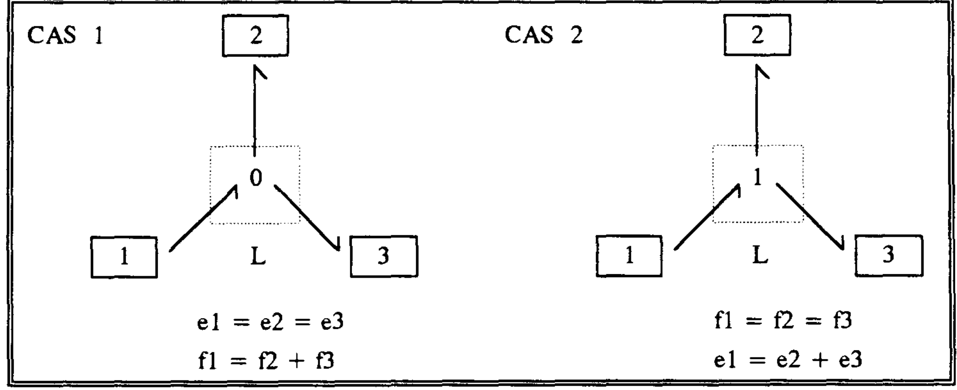


fig 2.22 : Blocs mathématiques avec les bond-graphs associés.

Le tableau de la figure 2.23 donne un résumé des différentes façons d'assembler des blocs à 2 ou plusieurs ports, en associant à chaque cas de figure la description la mieux adaptée.

DESCRIPTION BLOC	NATURE DES PORTS	LANGAGE DE DESCRIPTION		
		SERIE & PARALLELE	BLOC & NOEUD	MELANGE DES DEUX
in A out DIPOLE	*PORTS :* MEMES NATURES *BLOCS :* MEMES DOMAINES PHYSIQUES *MODELE :* HOMOGENE	BIEN ADAPTE	POSSIBLE	POSSIBLE
	PORTS : NATURES DIFFERENTES *BLOCS :* DOMAINES PHYSIQUES DIFFERENTS *MODELE :* HETEROGENE	NON ADAPTE	BIEN ADAPTE	POSSIBLE
in A in out out	*PORTS :* MEMES NATURES *BLOCS :* MEMES DOMAINES PHYSIQUES *MODELE :* HOMOGENE	POSSIBLE NON ADAPTE	BIEN ADAPTE	POSSIBLE EN UTILISANT LE LANGAGE SERIE ET PARALLELE AVEC DES BLOCS DIPOLES = PSEUDO-BLOCS
N N N N N N N	*PORTS :* NATURES DIFFERENTES *BLOCS :* DOMAINES PHYSIQUES DIFFERENTS *MODELE :* HETEROGENE	NON ADAPTE	BIEN ADAPTE	

fig 2.23 : Assemblage des blocs à 2 ou plusieurs ports.

II.2.3.2. Comment générer un modèle global ?

Rappelons que notre objectif est d'obtenir un modèle unique avec lequel on puisse travailler, analyser ou simuler. Cependant, quelle est la forme du modèle la mieux appropriée ?

A. Première étape de la procédure de génération d'un modèle

A partir du système général, on essaie de dégager des sous-systèmes bien distincts avec leurs connexions physiques ou énergétiques. Cette opération peut être réitérée, c'est-à-dire que les sous-systèmes sont décomposés en d'autres sous-systèmes.

A la fin, on obtient des sous-parties plus ou moins complexes qui permettent la constitution de blocs avec l'ajout des ports. Cette phase est parfois facilitée par la topologie du modèle et une méthode description systématique peut être ainsi déduite (ex : système électrique, mécanique, hydraulique...). Voir figure 2.24.

Souvent l'utilisateur doit approfondir le fonctionnement du modèle par ses connaissances physiques (analyse fonctionnelle) pour identifier des sous-parties qui n'ont pas forcément d'existence topologique, mais qui néanmoins participent aux échanges de puissance.

Par exemple le frottement sur une masse, pertes dans un conduit hydraulique, résistance interne d'un générateur de tension...

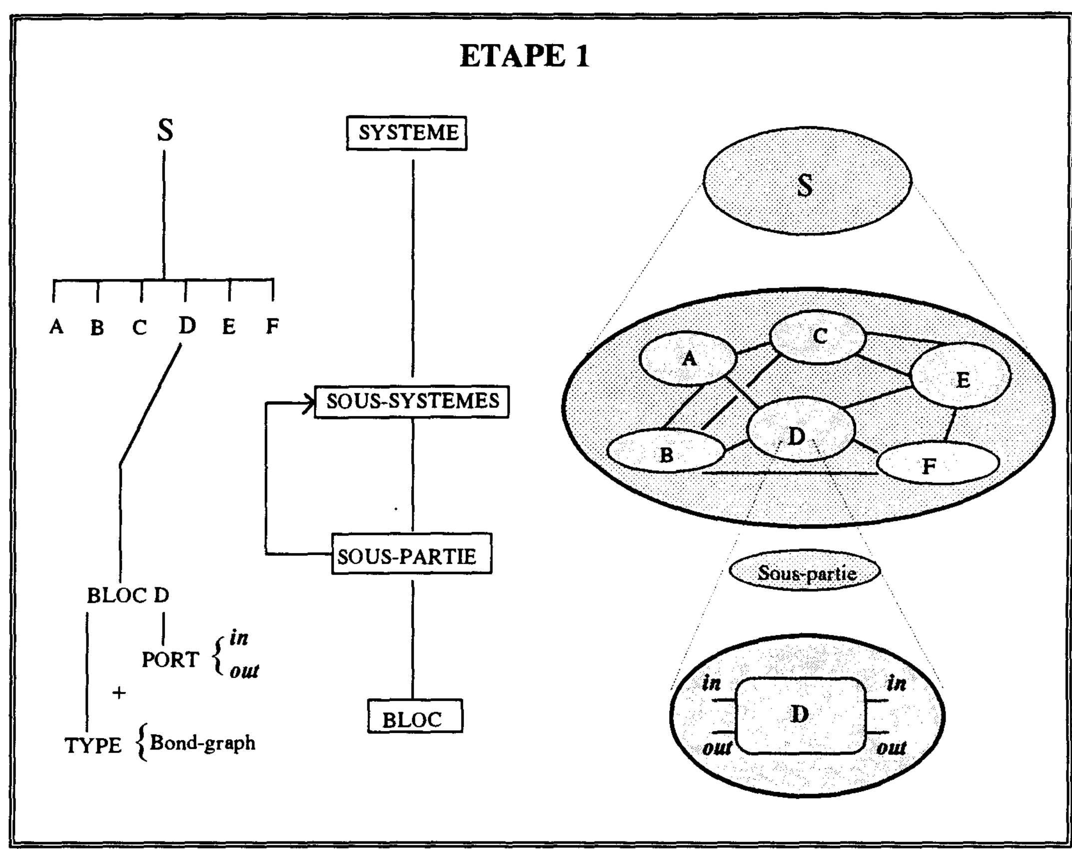


fig 2.24 : Identification des sous parties et de leurs connexions (ports) pour la constitution de blocs.

Nous allons au cours de notre étude, nous intéresser essentiellement à ces trois types de représentation suivant :

- Une description physique (D.PHY) lorsque cela est possible à l'aide d'un langage texte de description.

- Une description en langage texte bond-graph (BG) lorsqu'il est impossible de construire le bloc à l'aide d'une description physique.

- Une forme mathématique (MATH), sous forme de blocs qui ne peuvent être modélisés que par des équations d'état, des fonctions de transferts, des équations différentielles.

Une fois que le bloc a été créé et que l'on a spécifié la nature de ses ports, on le stocke dans une base de données. On se constitue ainsi une librairie de blocs, qui permet de modéliser un autre système, ou inversement créer un système original à partir de ces blocs (bivalence du bloc).

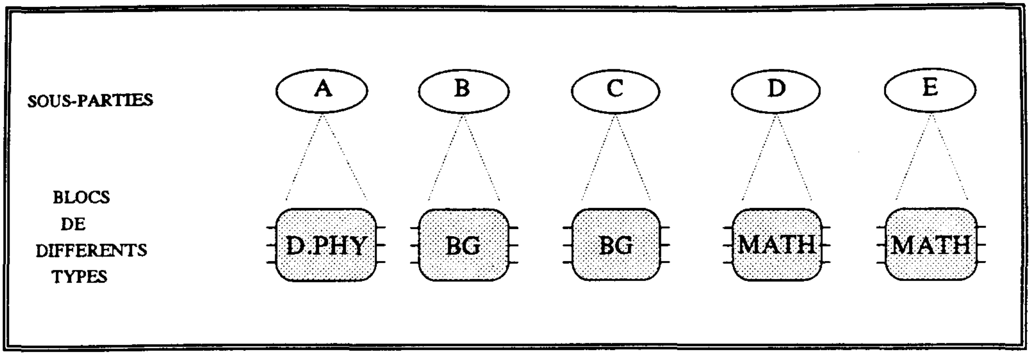


fig 2.25 : Types de modèles associés aux blocs du système de la fig 2.24.

Le couplage peut s'effectuer au niveau bond-graph pour les blocs physiques et bond-graphs. Pour les blocs mathématiques, il faut envisager un traitement différent, soit un couplage au niveau mathématique (une fois le bond-graph traduit en équations), soit un traitement particulier comme il sera présenté au chapitre IV.

B. Deuxième étape de la procédure

Son but est de décrire plus précisément le système étudié formé de blocs définis à la première étape. On utilise alors, pour coupler les blocs (B.G et D.PHY) entre eux, un langage texte de description basé sur les notions de blocs, noeud, série et parallèle [REMY 91].(*)

La procédure d'obtention d'un modèle global est décomposée en 3 niveaux, résumée figure 2.26, et définie de la façon suivante :

Niveau 1 : On génère des blocs bond-graphs acausaux à partir des blocs possédant une description physique.

Niveau 2 : On couple tous les blocs bond-graphs en "sélectionnant" puis en "fusionnant" les ports ***in*** et ***out*** deux à deux, afin de générer un code bond-graph acausal unique. Pour éliminer les jonctions redondantes, celui-ci est simplifié en suivant les règles usuelles de réduction (voir annexe Bond-graph). Un module de dessin permet de visualiser le bond-graph à l'écran [BOUAYAD 91].

Niveau 3 : Les équations mathématiques provenant du bond-graph peuvent être associées aux blocs mathématiques par l'intermédiaire d'un pré-processeur capable de générer un fichier de simulation sous un logiciel tel que MATLAB par exemple.

Remarque : A partir du niveau 2, il est possible de fabriquer des blocs bond-graphs après avoir spécifié les ports ainsi que leurs natures. Cette étape peut se répéter plusieurs fois. Au final nous obtenons un bloc formé de blocs. C'est le concept d'imbrication des blocs.

(*) Ces notions seront développées plus loin. Ici nous présentons la philosophie de la méthode avant tout.

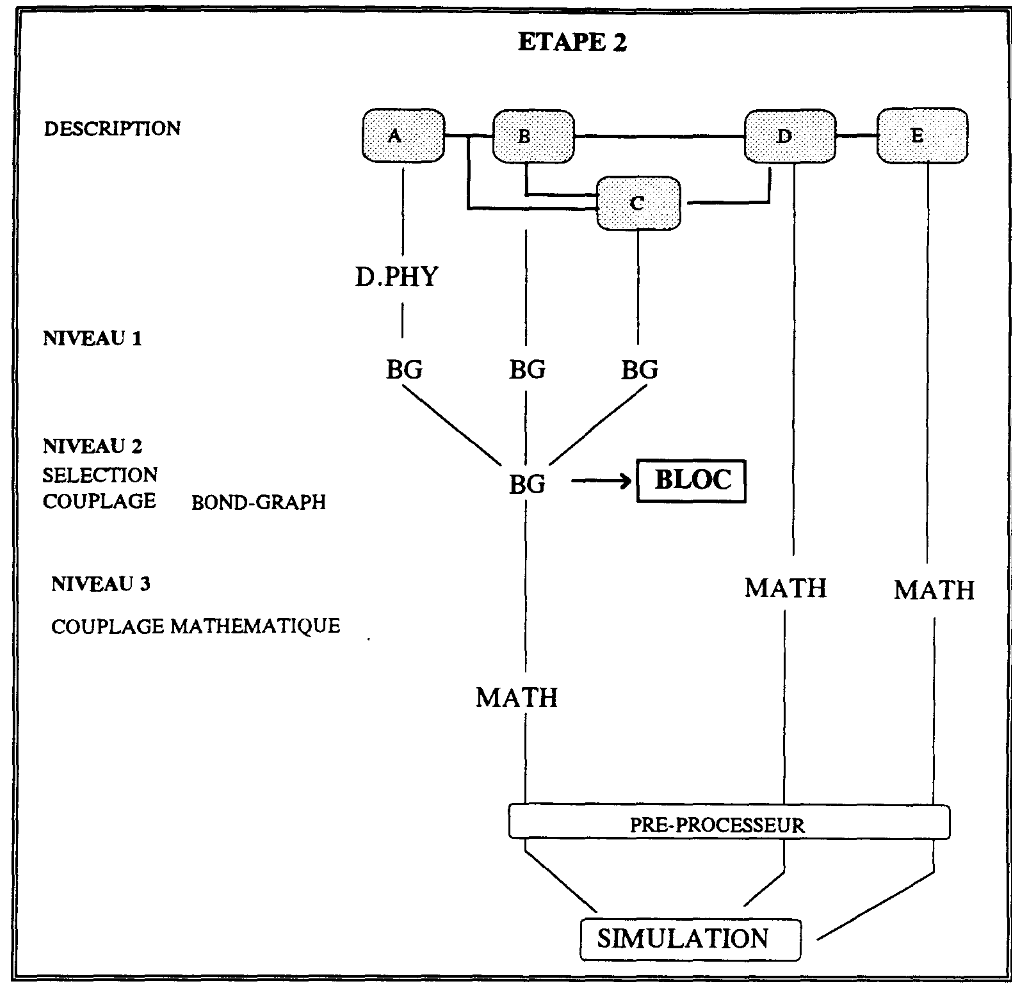


fig 2.26 : Description par bloc du système et couplage à différents niveaux.

II.2.3.3. Quelle convention adopter pour identifier et différencier les éléments des blocs BG ?

Chaque élément physique est identifié par son symbole bond-graph. Les éléments de même nature ou de même fonction sont différenciés par un indice. Si on retrouve le même élément dans plusieurs blocs (figure 2.27), le problème d'une numérotation se pose afin de suivre la provenance de chaque élément du système afin d'éviter toute ambiguïté.

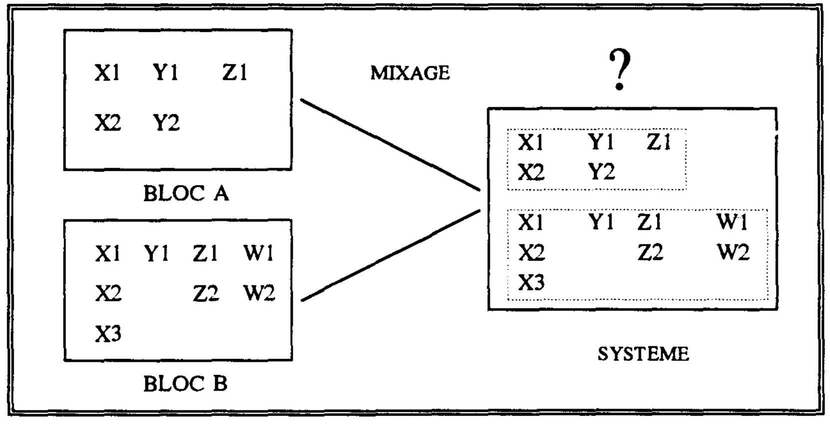


fig 2.27 : Fusion des éléments du bloc A et B

Il y a deux manières d'aborder ce problème :

- Dans la première on indice chaque élément avec le bloc père :

$X1_A$	$Y1_A$	$Z1_A$	$W1_B$
$X2_A$	$Y2_B$	$Z1_B$	$W2_B$
$X1_B$	$Y1_B$	$Z2_B$	
$X2_B$			
$X3_B$			

fig 2.28 : Numérotation par indiçage du bloc père.

Remarque : Un problème peut intervenir, si le nom des blocs possède trop de caractères, car l'indice sera alors trop lourd à gérer pour le module de causalité et le module graphique (affichage d'un bond-graph). En effet ceux-ci travaillent avec un indiçage numérique **N** appliqué à l'élément **E** : **(E_N).** Par exemple : R1, C4, I5.

- Dans la deuxième, la solution choisie est une renumérotation de tous les éléments en gardant une trace de leur descendance dans un fichier que l'on pourra consulter ultérieurement.

X1	Y1	Z1	W1
X2	Y2	Z2	W2
X3	Y3	Z3	
X4			
X5			

≡

X1 → X1 de A	Y1 → Y1 de A	Z1 → Z1 de A	W1 → W1 de B
X2 → X2 de A	Y2 → Y2 de A	Z2 → Z1 de B	W2 → W2 de B
X3 → X1 de B	Y3 → Y1 de B	Z3 → Z2 de B	
X4 → X2 de B			
X5 → X3 de B			

fig 2.29 : Renumérotation des éléments du système avec tableau des descendances.

Informatiquement, la solution que nous avons choisie est une combinaison des 2 propositions :

- Dans un premier temps, chaque bloc bond-graph verra l'indice de ses éléments augmenté par le nom du bloc. Lors du couplage, un brassage d'éléments sera alors possible sans confusion.

- Dans un deuxième temps, on renumérote tous ces éléments, en créant parallèlement le fichier de correspondance à l'aide de la notation intermédiaire.

Cette renumérotation est importante pour les autres modules d'ARCHER. Elle sera utilisée essentiellement au niveau 2 de l'étape 2 de la procédure de génération du modèle global.

II.2.4. Conclusion

La démarche proposée a des points communs avec la représentation par schéma-bloc. Notre but est ici d'approfondir cette description par blocs en la structurant au maximum dans une optique informatique. Le bloc est pensé comme **un objet** à part entière capable :

- de se multiplier,

- de posséder des connexions définies en quantités et natures,

- de s'associer avec d'autres blocs en suivant des règles de couplage pour en former d'autres.

Enfin il s'agit de définir un langage qui supporterait toutes ces contraintes, tout en étant assez souple, pour décrire facilement les différents couplages possibles comme nous l'avons vu dans le tableau de la figure 2.23.

Dans les prochains paragraphes, nous nous intéresserons essentiellement au niveau 1 et 2 de l'étape 2, c'est-à-dire à la fabrication d'un bond-graph à partir d'une description physique et du couplage entre blocs bond-graphs.

Le niveau 3 concernent couplage des blocs bond-graphs avec des blocs mathématiques sera étudié dans le cadre du chapitre IV.

II.3. Description d'un modèle constitué de blocs dipôles

Rappelons que l'un des objectifs d'ARCHER est de donner à des non bond-graphistes le moyen de construire, à travers une description texte, le bond-graph du modèle. Il y a des structures de modèles qui se prêtent mieux que d'autres à une description texte. Elles ont en général des architectures faites d'un assemblage de blocs dipôles issus d'une réticulation du système global. Ce genre de modèle se rencontre essentiellement dans des domaines homogènes.(*)

II.3.1. Notions de bases

II.3.1.1. La notation série et parallèle

Lorsque le bloc possède un seul port en entrée et en sortie, il n'y a pas d'ambiguïté dans la connexion des blocs par la loi des ports antagonistes. Il existe alors des liaisons qui facilitent la production d'une combinaison respectant la topologie structurelle du système étudié : ce sont les liaisons séries et parallèles que l'on représente par les signes "+" et "/" pour faciliter la description :

- La liaison série "+" : La sortie du bloc (*out*) est attaché à l'entrée du bloc suivant (*in*).

- La liaison parallèle "/" : Pour chaque bloc on attache les entrées (*in*) entre elles et les sorties entre elles.

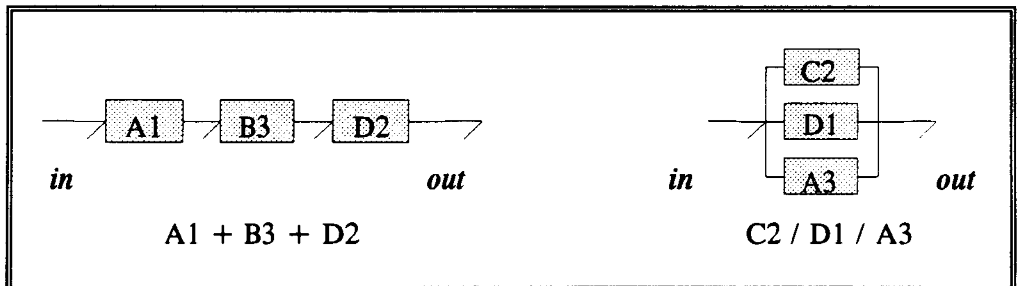


fig 2.30 : Description d'un assemblage série et parallèle.

Remarque : Il ne faut pas confondre avec le concept de THOMA [THOMA 75] qui associe la notion de parallèle **P** à une jonction **0** et la notion de série **S** à une jonction **1**, en conservant la définition électrique pour tous les domaines.

(*) Voir chapitre I.2.3.1.

- Parenthèses "(" ")" :

Elles servent à regrouper les blocs par sous-ensembles auxquels les liaisons + et / peuvent être appliquées.

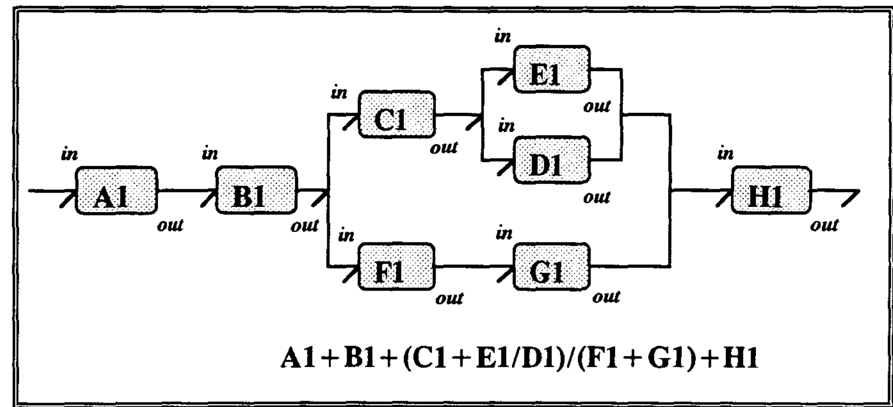


fig 2.31 : Description graphique par un schéma.

Avec ces quatre signes ["+", "/", "(", ")"], on peut décrire la plupart des schémas blocs dipolaires.

II.3.1.2. Le point de connexion

Lorsque la structure est un peu plus complexe et que les signes vus précédemment ne suffisent plus à décrire le système, on introduit le **bloc point** pouvant posséder plusieurs ports indifféremment en entrées ou en sorties en quantités non définies.

Ce point caractérise un échange de puissance par une équation mathématique, suivant la loi de conservation de la puissance.(*)

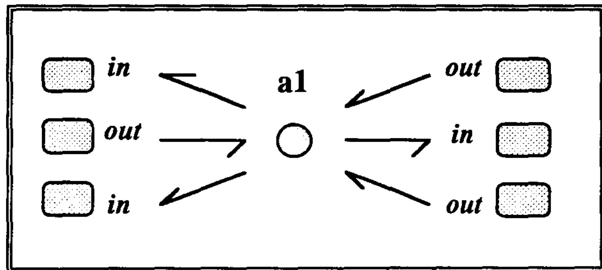


fig 2.32 Point de connexions reliant plusieurs blocs.

Pour ne pas le confondre avec des blocs déjà existants, on le nommera d'une minuscule ['a', ..., 'z'] suivie d'un numéro.

Mais voyons à travers un exemple l'utilité du point et sa souplesse dans la description de schéma bloc dipolaire :

(*) Voir figure 2.21.

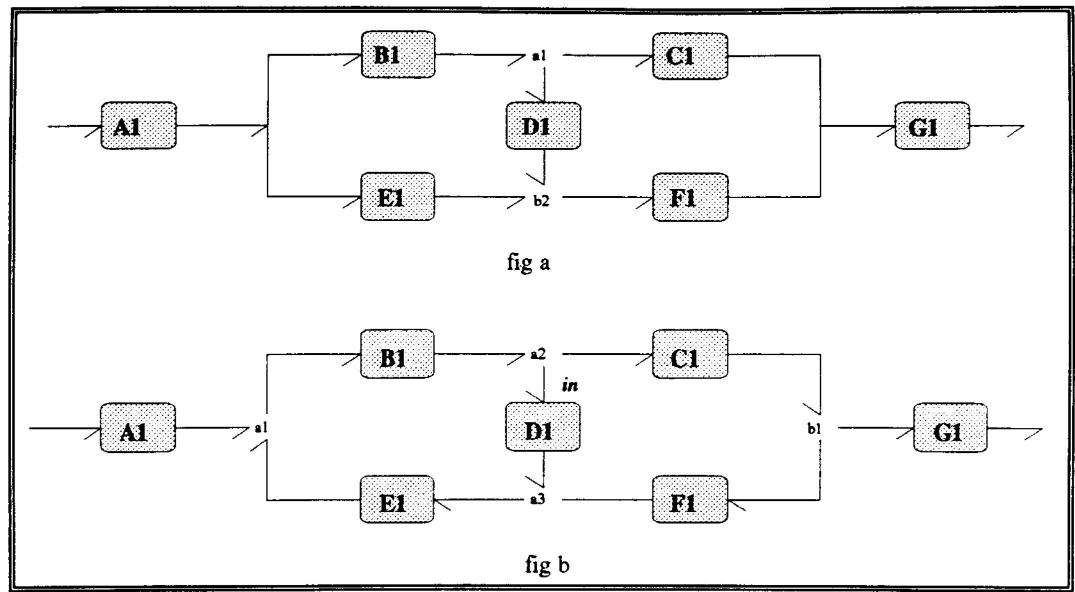


fig 2.33 Importance du sens du parcours des blocs pour la description texte du schéma.

- Lorsque les blocs sont orientés dans le même sens comme dans la figure 2.33a on peut appliquer les liaisons + et / pour avoir la description suivante :

A1 + (B1 + a1 + C1) / (E1 + b2 + F1) + G1

que l'on doit compléter par le bloc **D1** entre les points **a1** et **b2** :

a1 + D1 + b2

- Dans la figure 2.33b, la liaison / n'est plus applicable car **E1** et **F1** ont des sens opposés par rapport à **B1** et **C1**. Il faut donc créer des points supplémentaires pour permettre la description qui se fait en suivant les cycles créés par les blocs et les points dans le sens [(***in***)...(***out***)] ou [(***out***)...(***in***)].

Remarque : La structure de la figure 2.33b peut avoir plusieurs descriptions possibles, comme le montre la figure 2.34.

A1 + a1 + B1 + a2 + C1 + b1 + F1 + a3 + E1 + a1
b1 + G1
a2 + D1 + a3

a3 + E1 + a1 + B1 + a2 + D1 + a3
a2 + C1 + b1 + F1 + a3
b1 + G1
A1 + a1

fig 2.34 : Différentes descriptions de la figure 2.33b.

Suivant le domaine physique, la description est souvent complétée par des informations sur les composants et les points de connexions. Ainsi certains points ont des spécificités particulières (point de masse, point bâti, point référence,...) ; des actions peuvent être appliquées sur les composants (frottements, forces,...).

Nous avons présenté le langage texte de description pour les schémas graphiques construits autour des blocs dipolaires. Avant d'appliquer le langage, nous allons d'abord étudier les éléments composant différents domaines de la physique du point de vue des blocs à une entrée et une sortie.

II.3.2. Blocs et composants élémentaires

II.3.2.1. Electriques

- Chaque élément peut être considéré comme une boîte contenant une composante de base à une entrée (***in***) et une sortie (***out***) ; ce sont des dipôles :

ELEMENT PHYSIQUE	SCHEMA	BLOC	SYMBOLE UTILISE
RESISTANCE		*in* *out*	R
CAPACITE		*in* *out*	C
BOBINE SELF		*in* *out*	L
SOURCE DE TENSION		*in* *out*	E
SOURCE DE COURANT		*in* *out*	I

fig 2.35 : Tableau des composants électriques.

Remarque : Par convention, c'est le sens du courant qui donne les affectations "***in***" et "***out***" aux blocs qui porteront un nom unique suivi d'un numéro représentant leur multiplicité afin d'éviter toute ambiguïté.

- Le transformateur peut être représenté par un bloc à 4 connexions : 2 entrées et 2 sorties, c'est un quadripôle.

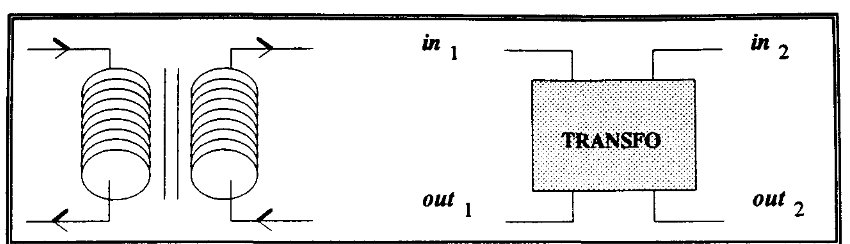


fig 2.36 : Bloc quadripôle associé au transformateur.

Pour garder la structure dipolaire on considère le transformateur composé de deux blocs à une entrée et une sortie. La figure 2.37 représente un transformateur composé d'un bloc primaire (**TFp**) et d'un bloc secondaire (**TFs**).

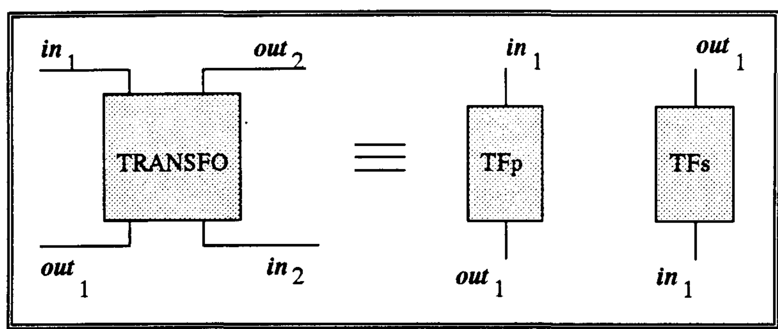


fig 2.37 : Transformateur décomposé en deux blocs.

II.3.2.2. Electroniques

La description des systèmes électroniques est analogue à celle des systèmes électriques. Aux éléments E, I ,R, C, L, on ajoute les transistors, les thyristors, les diodes qui seront modélisés en bond-graph par des switchs constitués d'un élément MTF booléen (m = 1 : passant, m = 0 : bloqué) et d'un élément R [DAUPHIN-TANGUY, ROMBAUT 93].

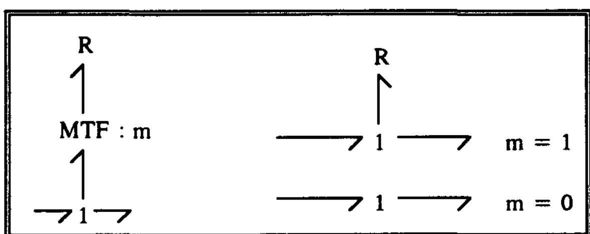


fig 2.38 : Bond-graph d'un circuit bloquant.

II.3.2.3. Mécaniques

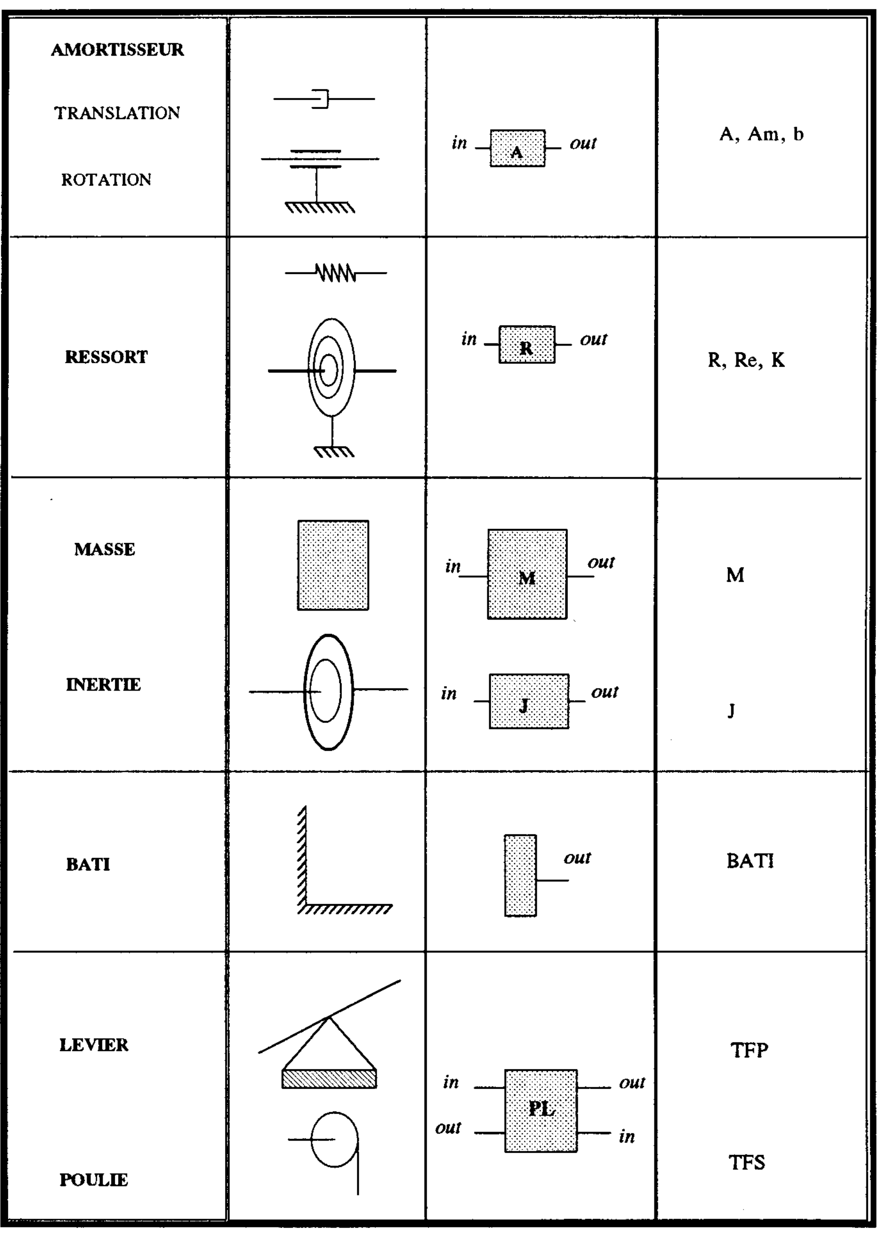
La mécanique est un domaine vaste et complexe, composé de mouvements en translation et en rotation à une ou à plusieurs dimensions. Un système mécanique à 1 dimension est constitué d'éléments qui peuvent être représentés facilement par des symboles schématiques (ressorts, masses, amortisseurs, inerties, frictions en translation et en rotation...).

Les domaines électriques et mécaniques à une dimension sont duaux. Tous les dipôles électriques doivent avoir leur correspondant dipôle en mécanique :

INDUCTANCE	MASSE , INERTIE
RESISTANCE	AMORTISSEUR
CAPACITE	RESSORT
SERIE	PARALLELE

fig 2.39 : Dualité des domaines électriques et mécaniques à 1 dimension.

Les ressorts (R) et les amortisseurs (A) peuvent être considérés comme des boîtes dipôles car ils possèdent deux connexions orientées suivant la vitesse des masses auxquelles ils sont attachés.



AMORTISSEUR TRANSLATION ROTATION		in A out	A, Am, b
RESSORT		in R out	R, Re, K
MASSE INERTIE		in M out in J out	M J
BATI		out	BATI
LEVIER POULIE		in out PL out in	TFP TFS

fig 2.40 : Tableau des composants mécaniques.

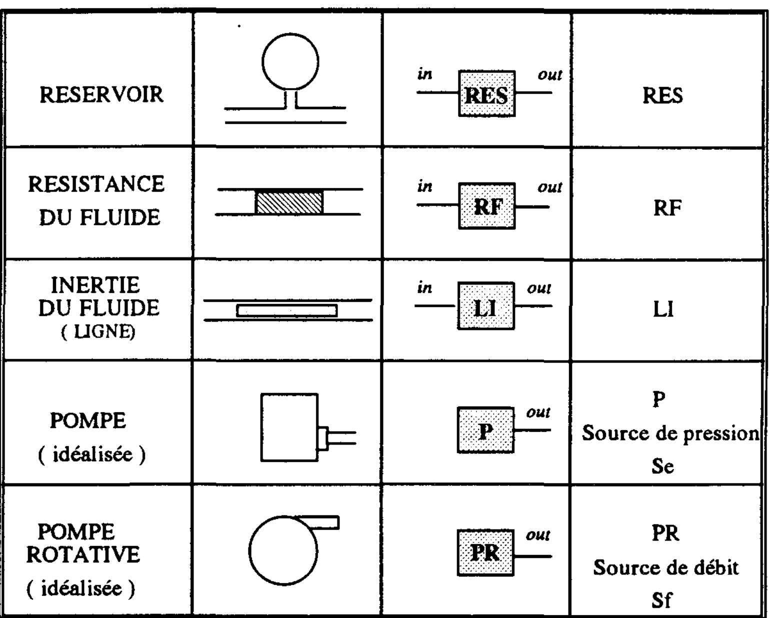
Le bâti est une masse inerte de vitesse nulle qui, comme pour la masse électrique, peut être représenté par un bloc donnant une information sur les entrées et sorties des autres blocs auxquels il est connecté.

Les poulies et les leviers sont traités à la manière des transformateurs électriques. en entrée nous avons une force **F1** et une vitesse **V1**, en sortie une force **F2** et **V2**.

Les domaines mécaniques de rotation (mr) et mécaniques de translation (mt) en 1 dimension se traitent exactement de la même façon.

II.3.2.4. Hydrauliques

Les systèmes hydrauliques se modélisent de la même manière que les systèmes électriques. La puissance hydraulique est le produit de la pression **P** et du débit **Q**.



RESERVOIR		in RES out	RES
RESISTANCE DU FLUIDE		in RF out	RF
INERTIE DU FLUIDE (LIGNE)		in LI out	LI
POMPE (idéalisée)		P out	P Source de pression Se
POMPE ROTATIVE (idéalisée)		PR out	PR Source de débit Sf

fig 2.41 : Tableau des composants hydrauliques.

II.3.2.5. Conclusion

Il conviendra pour chaque domaine physique d'utiliser un dictionnaire pour les éléments. Le nom du bloc utilisé sera toujours associé à sa fonction physique.

ELEMENT PHYSIQUE	FONCTION PHYSIQUE	SYMBOLES DE DESCRIPTION	SYMBOLE UNIVERSEL EN LINEAIRE
AMORTISSEUR	$\Phi(F , V) = 0$	A, Am, Amort, ...	b
RESISTANCE ELECTRIQUE	$\Phi(U , i) = 0$	R, Rés, Résist..	R
CONDENSATEUR	$\Phi (U \int idt) = 0$	C, Con, ...	C
RESSORT	$\Phi (F ,\int Vdt) = 0$	Re, Ress, ...	1 / k

fig 2.42 : Tableau des éléments physiques à différents symboles de descriptions.

Par exemple l'amortisseur et la résistance électrique relient des variables d'effort et de flux, et dissipent la puissance fournie sous forme d'énergie calorifique. On peut stocker cette notation et se constituer un dictionnaire de symboles pour la description de fonctions physiques en linéaire (*). L'important est pour l'utilisateur de rester cohérent dans sa notation qui, rappelons-le, est fortement liée à la numérotation des éléments élémentaires du système. L'utilisateur peut imposer sa notation en début de description et doit la garder tout au long de celle-ci.

II.3.3. Langage pour un modèle homogène

Pour chaque domaine, nous avons un langage texte afin de décrire le plus fidèlement possible un système sous son schéma physique. [REMY 90, RINGOT 90]

II.3.3.1. Electrique (e)

Par un exemple pris dans le domaine électrique, nous allons dégager des remarques intuitives qui aideront à structurer une démarche ainsi que les prémices d'un langage. Prenons le circuit électrique de la figure 2.43.

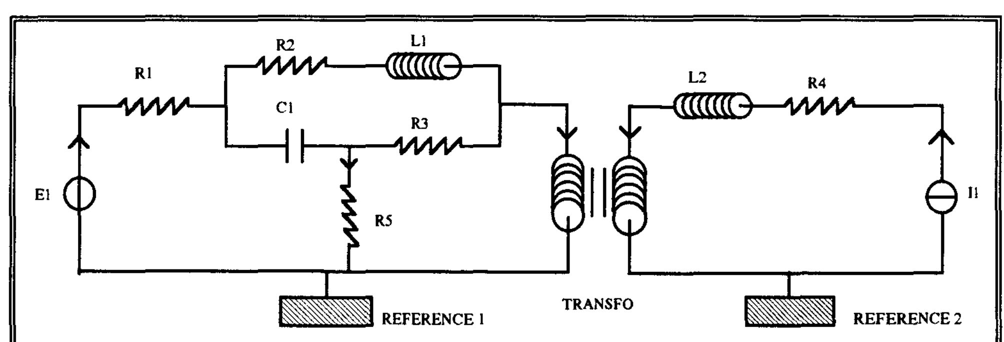


fig 2.43 : Schéma d'un système électrique avec un transformateur.

- Les points de références, notés m1 et m2 peuvent être considérés comme des points particuliers connectés sur les entrées et sorties des autres blocs. On obtient le schéma "blocs" suivant :

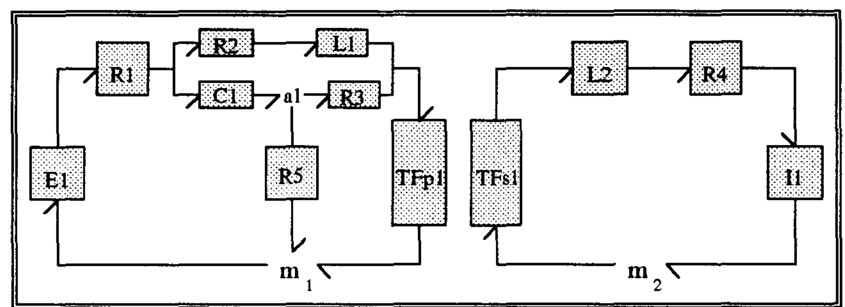


fig 2.44 : Description graphique du système de la figure 2.43 sous forme de bloc.

(*) A Chaque symbole de description d'un élément physique on associe souvent un élément bond-graph. Voir tableau de la fig 2.55.

La description se fait selon le sens du courant imposé dans les circuits à l'aide des symboles '+', '/', '(', ')', point, point de référence, composant). En en tête on déclare le domaine physique dans lequel on travaille, en l'occurrence "électrique".

```
electrique
m1 + E1 + R1 + (R2 + L1) / (C1 + a1 + R3) + TFp1 + m1
a1 + R5 + m1
m2 + TFs1 + L2 + R4 + I1 + m2
```

fig 2.45 : Description texte de la figure 2.44.

II.3.3.2. Mécanique à 1 dimension (m)

C'est un domaine assez vaste et complexe. Nous pouvons procéder de la même façon avec un système mécanique à une dimension, figure 2.46, en représentant les éléments de bases par des blocs.

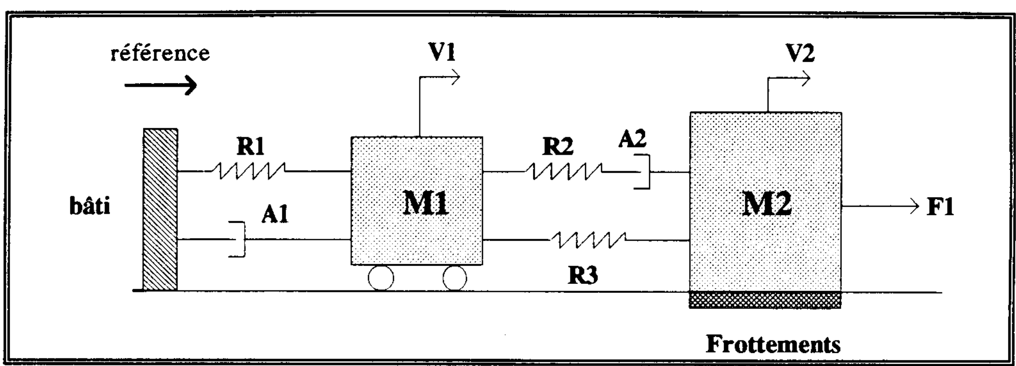


fig 2.46 : Schéma d'un système mécanique de translation à 1 dimension.

Dans ce cas on doit imaginer une "attache" pour connecter les blocs dipôles masses avec les blocs ressorts et amortisseurs, nous imaginons une attache virtuelle symbolisée par un point.

La figure 2.47 est le résultat de toutes les remarques faites plus haut. La description est composée de deux parties : une graphique, suivant la structure du système originel avec les notions de blocs et points, l'autre partie regroupe les informations des différentes composantes du système.

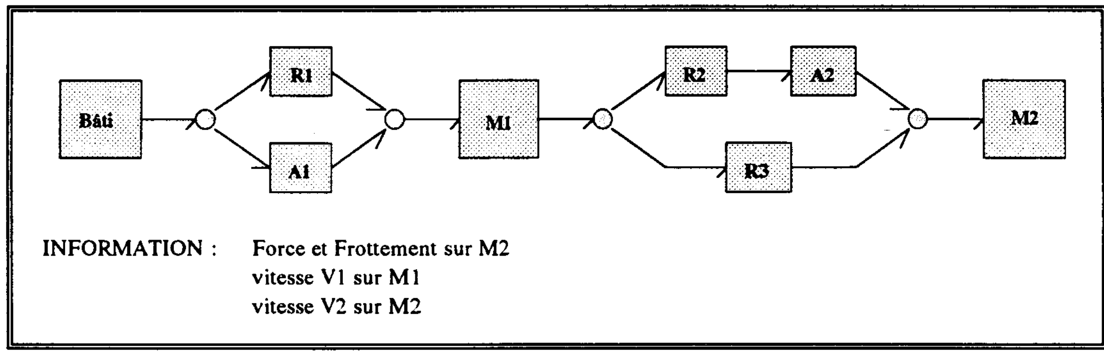


fig 2.47 : Description graphique du système de la fig 2.46.

La description se fait en suivant la propagation de la vitesse des masses, suivant une direction pré définie qui sert de référence.

```
mécanique
Bâti + R1 / A1 + M1 + (R2 + A2) / R3 + M2
Fr1 (M2)
F1 (M2)
V1 (M1)
V2 (M2)
```

fig 2.48 : Description texte de la fig 2.47.

Si l'entrée (force ou vitesse) est appliquée à un autre élément qu'une masse, il faut se définir un point d'application intermédiaire. Ce point est en quelque sorte une masse nulle.

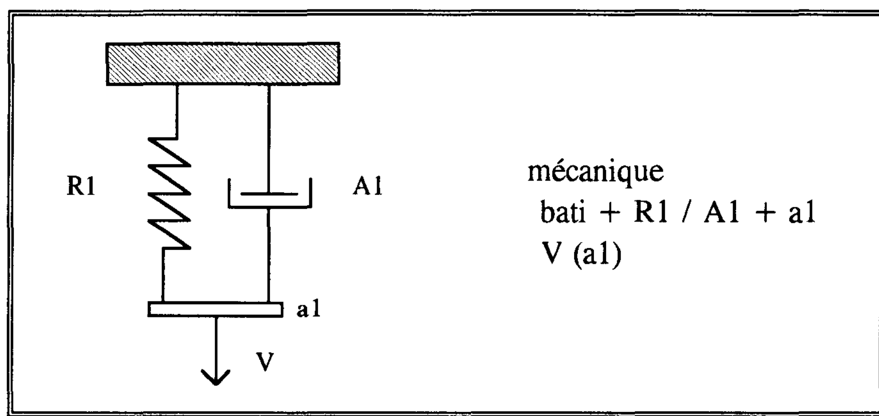


fig 2.49 : Modèle mécanique avec un point de connexion a1.

II.3.3.3. Hydraulique (h)

Lorsque le volume du fluide est conservé et qu'il est incompressible, les systèmes hydrauliques peuvent être traités à la manière des circuits électriques. Comme nous l'avons souligné au paragraphe 3.2.4, nous pouvons associer des composants élémentaires aux blocs dipôles hydrauliques.

L'exemple de la figure 2.50, montre un circuit hydraulique composé d'une pompe, d'un réservoir et d'une valve (**Va**) reliés entre eux par des lignes (**Li**) synthétisant l'inertie et la résistance du fluide dans les conduits hydrauliques. Le fluide provient d'un bac où la pression de référence sera notée "**mp**".

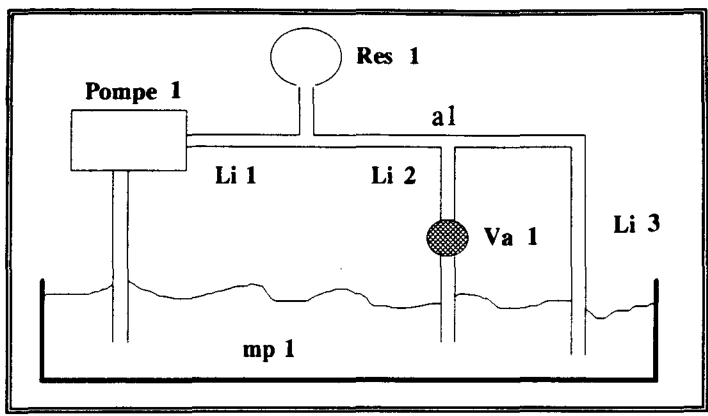


fig 2.50 : Système hydraulique.

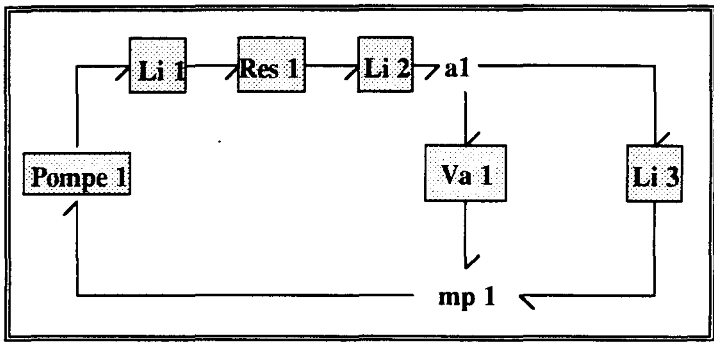


fig 2.51 : Description graphique de la fig 2.50.

La description se fait en suivant le sens de circulation du fluide dans le réseau hydraulique.

```
hydraulique
mp1 + Pompe1 + Li1 + Res 1 + Li2 + a1 +Li3 +mp1
a1 + Va1 + mp1
```

fig 2.52 : Description texte.

II.3.3.4. Conclusion

On constate une limitation de la représentation par blocs dipôles et la description série-parallèle. En effet, celle-ci demande une réticulation du système physique pour faire apparaître une représentation de type réseau.

De plus elle est limitée aux systèmes simples en mécanique (à 1 dimension seulement). Son intérêt réside dans son utilisation aisée surtout pour les systèmes électriques monophasés et ne demande pas de connaissances sur les bond-graphs.

Le but d'une telle description est de générer automatiquement un bond-graph par une méthode que nous allons présenter dans les prochains paragraphes.

II.3.4 Génération d'un Bond-graph à partir d'une description texte

II.3.4.1. Composants élémentaires et éléments bond-graphs

Les éléments physiques simples présentés dans le paragraphe précédent, ont leur représentant bond-graph 1-port ou 2-port [KARNOPP 83 - THOMA 75]. Cela est vrai pour des composants électriques et mécaniques qui peuvent être modélisés par des éléments bond-graphs usuels simples : R, C, I, Se, Sf, TF et GY que nous avons résumé dans le tableau de la figure 2.55.

Il est souvent nécessaire d'utiliser une combinaison de différents éléments bond-graphs (1-port, 2-ports, multi-port), pour modéliser plus finement des phénomènes ou des composants physiques.

Par exemple, la modélisation d'un vérin hydraulique peut être en première approximation un transducteur TF seul (figure 2.53).

En général ces "blocs" sont constitués de plusieurs composants élémentaires à plusieurs entrées et sorties où les notions de séries et parallèles sont difficiles à employer. Si on prend en compte la compressibilité du fluide dans la chambre du vérin et le frottement du piston dans la chambre, on obtient le modèle bond-graph de la figure 2.54.

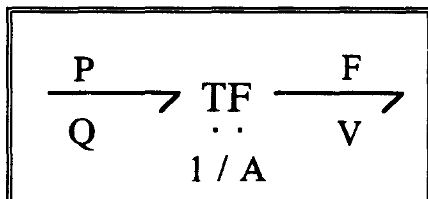


fig 2.53 : Bloc bond-graph d'un vérin.

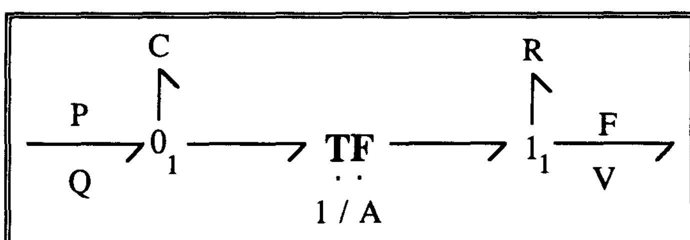


fig 2.54 : Bloc enrichi d'un vérin.

Elément Bond-graph	Domaine Physique	Elément Physique	Symbole Physique	Symbole Universel
R	Electrique (e) & Electronique (el)	Résistance	R, Res, ...	R
	Mécanique de translation (mt)	Amortiseur Force de Frottement sur une masse M	A, Am, Amort Fr (M)	b
	el	Commutateur	Sw	Sw
C	e & el	Condensateur	C	C
	mt	Ressort	Re, Ress, ...	1 / K
I	e & el	Bobine	L, Self, ...	L
	mt	Masse	M, Masse, ...	M
Se	e & el	Source de tension	E, SE, ...	E
	mt	Force sur une masse M	F (M)	F
Sf	e & el	Source d'intensité	I, SI, INT.	I
	mt	Vitesse sur une masse M	V (M)	V
TF	e & el	Transformateur	TFp, TFs	m
GY	e & el	Gyrateur	GYp, GYs	r

fig 2.55 : tableau de conversion associant éléments bond-graphs, éléments physiques et symboles universels linéaires.

Ce tableau sera utilisé pour construire une table de conversion des éléments physiques du domaine étudié. Il sera utilisé principalement par le module de dessin du bond-graph à l'écran. Dans le chapitre III, nous verrons la façon de le stocker en mémoire et son rôle dans la renumérotation lors du couplage.

II.3.4.2. Principe de la méthode

A. Systèmes électriques

Pour les systèmes électriques, la méthode suit fidèlement les procédures établies par KARNOPP et ROSENBERG [75] pour la construction systématique d'un bond-graph à partir d'un schéma du système physique idéal. Chaque noeud de tension est associé à une jonction **0**, deux éléments en séries sont traversés par le même courant (flux), deux éléments en parallèles ont la même tension (effort).

La méthode consiste, dans un premier temps, à construire automatiquement le bond-graph du système physique homogène à partir de sa description physique. Puis de réduire le bond-graph par les simplifications usuelles.

Elément 1-port :

Pour cela nous associons à chaque symbole physique, un élément bond-graph (voir tableau des conversions figure 2.55) attaché à une jonction 1 indicée suivant l'ordre d'apparition des éléments. On orientera le lien d'attache en fonction de l'élément bond-graph correspondant.

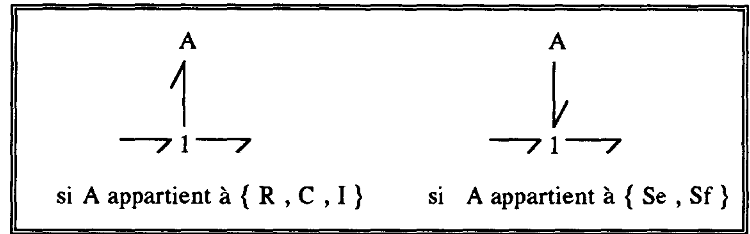


fig 2.56 : Représentation d'un élément bond-graph.

Elément 2-ports :

Nous avons vu que le transformateur est constitué des enroulements primaires (**TFp**) et secondaires (**TFs**) qui se comportent, dans la description, comme un élément 1-port. Il suffit de lier les jonctions **1** par une jonction **TF** et d'éliminer les éléments **TFp** et **TFs**.

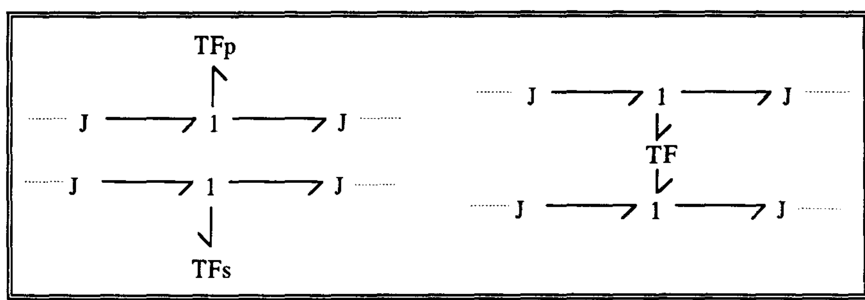


fig 2.57 : Traitement du transformateur électrique.

Opération série :

Pour chaque opération série ' + ' on crée une jonction **0** entre les éléments bond-graphs A et B, les liens seront orientés selon le sens de la description qui correspond à la circulation du courant dans le réseau :

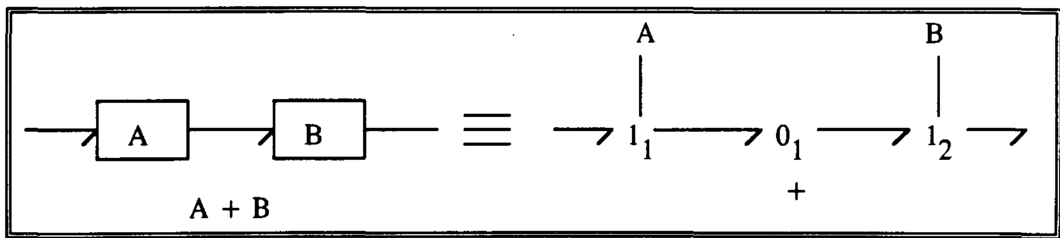


fig 2.58 : représentation bond-graph de l'opération série.

Cette jonction **0** est provisoire et sera simplifiée (*) par la suite comme le montre la figure suivante. Ainsi nous avons bien la traduction physique de l'opération série signifiant que les éléments **A** et **B** ont le même flux et sont donc attachés à une jonction.

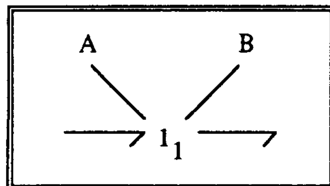


fig 2.59 : Simplification de la figure 2.58.

Le point :

Le point de connexion est, rappelons le, un point pris dans l'ensemble des minuscules {a ... z} et indicé selon l'utilisateur. Il est considéré comme un point d'effort, on l'associe à une jonction **0** à laquelle on attache le point en question, on obtient ainsi une **jonction_point**.

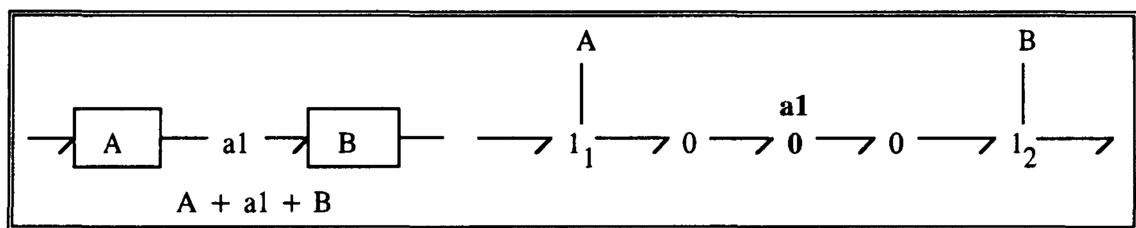


fig 2.60 : Représentation bond-graph d'un point.

Il sert à rendre la description plus facile et structurée en éclatant le réseau lorsque celui-ci ne peut être décrit en une seule fois. Il suffira ensuite de confondre les jonctions points tout en éliminant les redondances.

(*) Voir plus loin les règles de Simplification.

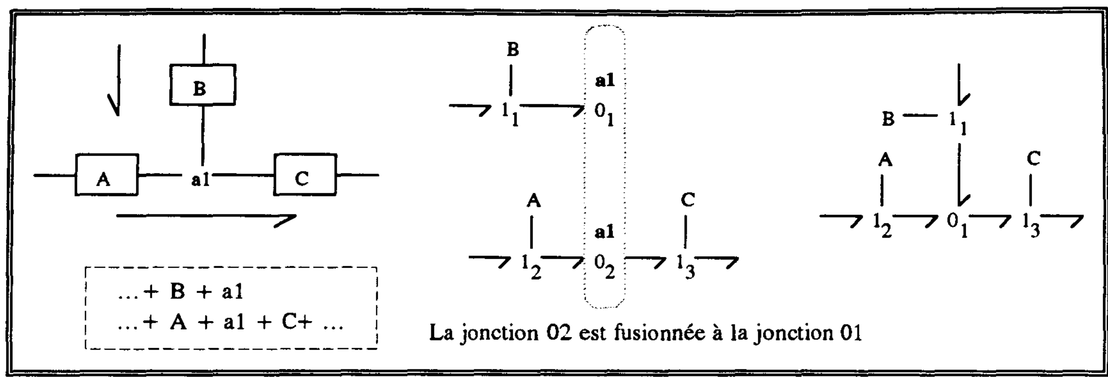


fig 2.61 : Point reliant plusieurs éléments bond-graphs.

Points particuliers :

Les masses électriques (m), le bâti mécanique (bâti) et la pression de référence (mp) sont considérés comme des points de connexions que nous traiterons différemment des autres points. Ces points nous les nommerons **points_références.**

Opération parallèle :

Pour l'opération parallèle ' / ' on crée un pont dont les extrémités sont des jonctions **0** pour traduire le même effort des éléments A et B.

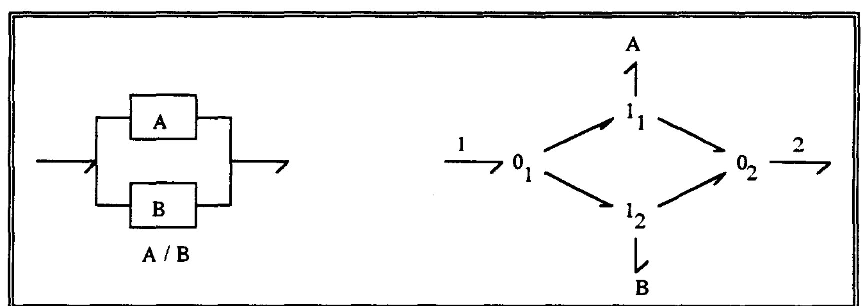


fig 2.62 : Représentation bond-graph de l'opération parallèle.

On aurait pu produire la figure 2.63 équivalente à la figure 2.62 Les avantages sont l'obtention d'un bond-graph avec un minimum de boucles qui permettent d'optimiser l'affectation de la causalité ainsi que la recherche des boucles causales nécessaires à la production des équations mathématiques.

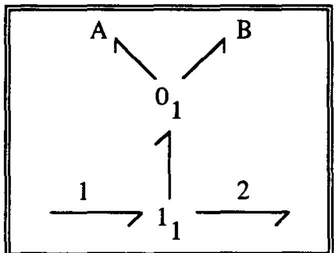


fig 2.63 : Transformation du pont bond-graph de la fig 2.62.

Cependant, cette forme est moins proche de la réalité topologique.

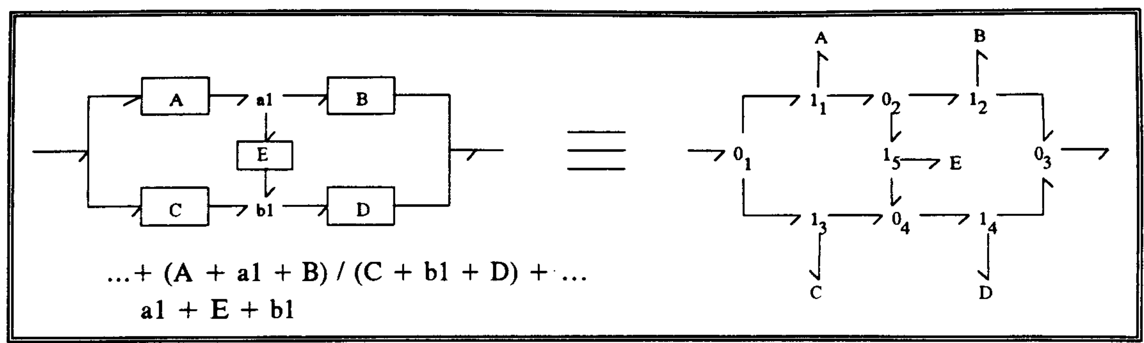


fig 2.64 : Combinaison du point et de l'opération parallèle.

Parenthèses :

Dans la figure 2.64, nous avons une application directe de l'utilisation des parenthèses. Elles interviennent lorsque plusieurs groupes d'éléments en série sont en parallèle.

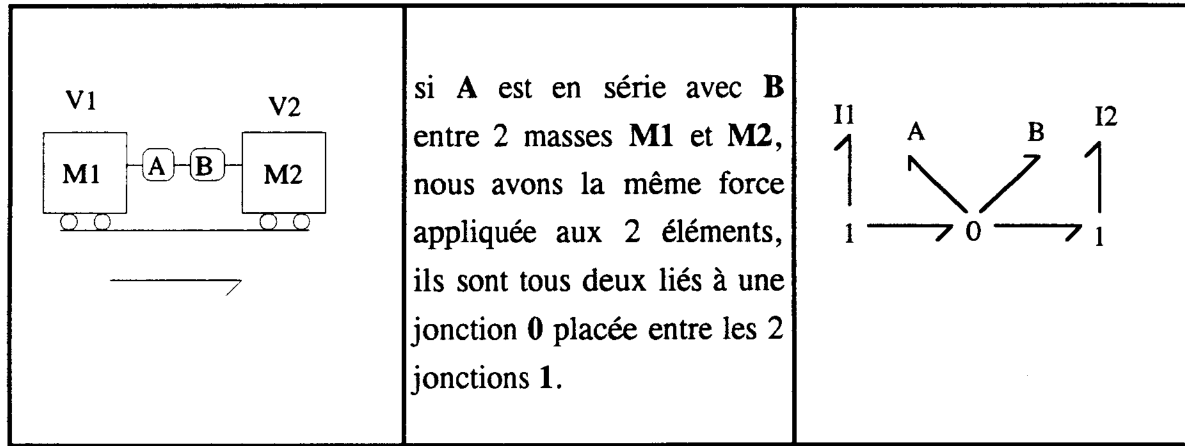
Pour les systèmes électroniques et hydrauliques, la procédure dans son principe reste la même.

B. Systèmes Mécaniques

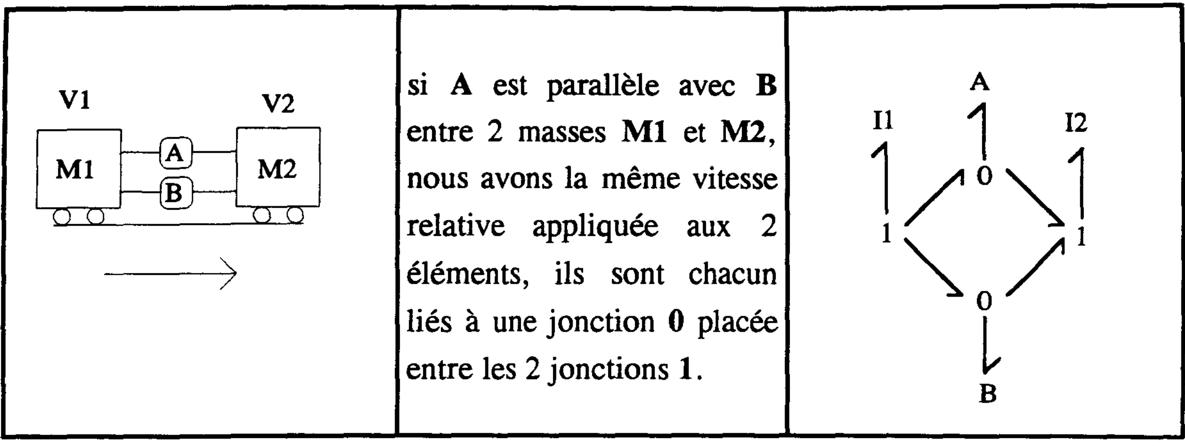
En mécanique, il est plus facile de visualiser les vitesses que les forces dans les différentes parties du système.

1. Pour chaque Masse (élément inertiel) on crée une jonction **1** associée à une vitesse absolue de déplacement V.

2. Connecter les Ressorts (éléments capacitifs) et les Amortisseurs (éléments résistifs) présents entre 2 Masses à l'aide des jonctions **0** liées par des liens orientés selon le sens de la vitesse relative des 2 Inerties. Soit A et B (éléments capacitifs et/ou résistifs), nous pouvons avoir les cas suivant :



si **A** est en série avec **B** entre 2 masses **M1** et **M2**, nous avons la même force appliquée aux 2 éléments, ils sont tous deux liés à une jonction **0** placée entre les 2 jonctions **1**.



3. Connecter les sources d'efforts et de vitesses correctement.
4. Orienter les liens qui ne le sont pas selon le sens conventionel.
5. Simplifier le graphe, en particulier si celui-ci possède un bâti avec une vitesse de référence nulle sur une jonction 1, dans ce cas on élimine tous les liens qui lui sont attachés.

Phase de simplification :

Après avoir construit le bond-graph du système, il faut le simplifier afin d'éliminer, dans un premier temps, les jonctions caractérisant les **points_références** et les liens qui leur sont attachés, dans un deuxième temps les jonctions redondantes par l'application des règles de simplifications usuelles (figure 2.65) [KARNOPP & ROSENBERG 75].

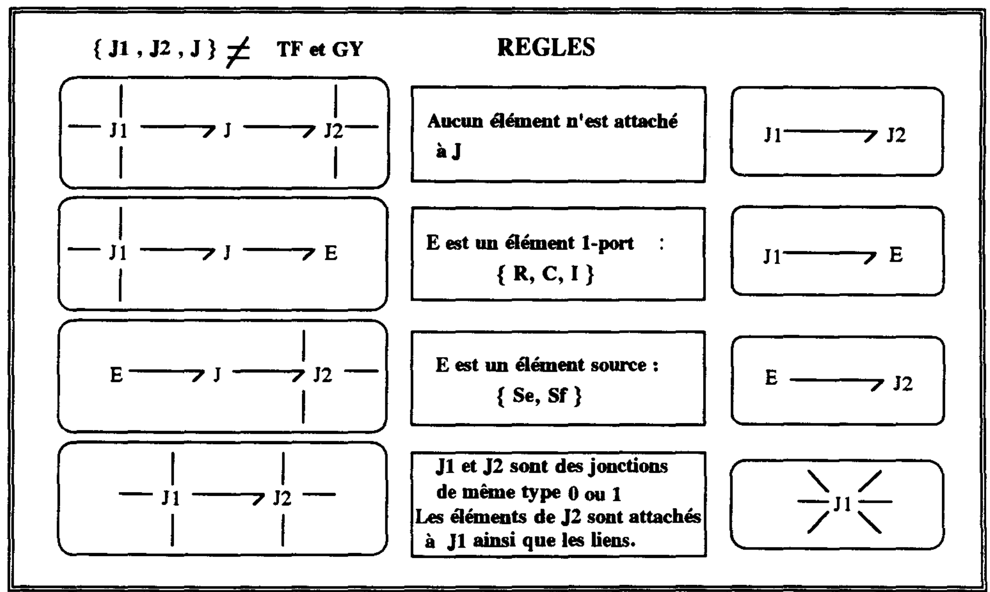


fig 2.65 : Principales règles de simplification.

Remarque : La notation série et parallèle est uniquement topologique, elle traduit ce que l'utilisateur voit sur le schéma. La construction du bond-graph dépendra de l'analyse de la description en fonction du domaine physique qui sera toujours précisé en tête du fichier texte.

Nous allons appliquer la méthode sur 2 exemples. Dans le chapitre IV, nous présenterons la grammaire et la génération du code bond-graph.

II.3.4.3. Exemples

A. Système électrique

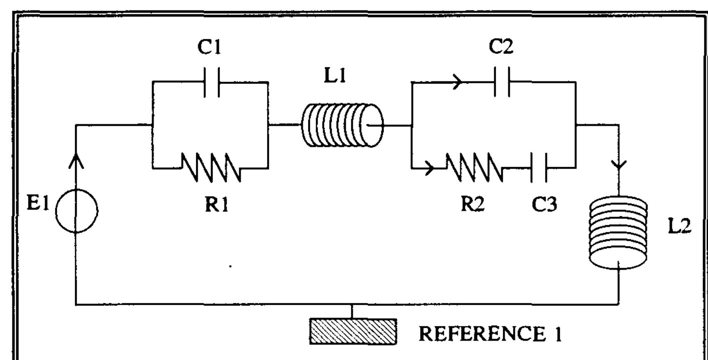


fig 2.66 : Schéma graphique du système électrique.

Electrique
m1 + E1 + C1 / R1 + L1 + C2 / (R2 + C3) + L2 + m1

fig 2.67 : Description texte du système électrique.

Lors de la lecture des éléments, la liste de conversion est créée en regard du tableau de la figure 2.55.

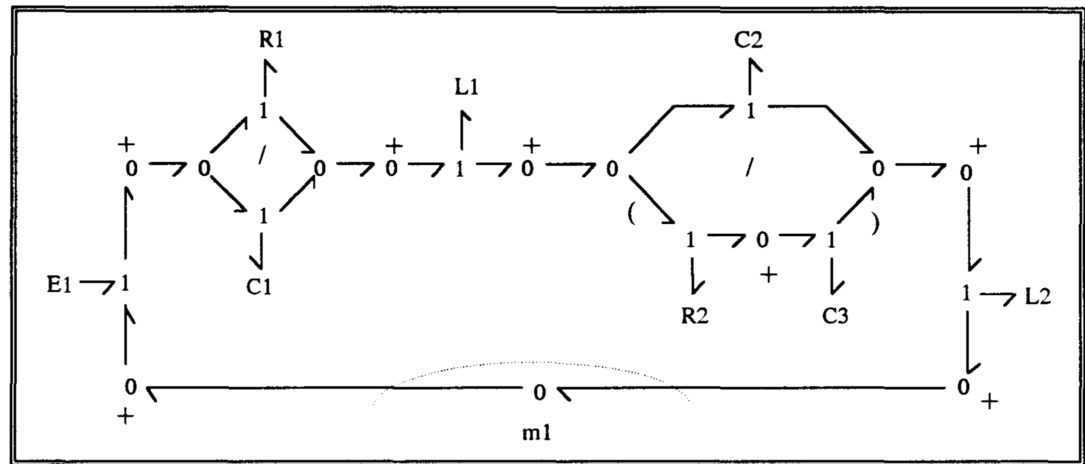


fig 2.68 : Bond-graph non simplifié construit à partir de la description texte.

Après la simplification des masses, des jonctions et des liens ; chaque type de jonction est numéroté dans l'ordre croissant.

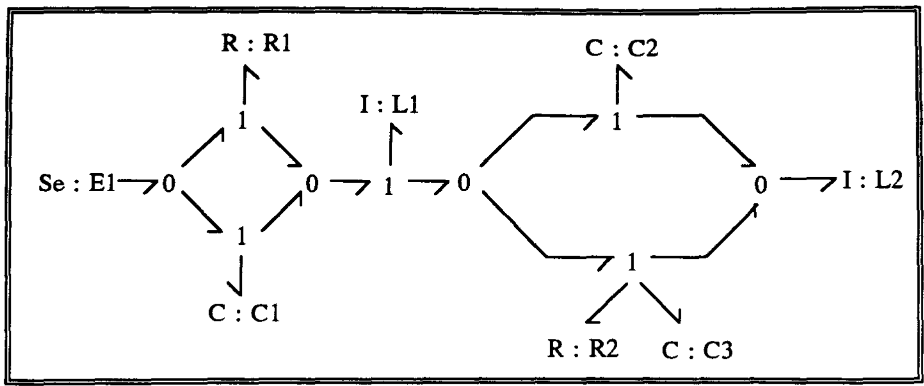


fig 2.69 : Bond-graph simplifié du système électrique avec le tableau de conversion.

B. Système mécanique

De la même façon, pour un système mécanique.

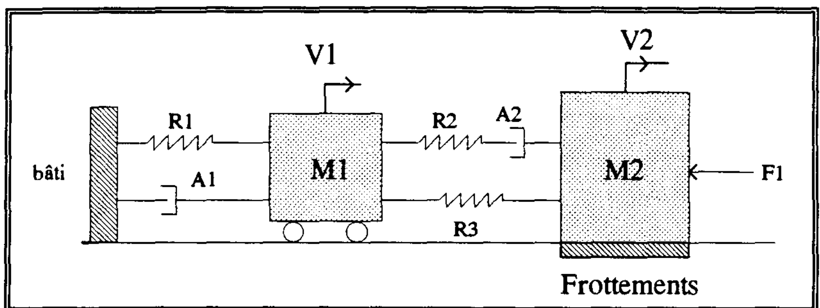


fig 2.70 : Schéma graphique d'un système mécanique à 1 dimension.

mécanique
bati + R1 / A1 + M1 + R2 / (R3 + A2) + M2
F1 (M2)
Fr1 (M2)

fig 2.71 : Description texte du système mécanique.

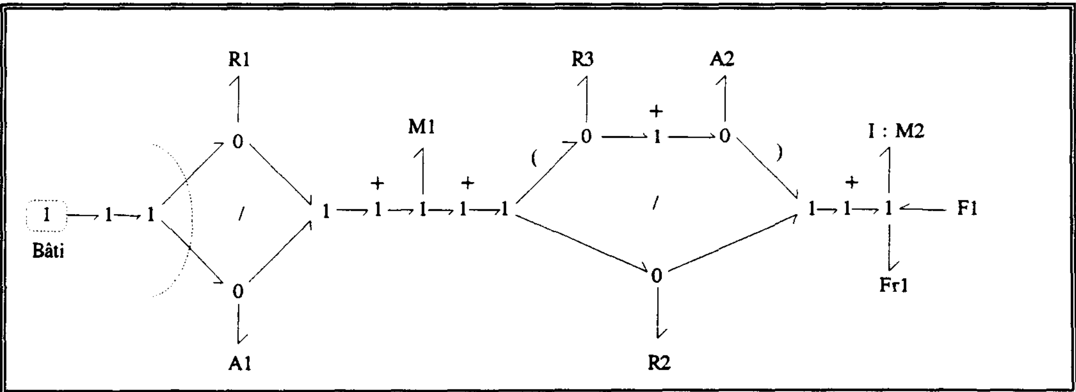


fig 2.72 : Bond-graph non simplifié.

Le bâti est considéré comme un noeud de vitesse nulle, on simplifie la jonction référence en éliminant les jonctions et les liens qui lui sont attachés. Le tableau des conversions est créé.

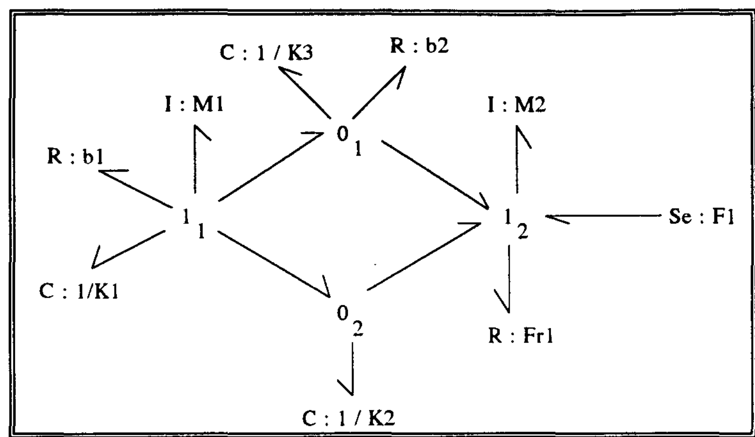


fig 2.73 : bond-graph simplifié du système mécanique.

Les deux modèles bond-graphs ont des descriptions série et parallèle presque équivalentes et pourtant les 2 modèles bond-graphs sont différents. Il est donc important de spécifier en début de description le domaine dans lequel on travaille, pour générer le bond-graph adéquat.

II.3.5. Etude de la description d'un bloc

II.3.5.1. A partir d'éléments de bases appartenant au même domaine

A partir des éléments de base d'un domaine physique donné, il est possible d'utiliser le langage de description physique afin de construire un bloc. Puisqu'un bloc est défini par ses ports, la solution consiste à créer des "liens ports" en entrées (***in***) et en sorties (***out***) sur le bond-graph construit à partir de la description.

Pour cela il faut enrichir le langage de manière à indiquer les connexions du bloc. L'idée est d'utiliser les points de connexions comme le support d'un ou plusieurs ports. Il y a deux étapes pour créer un bloc physique.

La première est la description du schéma physique comme nous l'avons vu (commençant par le domaine physique), à la différence que chaque emplacement susceptible d'être un port sera représenté par un point de connexion.

La deuxième est une validation des ports en indiquant les points qui seront en entrées (***in***) ou en sorties (***out***) à l'aide de la syntaxe.

'port'

'point' = 'liste des ports assignés à ce point'

Au niveau bond-graph, nous avons vu que les points sont représentés, soit par des jonctions **0** dans le cas des systèmes électriques, soit par des jonctions **1** dans le cas mécanique. Les ports sont des liens particuliers ajoutés aux **jonctions_points** du bond-graph.

Par exemple le point **a1** a deux ports, un en entrée (***in***$_n$), et un en sortie (***out***$_m$).

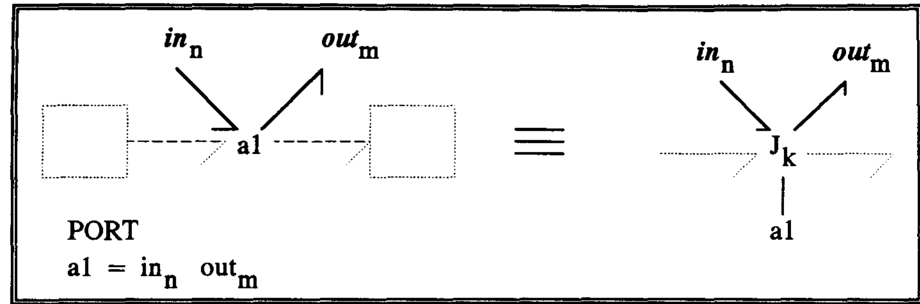


fig 2.74 : Représentation Bond-graph des ports in et out attachés au point a1.

Pour un bloc et à l'inverse des autres **jonctions_points**, les points utilisés pour décrire les ports du système ne seront pas éliminés lors de la simplification.Nous aurions pu introduire le port dans la description même du bond-graph. Mais dans ce cas nous aurions perdu la modularité et la mobilité des ports dans la description du bloc.

Exemple :

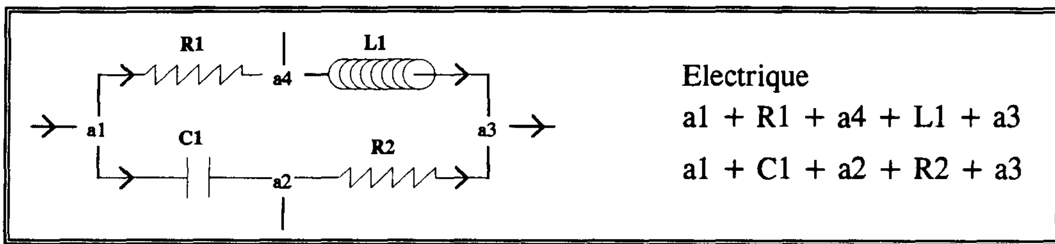


fig 2.75 : Bloc électrique avec la description texte associé.

Le fait de séparer la description en deux parties, a pour avantage de permettre d'ajouter ou de changer l'affectation des ports sur les points sans changer l'architecture du système, donc sa description physique. La nature des ports est fonction du domaine traité. Par exemple, tous les ports du système de la figure 2.75 sont automatiquement de nature électrique.

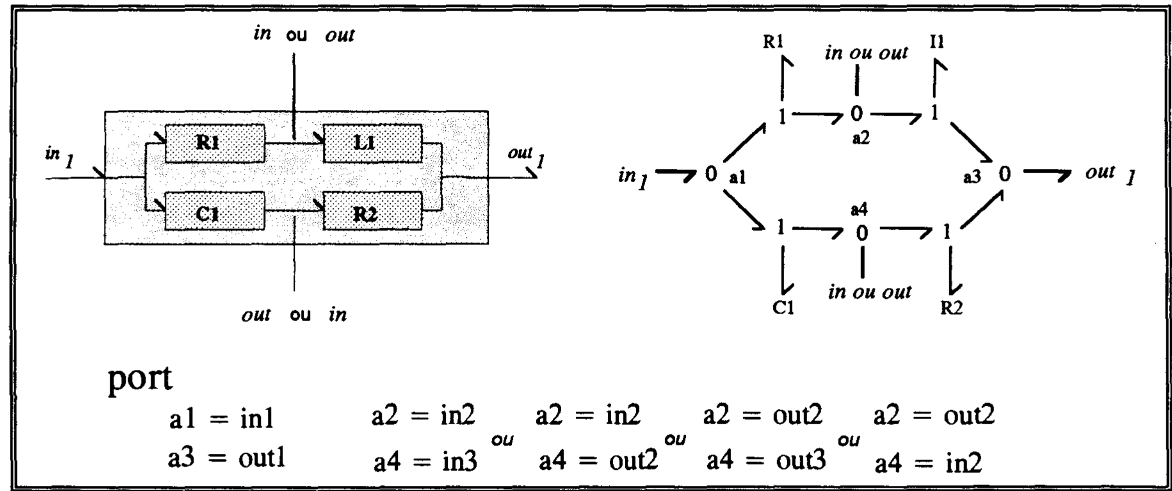


fig 2.76 : Affectation des ports et bond-graph associés.

Si les points **a2** et **a4** ne sont pas utilisés comme ports, les **jonctions_points** correspondantes seront éliminées lors de la phase de simplification.

Remarque : Lorsque l'utilisateur décrit un système électrique, il impose en quelque sorte le sens du courant dans le réseau. Pour construire un bloc, nous avons vu qu'il lui suffit d'ajouter des points ou noeuds de tension dans le réseau, aux endroits où il désire créer des ports en entrées ou en sorties.

D'après l'exemple de la figure 2.85, certains points, comme **a2** et **a4** peuvent être arbitrairement en ***in*** ou en ***out***, selon l'utilisation du bloc, tandis que d'autres, comme **a1** et **a3** seront obligatoirement en entrées ou en sorties.

Cette mixité du port, que l'on peut rapprocher de la causalité variable proposée par [BIDARD 93], est donc limitée par l'orientation du réseau. Le problème de l'orientation d'un réseau est d'ailleurs un problème combinatoire.

De plus, de toutes les orientations possibles, il s'agit de repérer celles qui ont un "sens" du point de vue de la physique. Dans le cas général, on peut se poser la question du sens physique du couplage des blocs bond-graphs, à savoir l'importance du sens des ports dans un bloc, la cohérence physique du bloc bond-graph qui lui est associé, et enfin l'existence physique du bond-graph résultant du couplage en lui-même.

Par exemple un filtre passe bas peut être décrit par le schéma de la figure 2.77 suivante. Les points **a1** et **a2** ont été rajoutés au schéma initial pour créer les ports in_1 et out_1 de nature électrique.

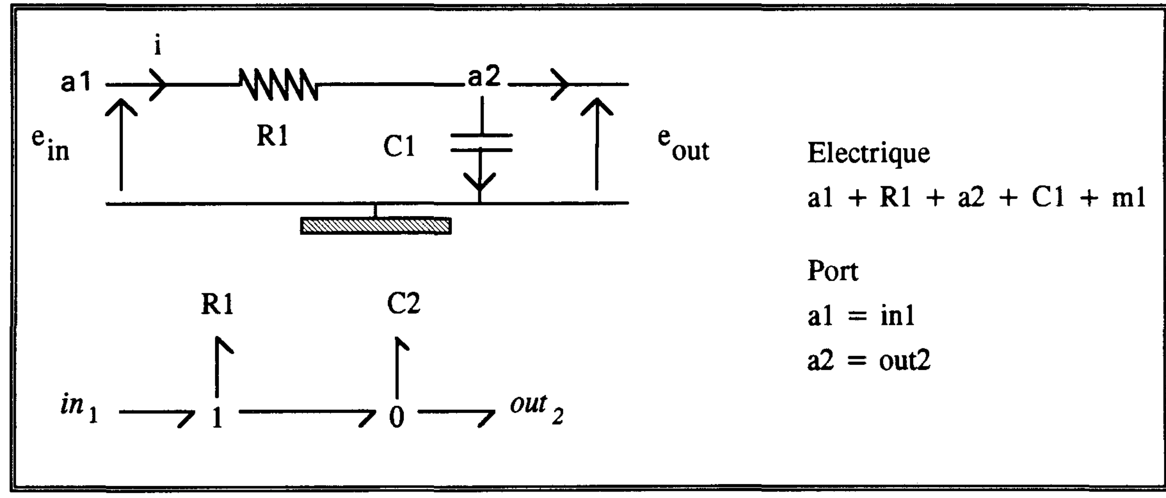


fig 2.77 : Description physique du bloc filtre.

II.3.5.2. A partir d'éléments de base appartenant à des domaines différents

Pour l'instant, nous avons décrit des systèmes appartenant à un seul domaine physique à la fois. Considérons maintenant des systèmes mélangeant différents domaines physiques.

La méthode de génération d'un bond-graph est basée essentiellement sur les modèles homogènes. Comme le montre la figure 2.78, Le fait de mélanger les éléments physiques entre eux sans imposer un ordre dans la description risque de générer un mauvais bond-graph qui n'aurait pas vraiment de sens physique.

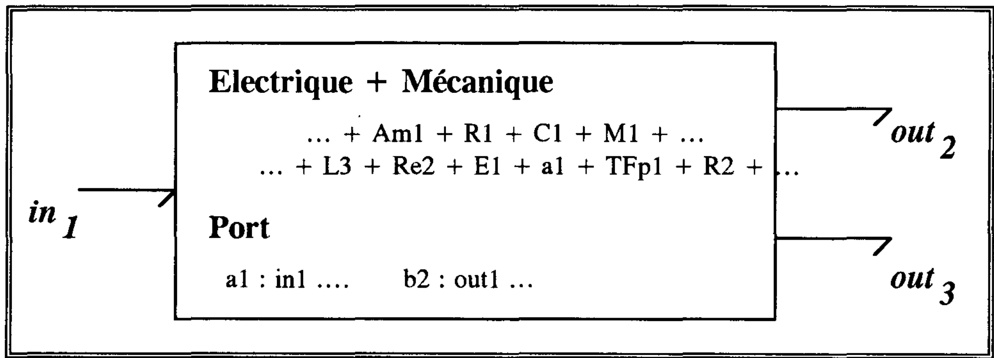


fig 2.78 : Bloc formé d'éléments appartenant à plusieurs domaines.

Nous préférons contrôler un tel mélange par la définition de bloc appartenant à un seul domaine à la fois, puis une association ou couplage par le langage blocs & noeuds.

Pour obtenir un bloc, il suffit de spécifier des ports d'entrée et de sortie sur les noeuds en question. Les blocs peuvent être composés par l'association de plusieurs blocs dipôles à 1 entrée et 1 sortie qui à leur tour peuvent se combiner entre eux afin de constituer un bloc avec plusieurs entrées et sorties. C'est le principe "d'imbrication" que nous illustrons dans la figure 2.79. (Voir le chapitre IV)

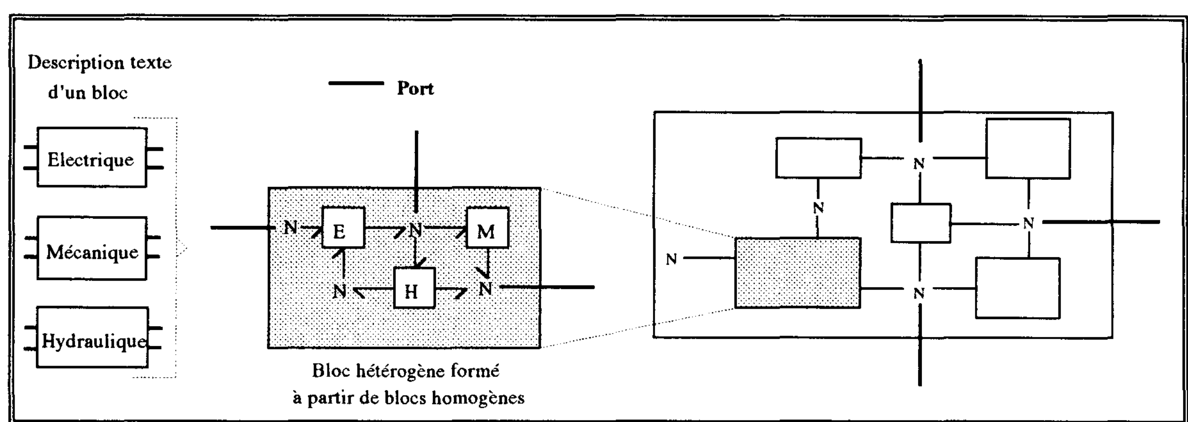


fig 2.79 : Principe d'imbrication.

Ici, les noeuds de connexion ne pourront pas toujours se ramener à de simples jonctions (**0** ou **1**), mais comporteront obligatoirement des éléments transducteurs (**TF** et **GY**) qui régissent le transfert de puissance entre différents domaines de la physique.

Conclusion :

On ne peut pas toujours représenter les blocs hétérogènes par une description uniquement physique. Certains blocs ne peuvent être représentés que sous une forme bond-graph, mathématique, ou une combinaison des deux représentations. ARCHER étant axé sur l'analyse du bond-graph, nous étudierons, dans les paragraphes suivants, la façon de décrire un système formé par un bond-graph ou par des blocs bond-graphs.

II.4. Bond-graphs et blocs Bond-graphs

Nous proposons ici une nouvelle procédure d'écriture d'un bond-graph, bien adaptée à notre problème, en utilisant la définition d'un graphe simple et la convention d'écriture citée plus haut.

Comme son nom l'indique, le bond-graph est avant tout un graphe dont les noeuds regroupent l'ensemble des jonctions et éléments bond-graphs (1-ports et multi-ports) et les arcs l'ensemble des orientations possibles (puissance et causalité). La figure 2.80 montre l'organisation graphique d'un bond-graph :

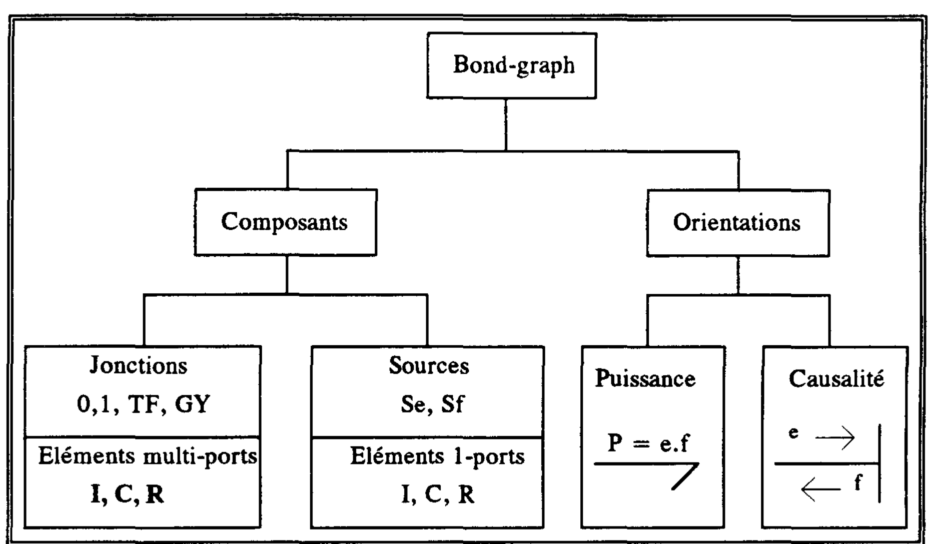


fig 2.80 : Structure graphique d'un bond-graph.

Les éléments bond-graphs sont numérotés afin de ne pas les confondre entre eux et éviter ainsi toute ambiguïté. De même nous préférons numéroter les **jonctions** car elles sont souvent moins nombreuses que les liens.

Chaque élément bond-graph peut représenter un élément physique, dans ce cas on numérote les éléments physiques. La syntaxe suivante est utilisée :

'Elément bond-graph' ' : ' 'Symbole physique numéroté'

Exemple :

R : Rl, R : Aml, C : Ressl, C : Cl, I : Ml, I : Jl, I : Ll ...

Si l'on préfère travailler uniquement avec des éléments bond-graphs, on procède de la façon suivante, on numérote les éléments bond-graphs dans un ordre propre à l'utilisateur comme le montre la figure 2.90.

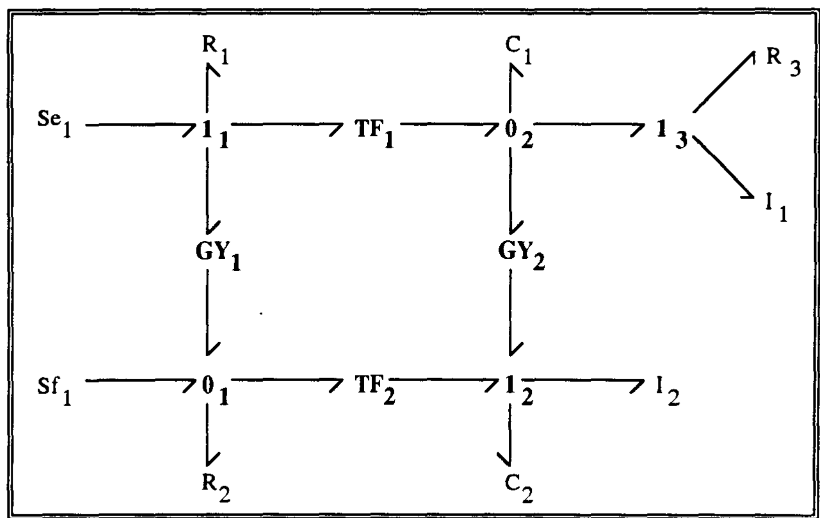


fig 2.81 : Convention de notation pour ARCHER.

Cette convention de numérotation sera suivie tout au long de la thèse et sera respectée dans tous les modules d'ARCHER. Le bond-graph est entré sous sa forme acausale.

L'affectation de la causalité est automatisée avec choix de la modifier sur certains éléments. De plus pour le couplage bond-graph, il est préférable, dans un premier temps, d'associer des bond-graphs acausaux et dans un second temps d'affecter la causalité.(*)

La numérotation des liens se fait automatiquement, dans l'ordre d'affectation causale. Elle peut être à l'écran lors du dessin du bond-graph [BOUAYAD 91]. Elle est juste indicative.

II.4.1. Langage de description bond-graph sous forme texte

Nous proposons un langage adapté à la convention adoptée précédemment pour décrire un bond-graph le plus efficacement possible. La description topologique se fait en 2 parties :

1. On fait état des jonctions J dans le bond-graph, on note chaque jonction avec la liste des éléments 1 port attachés à la jonction J. Lorsque la jonction J n'est attachée à aucun élément 1-port, on ne la note pas. C'est souvent le cas avec les jonctions TF et GY qui sont considérées comme des jonctions. La syntaxe suivante est utilisée :

(*) Voir la discussion sur le couplage à partir de bond-graphs acausaux, paragraphe II.5.1.

'jonction'
'J' 'Liste des éléments (physiques ou bond-graph) attachés à la jonction J'

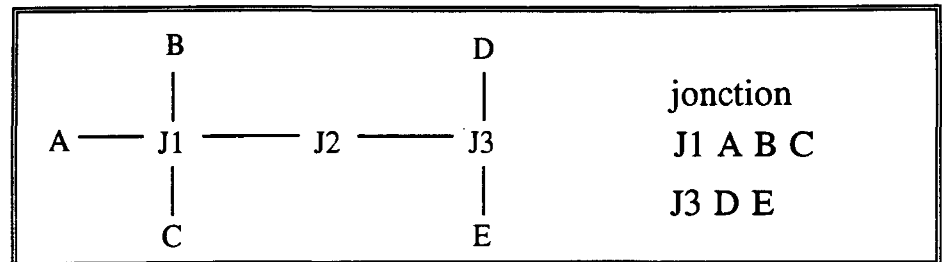


fig 2.82 : Description texte des jonctions d'un bond-graph.

2. On décrit les chemins et boucles formés par toutes les jonctions de la manière la plus naturelle en suivant le sens de la puissance dans les liens inter-jonctions. La syntaxe est la suivante :

'lien'
'Liste des jonctions J formant un chemin orienté dans le bond-graph'

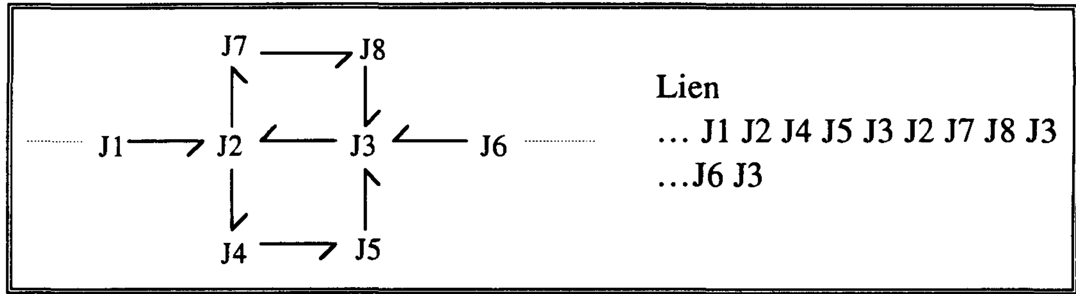
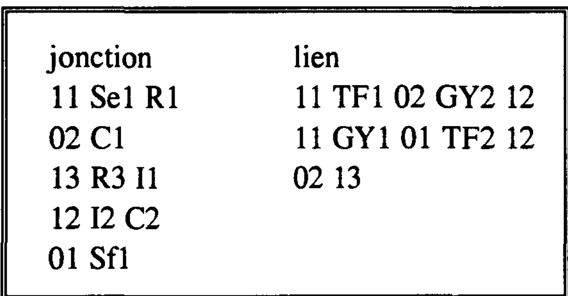


fig 2.83 : Description texte d'un réseau de liens bond-graph.

Exemple :



jonction	lien
11 Se1 R1	11 TF1 02 GY2 12
02 C1	11 GY1 01 TF2 12
13 R3 I1	02 13
12 I2 C2	
01 Sf1	

fig 2.84 : Description texte du bond-graph de la figure 2.81.

Remarque : L'avantage d'une telle description est la séparation, d'une part des éléments attachés aux jonctions et, d'autre part de l'architecture formée par les jonctions et les liens. Ainsi on peut changer rapidement les éléments tout en gardant la même structure du bond-graph. A noter qu'il y a plusieurs façons de décrire un même bond-graph.(*)

(*) Nous verrons au chap III que le code généré est cependant le même.

Nous allons adapter ce langage à l'introduction des blocs bond-graphs en enrichissant la syntaxe présentée en prenant en compte les ports et éventuellement la possibilité de forcer le sens causal sur les ports lors de l'opération de l'affectation de la causalité [AZMANI 91].

II.4.2. Description d'un bloc bond-graph

La définition d'un bloc bond-graph pour un sous-système est composée de deux parties :

- La première est la description du bloc bond-graph (jonctions et liens) lui-même sans les connexions, avec le langage défini plus haut.

- La deuxième contient les informations sur les points de connexions du sous-système, avec la spécification des domaines affectés à chaque port en ***in*** ou en ***out***.

Description des ports :

On commence par écrire la jonction d'attache, puis les ports (***in*** pour input et ***out*** pour output) ; puis la nature des ports (**e**, **mr**, **mt**, **h**). La syntaxe est la suivante :

'port'
'J' 'in ou out' ':' 'Nature du port' ...

Le bloc est construit à l'aide d'un éditeur de texte avec un compilateur autorisant l'introduction d'un BG avec un langage utilisateur.

Par exemple le bloc bond-graph de la figure suivante est constitué d'un port ***in1*** de domaine électrique(e) et de deux ports ***out1*** et ***out2*** appartenant au domaine de la mécanique de rotation(mr).

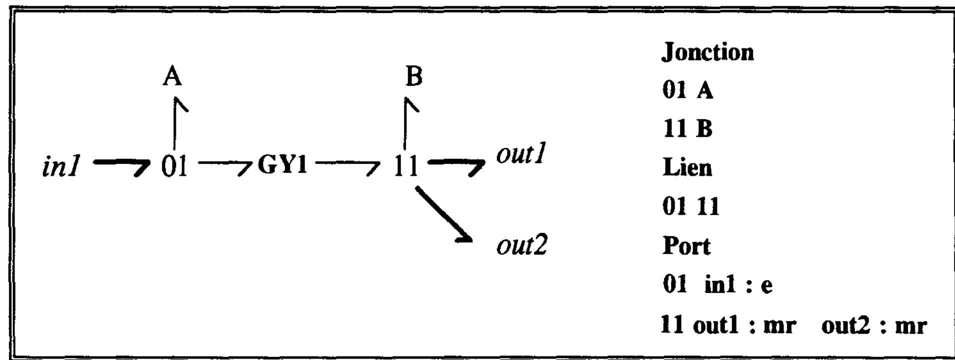


fig 2.85 : Description texte d'un bloc bond-graph.

II.5. Description avec Blocs à plusieurs entrées et sorties

Nous avons, au cours des paragraphes précédents, donné implicitement des éléments de réponse sur un langage de description pour l'introduction des systèmes et des blocs que l'on veut modéliser, notamment sur l'utilisation des blocs et des noeuds dans la description des blocs d'un point de vue topologique. Avant de présenter la syntaxe, nous allons montrer intuitivement le principe du couplage bond-graph, à savoir :

- La façon de "fusionner" les bond-graphs entre eux ?

- L'utilisation du bond-graph causal ou acausal ?

II.5.1. Couplage

II.5.1.1. Avec un bond-graph causal

Pour les exemples, quel que soit le domaine physique étudié, nous employons la notation des éléments bond-graphs. Nous allons prendre le bond-graph de la figure 2.86 et lui faire subir l'opération inverse de simplification sur les liens inter-jonctions.

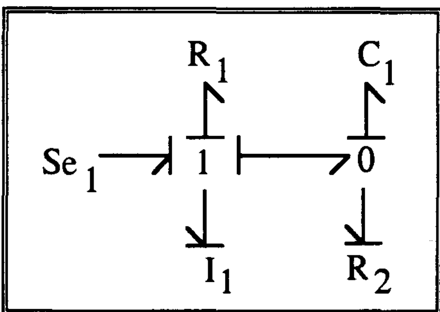


fig 2.86 : Modèle bond-graph causal.

L'idée du couplage bond-graph est de relier les liens entre eux par l'intermédiaire d'une jonction caractérisant l'échange de puissance.

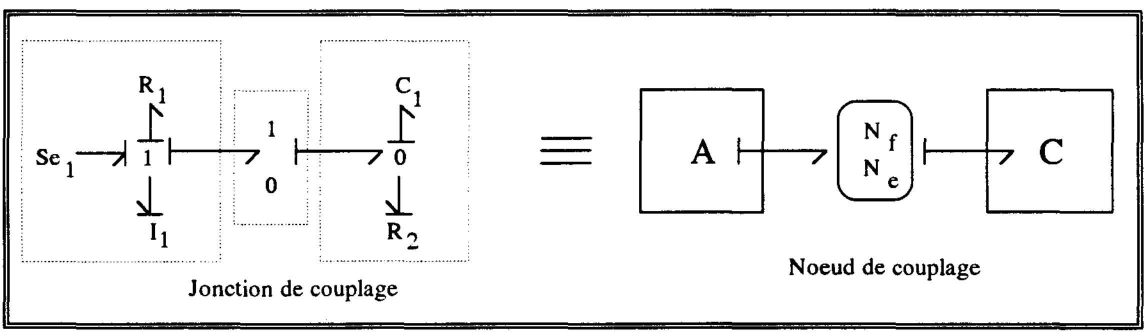


fig 2.87 : Mise en évidence du noeud de connexion.

Le noeud :

Il existe deux variables principales, la variable d'effort et la variable de flux dont le noeud défini plus haut aura deux représentations possibles :

- Un noeud d'effort que nous pouvons symboliser par N_e (Jonction **0**).
- Un noeud de flux par N_f (Jonction **1**).

Remarque : Dans certains cas, le noeud peut être défini comme un noeud de référence qui sera éliminé à la manière des masses électriques.

Les sous-parties **A** et **C** peuvent être considérées comme des blocs indépendants. Pour cela, il suffit de définir la nature des ports, la numérotation des éléments et le nom du bloc.

La figure donne les représentations des blocs bond-graph **A** et **C** qui possèdent respectivement un port en sortie (***out***) de nature électrique (e) par exemple, et un port en entrée (***in***) de même nature. Nous avons gardé la causalité.

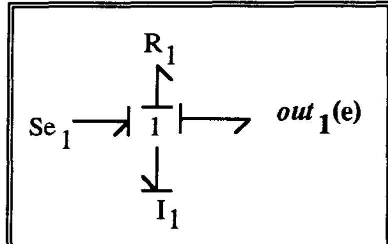


fig 2.88 : Bond-graph causal du bloc A.

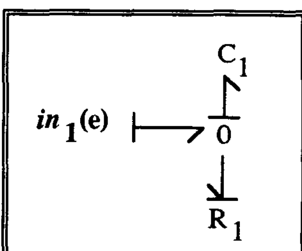


fig 2.89 : bond-graph causal du bloc C.

Afin de pouvoir utiliser, dans une même description plusieurs blocs **A** et **C**, nous utiliserons la notation **B** suivie d'un numéro **n** en indice "$\mathbf{B_n}$". Chaque $\mathbf{B_n}$ appartient à un type de bloc et devra être déclaré avant la description du modèle.

Soit 3 blocs, **B1** de type **A**, **B2** et **B3** de type **C**. Nous voulons relier ces 3 blocs par un noeud de flux **Nf** en gardant la causalité imposée dans chaque bloc comme le montre la figure 2.90.

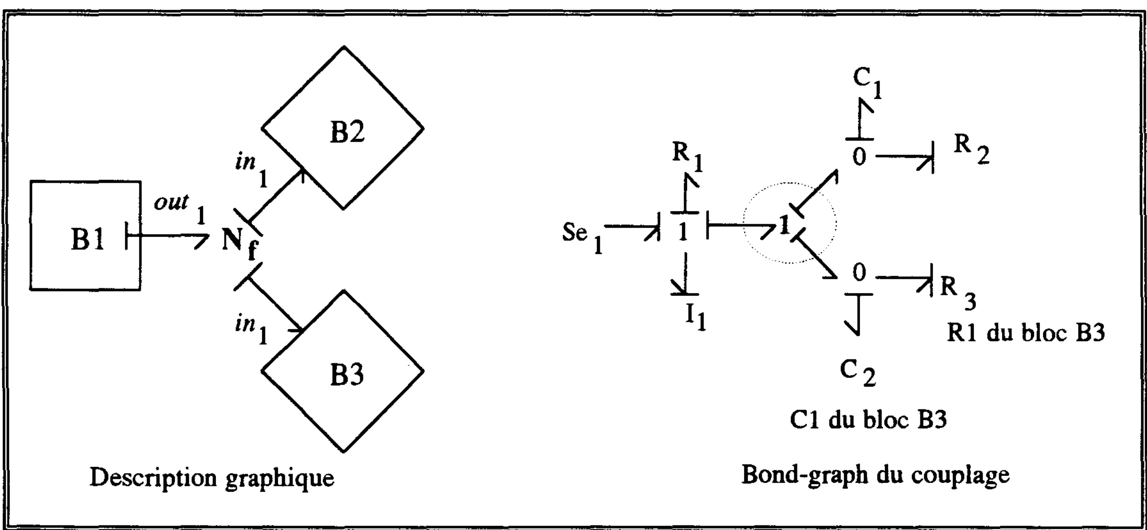


fig 2.90 : Couplage de trois blocs par un noeud de flux.

Dans ce cas, le couplage s'effectue sans conflit causal puisque la jonction **1** accepte un seul lien sans trait causal prés du **1**. Les éléments sont renumérotés pour éviter de les confondre.

Par contre avec un noeud d'effort, un conflit apparaît inévitablement sur la jonction **0** qui, par définition n'accepte qu'un seul lien avec un trait causal prés du **0**.

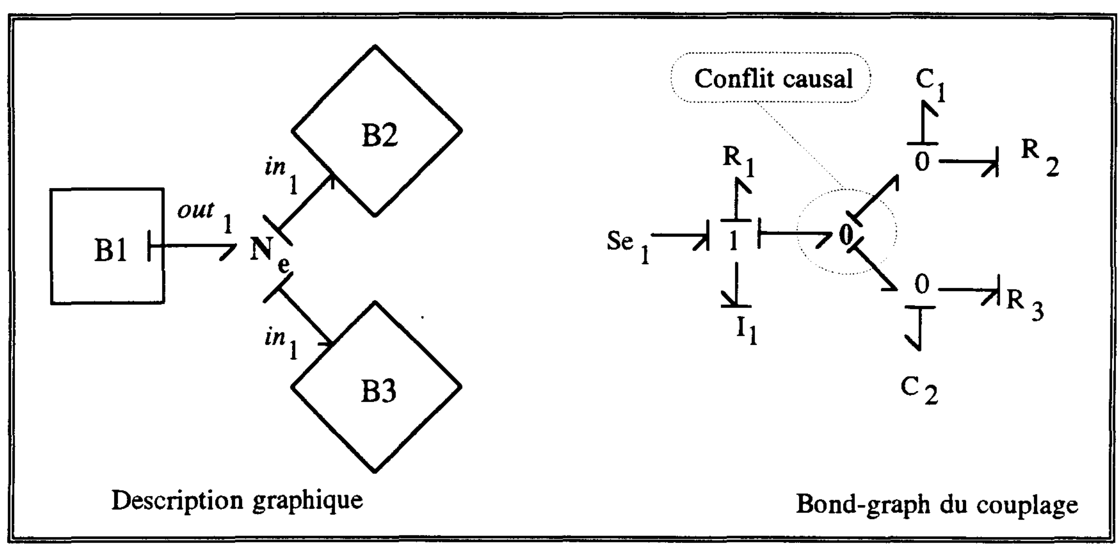


fig 2.91 : Conflit sur le noeud d'effort.

Pour résoudre ce conflit, il faut lancer le module d'affectation de la causalité qui proposera de mettre C1 ou C2 en causalité dérivée, ou d'ajouter des jonctions et des éléments bond-graphs à la structure en conflit, comme le montre la figure 2.92 afin de garder C1 et C2 en causalité intégrale.

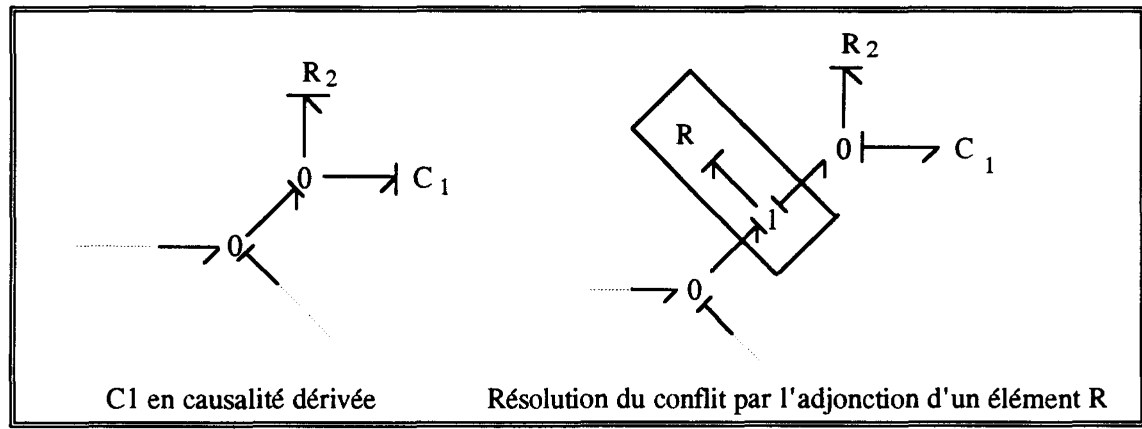


fig 2.92 : Propositions pour la résolution du conflit de la figure 2.91.

Cet exemple montre que, la causalité définie dans les blocs n'est pas respectée lors du couplage dans le bond-graph final. De plus il reste l'étape de la simplification des jonctions qui, habituellement se fait sur un bond-graph acausal.

Si, dans le bloc **B**, nous remplaçons la capacité C par une inductance I en causalité intégrale en gardant la même causalité pour le port ***in₁***, nous obtenons le bloc **C** de la figure 2.93.

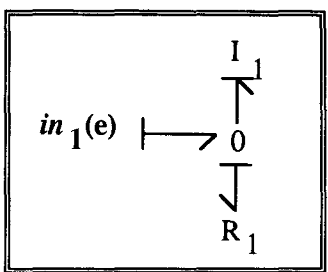


fig 2.93 : Bond-graph causal du bloc C.

Nous reprenons le couplage précédent avec **B2** et **B3** de type **C**. Le conflit, sur la jonction **0** peut être facilement levé par un changement de causalité sur R2 ou R3.

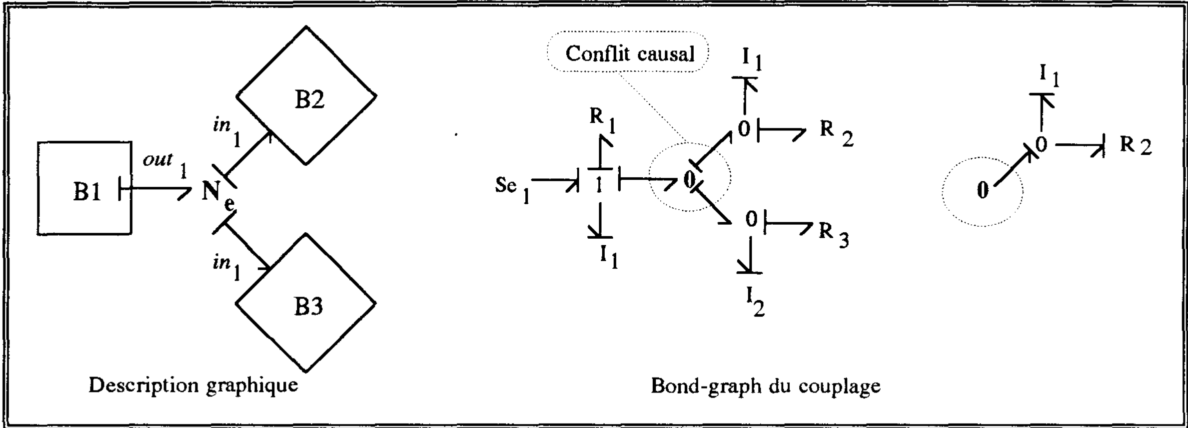


fig 2.94 : Cas d'un couplage avec résolution du conflit par un changement de causalité sur un élément R.

II.5.1.2. Bond-graph acausal

Définir les blocs d'une manière causale amène souvent des conflits sur les jonctions lors du couplage. Comme nous l'avons vu, on peut ajouter des éléments bond-graphs pour lever les conflits et garder la causalité des blocs. Mais dans ce cas, on change la structure du système que l'on voulait construire, avec l'inconvénient d'ajouter des éléments R, C, I qui risquent de compliquer les simulations et peuvent introduire des boucles causales et algébriques lors du calcul des équations mathématiques. En général, la procédure pour la construction d'un bond-graph est la suivante :

- Réticulation du système.
- Repérer les noeuds de vitesse (jonction **1**) ou de tension, pression (jonction **0**).
- Construire l'ossature des jonctions **0** et **1**.
- Affecter les éléments R, C, I, Se, Sf aux jonctions **1** ou **0**.

  Obtention d'un bond-graph ("BG").
- Appliquer les règles de simplification.

  Obtention d'un bond-graph simplifié ("BGS").
- Affecter la causalité.

  Obtention d'un bond-graph causal ("BGC").

Pour le couplage des blocs bond-graphs, nous désirons procéder de la même façon. L'idée est d'employer le bond-graph acausal pour la description, le stockage et le couplage des blocs bond-graphs.

Nous évitons ainsi les faux cas de conflits, pouvant facilement être résolus par le module d'affectation de la causalité, à l'aide de changement de causalité sur des éléments Résistifs. Si d'autres conflits apparaissent, ils seront traités par un dialogue avec l'utilisateur avec les options suivantes.

- Ajouter un élément dynamique,
- Ajouter un élément résistif
- Réimposer la causalité
- Tolérer tous les problèmes

Cette démarche, appliquée à l'exemple précédent est illustrée dans la figure suivante.

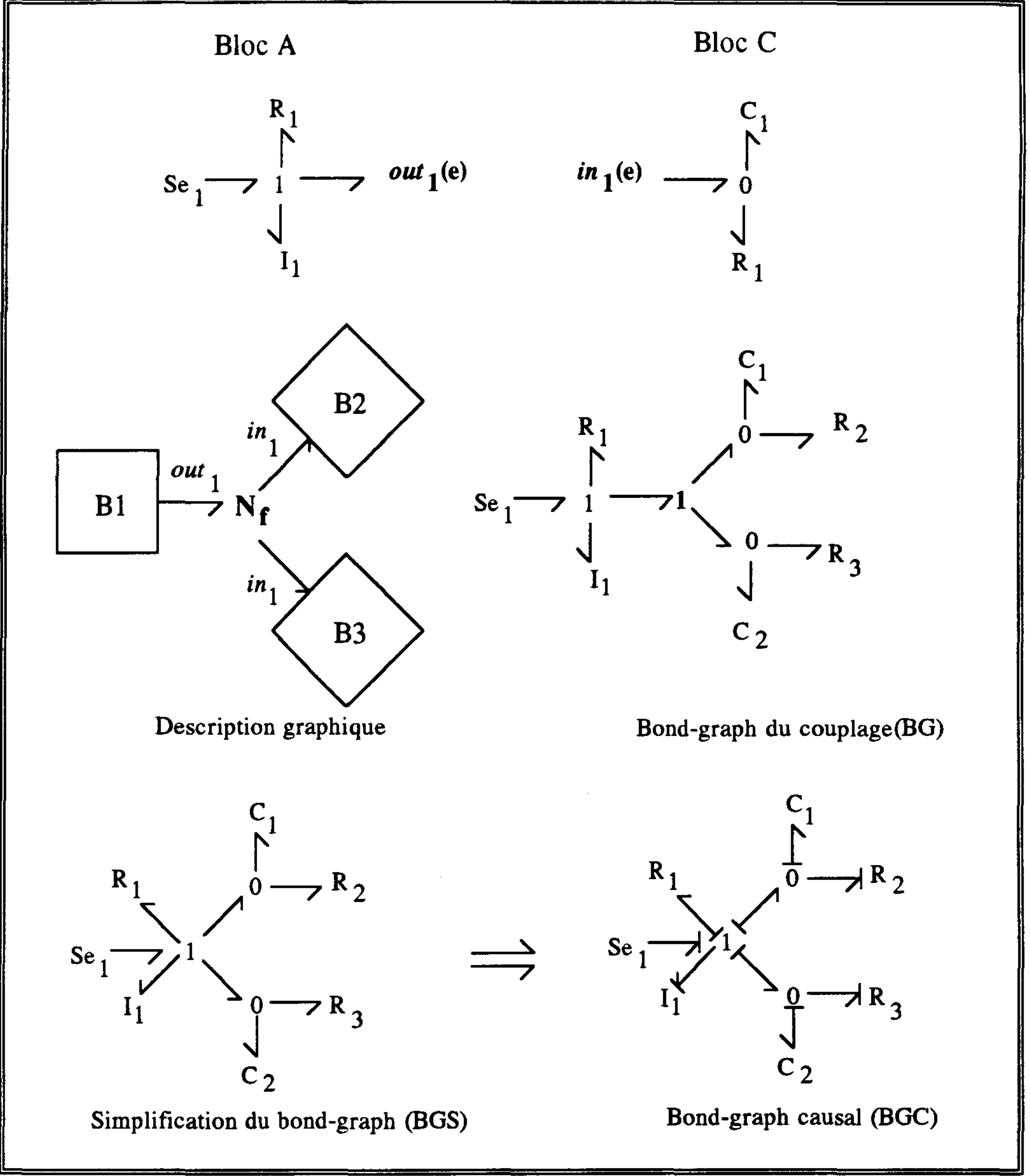


fig 2.95 : Solution du couplage acausal entre les blocs bond-graphs acausaux.

II.5.2. Choix d'une syntaxe adaptée pour le couplage

A travers l'exemple d'une chaîne audio, nous allons définir un langage pouvant supporter ce genre de description. Nous nous intéressons au niveau 2 de la deuxième étape de la procédure de génération d'un modèle, c'est-à-dire le couplage des blocs bond-graphs.(*)

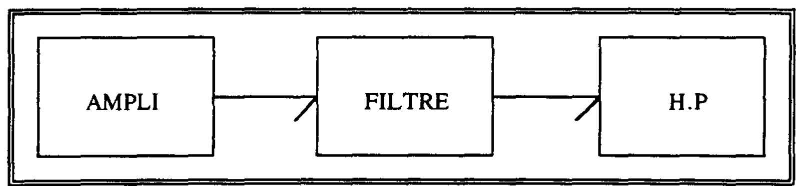


fig 2.96 : Schéma bloc de la chaîne audio.

Dans le paragraphe précédent nous avons introduit la notion de noeud qui symbolise la connexion entre plusieurs blocs. Ici la puissance entre les blocs est transmise par des noeuds d'effort Ne. Chaque noeud caractérise une jonction ou une combinaison de jonctions suivant la nature des domaines physiques des ports liés au noeud.

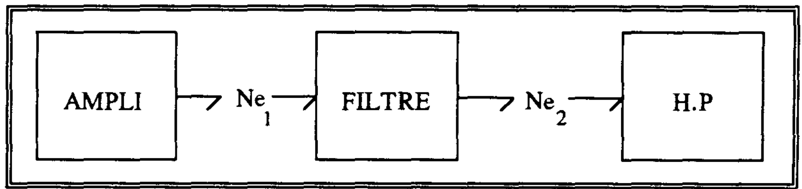


fig 2.97 : Description Blocs & Noeuds de la figure 2.96.

Pour la description texte du modèle de la figure 2.97, l'idée est de scinder la "description" du modèle en deux parties : une partie déclaration et une partie description.

II.5.2.1. La partie déclaration des blocs

Elle fait état de tous les genres de blocs (Nom) présents dans le modèle à décrire. la syntaxe est la suivante :

'dec'

'Bn' '=' 'Nom'

La représentation graphique de la figure 2.97 devient :

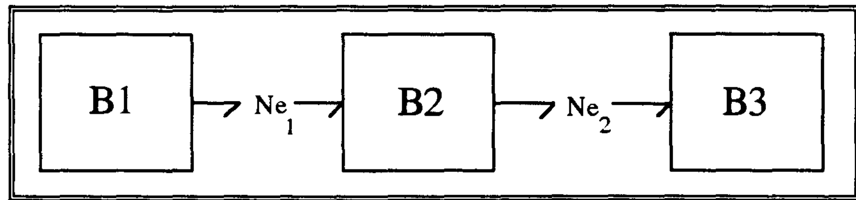


fig 2.98 : Description graphique de la chaîne audio en notation B_n.

(*) Voir le paragraphe II.2.3.2.B.

et la déclaration du modèle de la figure 2.98 est la suivante :

```
dec
B1 = AMPLI
B2 = FILTRE
B3 = HP
```

fig 2.99 : Déclaration du modèle de la chaîne Audio.

Si un type de bloc est présent plusieurs fois dans le bond-graph, on le déclare en ajoutant un bloc supplémentaire avant le signe '='. Les blocs sont considérés comme des objets à part entière.

Par exemple, nous voulons ajouter un bloc Filtre au modèle précédent, il suffit de déclarer un bloc **B4** au niveau de **B2** avec une virgule entre les deux blocs comme le montre la figure 2.100.

```
B2 , B4 = FILTRE  ==>  B2 = FILTRE1
                       B4 = FILTRE2
```

fig 2.100 : Duplication du bloc FILTRE.

II.5.2.2. La partie description d'un réseau Blocs & Noeuds

Après avoir déclaré les blocs, nous décrivons la structure formée par l'assemblage des blocs et des noeuds. Celle-ci s'apparente à un réseau maillé et on la décrit à la manière de la structure de jonction introduite dans le langage bond-graph.(*)

Remarque : Puisque les noeuds **Ne** et **Nf** se comportent de la même façon topologiquement, nous allons employer dans un cas général la notation **N** dans les exemples qui suivent.

Chaque 'chemin' sera décrit par une alternance de Blocs (**B_n**) et de Noeuds (**Ne** ou **Nf**), dans le sens du parcours de la puissance, de façon à ne pas oublier de décrire le plus simplement et le plus complètement possible le modèle.

La syntaxe de la partie description est donc la suivante :

'des'
'Bloc' 'Noeud' 'Bloc'...
'Noeud' 'Bloc' 'Noeud'...

(*) Voir paragraphe II.4.1.

Cela veut dire que dans une description telle que **'B1 Ni B2'** on suppose que le bloc **B1** (resp **B2**) possède au moins un port en sortie (resp en entrée).

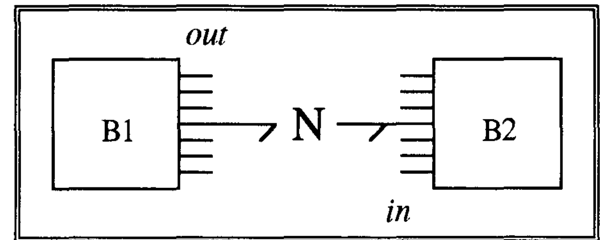


fig 2.101 : Principes de connexion pour les Blocs & Noeuds.

Comme pour le langage bond-graph, il y a plusieurs façons de décrire un réseau de blocs & noeuds. Nous verrons, au chapitre IV, que le code bond-graph généré est cependant le même. On peut faire l'analogie avec le point de connexion.

Par exemple la figure 2.103 montre quelques unes des descriptions qu'il est possible de faire avec le modèle blocs & noeuds de la figure 2.102.

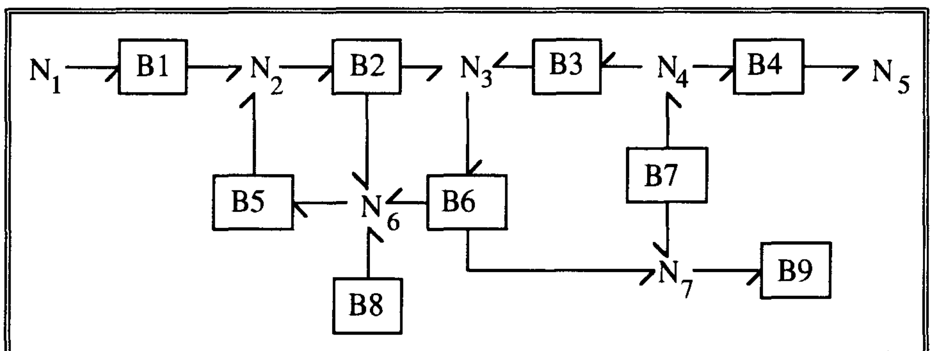


fig 2.102 : Exemple de modèles Blocs & noeuds.

des		des	
N1 B1 N2 B2 N3 B6 N6 B5 N2		N1 B1 N2 B2 N3 B6 N7 B9	
B2 N6		B7 N4 B3 N3 B6 N6 B5 N2	
B8 N6	ou	B8 N6	....
B7 N4 B3 N3		B2 N6	
B6 N7 B9		B7 N7	
N4 B4 N5		B4 N5	

fig 2.103 : Descriptions possibles de la figure 2.102.

La description de la chaîne audio augmenté d'un deuxième bloc Filtre en série avec le premier se fait de la façon suivante :

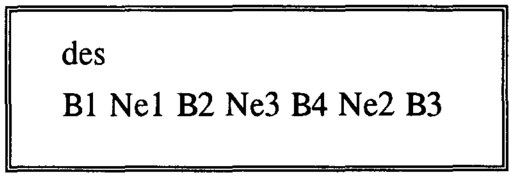

des
B1 Ne1 B2 Ne3 B4 Ne2 B3

fig 2.104 : Partie description de la chaîne audio.

II.5.2.3. Avantages du langage

Un tel langage offre divers avantages dont les principaux sont les suivants :

- Une déclaration des blocs est intéressante au niveau informatique pour le contrôle des erreurs de déclaration et l'existence des blocs qui doivent être pré-définis.

- Elle permet de changer la déclaration des blocs tout en gardant la même structure. Par exemple, nous pouvons reprendre la description de la figure 2.98 en changeant la déclaration des blocs **B1**, **B2**, **B3** :

```
dec
B1 = GENERATEUR
B2 , B3 = RLC
```

fig 2.105 : Autre déclaration pour la description de la figure 2.108.

Cette opération n'aurait pas été possible si le nom des blocs avait été directement utilisé lors de la description comme c'est le cas dans les deux descriptions suivantes.

```
AMPLI 1  Ne1  FILTRE 1  Ne2  HP 1
.....................................
GENERATEUR 1  Ne1  RLC 1  Ne2  RLC 2
```

fig 2.106 : Déclaration en occultant la partie déclaration.

La présence de la notation indicée $\mathbf{B_n}$ associée à une partie déclaration est ainsi justifiée.

- La façon de décrire le système n'est pas imposée à l'utilisateur, celui-ci est libre de conduire sa description en toute liberté en commençant par n'importe quel noeud ou bloc.

II.5.2.4. Sélection des ports

L'utilisateur a donné jusqu'à présent une description du système sans préciser, dans un premier temps, les ports connectés qui le seront ultérieurement par une sélection automatique.

Une fois la description faite, il reste à sélectionner les ports des blocs connectés entre eux par l'intermédiaire des noeuds. Cette sélection peut être abordée de deux façons différentes :

- La première est le passage en revue de tous les noeuds contenus dans la description du modèle. Pour chaque noeud on fait état de tous les blocs qui lui sont attachés en laissant à l'utilisateur la liberté de choisir le port (entrée ou sortie) attaché parmi ceux qui lui sont proposés (figure 2.107).

Remarque : Les ports déjà sélectionnés ne seront plus proposés pour le traitement des autres noeuds.

- La deuxième consiste à conduire la sélection par rapport aux blocs. La méthode est l'inverse de la précédente, c'est-à-dire que l'on passe en revue tous les blocs du modèle. Pour chaque bloc on fait état de tous les noeuds qui lui sont attachés et pour chaque noeud on sélectionne le port adéquat (figure 2.108).

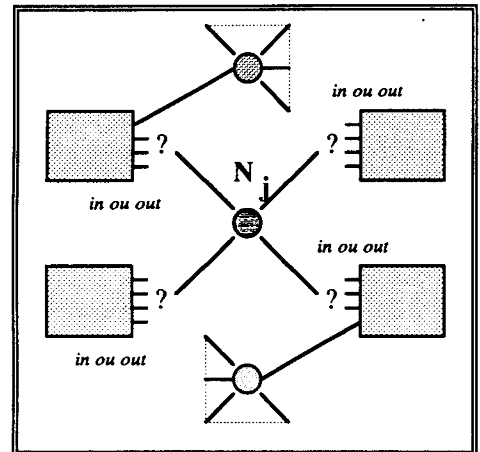


fig 2.107 : Sélection conduite par le noeud.

fig 2.108 : Sélection conduite par le bloc.

De ces deux choix, nous préférons la première qui nous semble plus logique et évidente à mettre en oeuvre. En effet, dans une description, il y a en général plus de blocs que de noeuds. En outre, il est plus facile de repérer les noeuds que les ports des blocs. De plus elle permet un contrôle de la sélection par des conditions d'association qui la rendent plus rapide et efficace.

II.5.2.5. Choix du noeud de couplage

Le but de cette étape est de vérifier, lors de la sélection, si les blocs connectés ont un sens physique. Ce "sens physique" est obtenu en analysant la nature des ports associés à l'aide de la connaissance des domaines physiques des ports.

Lorsque la nature des ports est différente, il est évident que le couplage ne peut s'effectuer convenablement. Dans un bond-graph les jonctions **0** et **1** établissent des relations entre les puissances d'un même domaine physique (voir tableau des variables physiques de la figure 1.18).

Par exemple autour d'un noeud de flux ou d'effort **Ni** le cas de la figure 2.109 ou plusieurs domaines physiques sont en présence (électrique, mécanique de translation, hydraulique, mécanique de rotation) n'est pas correct du point de vue physique :

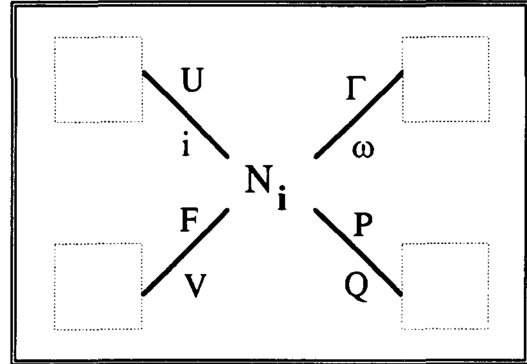


fig 2.109 : Conflit physique sur un noeud Ni.

Il faut changer la sélection, ou modifier les connexions de manière à avoir une homogénéité dans le couplage des domaines.

Pour relever de tels conflits, l'idée est d'établir, lors de la sélection, des tests de compatibilité et un échange de messages. Une mini expertise sera effectuée à l'issue de laquelle l'utilisateur pourra choisir entre plusieurs options et suggestions. Cette expertise se fait parallèlement à la sélection des ports.

Pour cerner le problème, nous allons l'illustrer à travers un exemple où quatre blocs sont couplés à travers un noeud **Ni**. Chaque bloc a un de ses ports (en entrée ou en sortie) attaché au noeud. Chaque port est associé à un domaine physique **Di** comme le montre la figure 2.110.

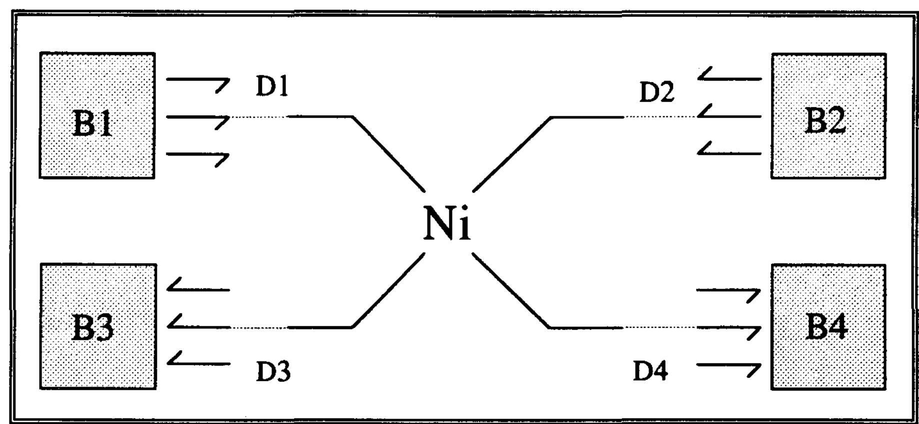


fig 2.110 : Etat du noeud Ni après la sélection des ports.

La solution consiste à placer entre le bloc et le noeud **Ni** un élément transducteur (**TF** pour transformateur ou **GY** pour gyrateur) avec les lois caractéristiques suivantes :

$$\xrightarrow[f1]{e1} TF \xrightarrow[f2]{e2} \qquad (t)$$

$$\begin{cases} e1 = t \cdot e2 \\ f2 = t \cdot f1 \end{cases}$$

$$\xrightarrow[f1]{e1} GY \xrightarrow[f2]{e2} \qquad (g)$$

$$\begin{cases} e1 = g \cdot f2 \\ e2 = g \cdot f1 \end{cases}$$

fig 2.111 : Relation mathématique du transformateur et du gyrateur.

Les transducteurs peuvent être considérés comme des blocs bond-graphs constitués soit d'un élément transformateur, soit d'un élément gyrateur, avec un port en entrée et un port en sortie affectés des domaines physiques adéquats.

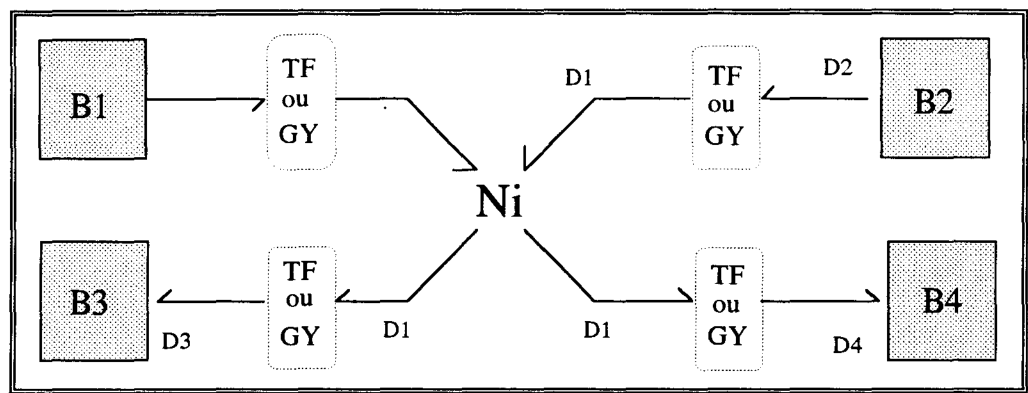


fig 2.112 : Couplage par des transducteurs TF ou GY.

L'échange de puissance entre certains domaines se fait souvent par l'intermédiaire du même transducteur. Par exemple entre les domaines électriques et mécaniques, on observe souvent l'utilisation d'un gyrateur pour transmettre la puissance, entre les domaines mécanique et hydraulique c'est un transformateur...

Le but est de construire à partir de ces observations, une base de données consultable par l'utilisateur pour lui suggérer un transducteur en fonction des domaines en présence. La proposition d'ARCHER est juste indicative, l'utilisateur peut en fait opter entre un transformateur ou un gyrateur selon la loi physique qu'il désire montrer.

La gestion de ce dialogue sera traitée au chapitre IV.

II.6. CONCLUSION

Dans ce chapitre, nous avons proposé pour ARCHER une méthode qui à travers la description d'un modèle physique permet de construire son bond-graph associé.

Pour cela, nous avons proposé les notions de bloc bond-graphs, de ports de connexions, et les notions de noeuds de couplage qui focalisent les échanges énergétiques entre les blocs. Suivant le domaine physique et le nombre de ports pour chaque bloc, nous avons élaboré des langages différents.

Lorsque tous les blocs possèdent une entrée, une seule sortie et qu'ils appartiennent aux mêmes domaines physiques, le modèle est décrit à l'aide des opérateurs série et parallèle.

Dans les autres cas, où les blocs possèdent plusieurs ports en entrées et en sorties, et que les domaines physiques sont le plus souvent différents, on utilise le langage des blocs et noeuds. L'originalité de ce langage réside dans la séparation du modèle en trois parties :

- la partie déclaration pour recenser les types de blocs présents,
- la partie description à l'aide des blocs et des noeuds sans indiquer le numéro des ports attachés aux noeuds relativement à chaque bloc,
- la partie sélection automatique des ports qui, à l'aide d'un dialogue avec l'utilisateur, isole les conflits "physiques" entre les ports des blocs et propose des solutions. Ce procédé permet de décrire la plupart des structures existantes.

Il est possible de mélanger le langage série & parallèle avec le langage blocs & noeuds à travers la notion de pseudo-bloc. Elle consiste à transformer une description série et parallèle en un bloc afin de se ramener toujours à une description par le langage Blocs & noeuds.

Au final, l'association de ces blocs doit aboutir à la génération d'un modèle bond-graph acausal unique, avec une notation des éléments qui permet d'une part de les identifier et d'autre part de connaître leur provenance, c'est-à-dire le bloc d'origine.

Si le modèle bond-graph est connu, nous pouvons le décrire directement par un langage texte qui autorise également la définition de blocs bond-graphs.

Jusqu'à présent, nous avons simplement défini les langages en présentant d'une manière non formelle leur syntaxe respective.

Les principes qui conduisent à la génération d'un bond-graph. à partir des langages proposés seront présentés sous la forme d'un formalisme plus rigoureux à travers les notions de grammaires et d'analyses qu'il est possible d'effectuer sur ces langages.

Le but recherché est une implantation informatique du processus de génération d'un bond-graph, que nous venons de voir, à partir d'une description texte. Le tout compose le compilateur d'ARCHER et celui-ci fera l'objet des deux prochains chapitre qui auront une orientation plus informatique.

CHAPITRE III

CONSTRUCTION D'UN COMPILATEUR POUR LES LANGAGES DE DESCRIPTION SOUS ARCHER.

CONSTRUCTION D'UN COMPILATEUR POUR LES LANGAGES DE DESCRIPTION SOUS ARCHER

III.1. Introduction

Dans ce chapitre, nous allons exposer le principe algorithmique et l'implantation informatique du compilateur d'ARCHER générant un modèle bond-graph à partir d'une description texte d'un système physique. ARCHER a été initialement conçu autour des techniques de l'Intelligence Artificielle traduite à travers un langage basé sur la logique des prédicats : Prolog.

Dans le respect des travaux réalisés autour de ce concept de programmation d'ARCHER, nous avons choisi de présenter nos méthodes dans un formalisme Prolog.

Pour cela nous allons commencer par rappeler succinctement les principales caractéristiques et les différentes formes de représentations de ce type de programmation. Nous présentons ensuite, le codage d'un bond-graph à partir des prédicats de Prolog.

Nous verrons au chapitre 4, comment ce code va être généré par le compilateur que nous présentons avec l'avantage d'être unique pour les langages de descriptions utilisés.

Dans les autres parties, nous développons les différentes phases de notre compilateur. Nous allons insister sur les parties communes aux différents langages de descriptions d'un système physique sous ARCHER. Dans ces parties communes, nous trouvons :

- Un analyseur lexical dont le rôle est de vérifier si une entité donnée, de la ligne étudiée, correspond aux mots autorisés par le langage.

- Un analyseur syntaxique qui vérifie si l'ordre des entités correspond à la grammaire du langage.

- Un analyseur sémantique et plus particulièrement une des phases qui le compose et que nous avons nommé post-sémantique car elle intervient sur le code bond-graph acausal généré après l'analyse pré-sémantique qui dépend du langage proprement dit. La finalité de cette phase est de repérer et de signaler d'éventuelles anomalies du code bond-graph lié à une description erronée du système étudié.

III.2. Représentations symboliques des données d'ARCHER

ARCHER est basé sur la méthodologie bond-graph. Comme c'est une représentation graphique, il était naturel de choisir à l'origine du projet un langage capable de manipuler facilement une telle topologie. Ce langage est le Turbo-Prolog. Depuis, ARCHER s'est enrichi par l'ajout de divers modules, tous développés en langage Turbo-Prolog pour des raisons de compatibilité entre les données.

Avant de présenter la forme des données sous ARCHER, nous allons rappeler les principes de bases de Prolog utiles pour la bonne compréhension de ce chapitre à travers la représentation des graphes.

III.2.1 Outils informatiques

III.2.1.1. PROLOG

Initialement conçu par A. COLMERAUER et son équipe de l'université de Marseille en 1972, **PROLOG** repose sur l'idée d'utiliser la **LOG**ique comme langage de **PRO**grammation. A partir d'un corps de faits et de règles sur ces faits, la machine doit être capable de trouver la solution. [GIANNESINI & AL 85] [CONDILLAC 86]

Prolog est un langage déclaratif, fondé sur le calcul des prédicats du premier ordre selon le principe de résolution (inférence) dû à A.ROBINSON. Cela signifie que le programmeur énonce des règles ou des assertions écrites sous la forme de clauses "*conclusion(s)* si *condition(s)*" dites clauses de Horn.

Un fait :

Est une clause de Horn sans condition.

Par exemple nous pouvons avoir les deux faits suivants,

Le domaine est électrique si rien.

'I' est un élément physique si rien.

applicables à la clause suivante,

L'élément E est une source de courant si ***E est un élément physique*** et *le domaine est électrique* et ***E égal 'I'.***

Lorsqu'une question est posée, l'interpréteur de PROLOG essaie de prouver successivement les termes de l'expression. Pour cela il les recherche en conclusion des règles présentes, ce qui l'amène à rechercher la preuve des prémisses correspondantes.

S'il n'y a pas de variable dans la question, la réponse sera oui ou non. S'il y a des variables, l'interpréteur fournira toutes les combinaisons de valeur qui rendent l'expression en question vraie par le principe d'unification qui consiste à leur appliquer la même substitution pour obtenir le même résultat. L'algorithme interne suit une stratégie fondée sur l'emploi du principe de résolution.

La résolution peut se décrire comme une recherche dans un arbre où, à chaque noeud, des choix peuvent être laissés en attente. En fonctionnement normal, tous les choix seront tentés dans l'ordre de leur occurrence, puis en cas d'échec, une remontée dans l'arbre (backtracking) a lieu vers les différents choix en instances. Il s'agit d'une stratégie en profondeur où l'on explore complètement une voie avant d'en explorer une autre.

L'un des attraits de Prolog réside dans l'intégration complète d'une structure de données extrêmement utilisée en intelligence artificielle, la liste. Une liste est formée d'éléments qui se suivent, et qui peuvent être ou non eux-mêmes des listes. La liste vide est considéré elle même comme une liste communément appelé 'nil'.

Une autre force importante de Prolog, est la séparation de la base de connaissances regroupant l'ensemble des données élémentaires permettant de décrire un problème, avec la partie raisonnement ou moteur d'inférence construit autour d'une base de règles. Les faits de la base de connaissance peuvent être stockés dans un fichier qui peut être enrichi de nouveaux faits par l'application des règles.

Prolog est un langage d'une conception originale qui change les habitudes de programmation. Le formalisme relativement simple permet à l'utilisateur d'énoncer, de façon assez naturelle, le problème. La réflexion se porte essentiellement sur la pertinence et l'adéquation des règles, la part algorithmique étant automatisée.

Cependant, le principe de résolution, remarquable sur le plan théorique entraîne rapidement des temps prohibitifs lorsque le nombre de possibilités est important.

III.2.1.2. TURBO PROLOG

Pour remédier à ce problème de rapidité, BORLAND a introduit en 1986 Turbo-Prolog 2.0 qui est une implantation du Prolog d'Edimbourg qui a donné à ce langage sa syntaxe actuelle.

En Turbo Prolog on traduit chaque assertion par des faits, et le **'si'** par le symbole **':-'**. La conjonction **'et'** est représentée par une virgule **','** et la conjonction **'ou'** par un point virgule **';'**. Chaque clause se termine par un point **'.'**.

Par exemple la clause suivante :

conclusion si *condition1* et *condition2* ou *condition3*

se traduit ainsi en Turbo Prolog par :

conclusion :- condition1 , condition2 ; condition3.

Il est préférable de l'écrire sous la forme suivante :

conclusion :- condition1 , condition2.
conclusion :- condition3.

Ainsi les faits cités plus haut se décrivent ainsi :

domaine('électrique').
élément_physique('I').
identification(E , 'source de courant') :- élément_physique('I'), domaine('électrique'), égal(E = 'I').

Conclusion :

La grande différence, avec les autres PROLOG, réside dans la nécessité de déclarer obligatoirement le type des caractères, les prédicats utilisés dans les clauses, les data-bases... à la manière des langages plus procéduraux.

Nous n'avons non plus un interpréteur, mais un compilateur avec les avantages suivants :

- Une grande vitesse d'exécution, des programmes créés avec TURBO-PROLOG, liée à la compilation du langage.

- Un environnement de compilation qui permet, à travers un éditeur puissant, de gérer un ensemble de fenêtres permettant l'exécution, la trace de mise au point, le dialogue avec le système d'exploitation, un diagnostic des erreurs de compilation etc... sans quitter Turbo-prolog.

- Un environnement d'exécution soignée, avec des primitives et des prédicats prédéfinis en nombres importants et qui permettent : la gestion de fichiers, la gestion de fenêtres, l'accès à un ensemble complet de fonctions mathématiques, la gestion du graphique...

A noter que, parallèlement à la version du Prolog Borland, se sont développés des Prologs sur d'autres systèmes tels que le Macintosh, le système UNIX, et autour de l'environnement Windows depuis son apparition sur les PC dans les années 90.

III.2.1.3. La liste

La liste est la structure de données privilégiées pour le traitement symbolique. Elle est sollicitée dans de nombreux algorithmes, c'est pourquoi nous lui consacrons un sous-paragraphe. La liste est un terme structuré que l'on représente en Turbo Prolog à l'aide des symboles '[' et ']'. Ainsi la liste suivante :

L = [a , b , c , d]

est équivalente au terme structuré suivant

liste(a , liste(b , liste(c , liste(d))))

qui peut être associée à une représentation graphique à travers un arbre binaire dont chaque feuille représente un élément de la liste comme le montre la figure 3.1. A remarquer que la liste vide est à considérer implicitement.

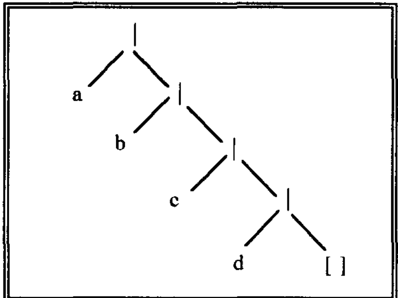


fig 3.1 : Représentation d'une liste par un arbre binaire.

On met en évidence l'opérateur ' | ' qui sépare, dans une liste **L**, le premier élément en tête (T) de la liste avec le restant de la liste appelée queue **Q**. Si nous appliquons le séparateur à la liste **L** précédente, nous obtenons :

[T | Q] avec T = a et Q = [b , c , d].

En instanciant la liste L à Q on peut recommencer le processus et obtenir les autres éléments du système.

Q = [b , c , d] = L ⇔ [T | Q] avec T = b et Q = [c , d]
Q = [b , c , d] = L ⇔ [T | Q] avec T = c et Q = [d]
Q = [d] = L ⇔ [T | Q] avec T = d et Q = [ ]

Pour traduire ce programme en Turbo-Prolog, on utilise la récursivité sur un prédicat ***décompose(Liste)*** qui décompose et affiche les éléments **E** de la liste L.(*)

décompose([ ]).	Condition d'arrêt si la liste est vide
décompose([Tête_de_liste | Reste_de_la_liste]) :-	Extraire le premier élément de la liste
écrire(Tête_de_liste),	Afficher la Tête de la liste
décompose(Reste_de_la_liste).	Appel récursif avec le reste de la liste

A partir de ces manipulations basiques, mais néanmoins puissantes sur la tête ou le reste de la liste, nous pouvons construire de nombreuses fonctions sur les listes qui permettent la concaténation, le renversement, la permutation, le tri sélectif, l'ajout ou la suppression d'élément, etc...

III.2.1.3 Le graphe en PROLOG

Définition : [GONDRAN 79]

Un graphe **G = [X,U]** est le couple de données tel que :

X est l'ensemble des éléments appelés **sommets** ou **noeuds** du graphe.

U est l'ensemble des couples de sommets appelés **arcs** lorsqu'ils sont orientés, ou **arêtes** lorsqu'ils ne le sont pas.

En conséquence le graphe sera dit orienté ou non- orienté.

Si X = N alors le graphe G est dit **d'ordre N.**

Informatiquement, il y a plusieurs façons de représenter un graphe. On peut choisir la forme matricielle, à savoir un tableau d'incidence comme le montre la figure 3.2.

En colonne et en ligne nous avons les sommets du graphe, et les intersections de celles-ci sont valuées, à **un** si un arc existe entre deux sommets orientés selon le sens de lecture du tableau (en général des lignes vers les colonnes), à **zéro** dans le cas contraire. Cette méthode a le désavantage de prendre beaucoup de place en mémoire car les arcs qui n'existent pas dans le graphe sont présents dans le tableau d'incidence.

Par exemple le graphe de la figure 3.2 nécessite 25 informations pour sa représentation.

(*) On met en exergue les propriétés récursives des listes, trés utiles dans la résolution d'un problème appelant un processus répétitif.

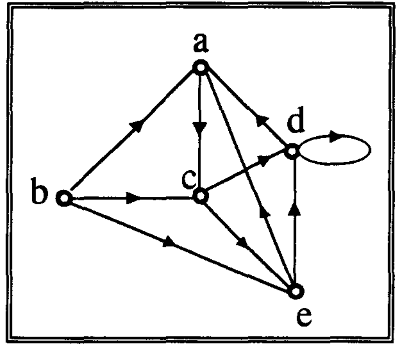


fig 3.2 : Graphe orienté.

	a	b	c	d	e
a	0	1	0	1	1
b	0	0	0	0	0
c	1	1	0	0	0
d	0	0	1	1	1
e	0	1	1	0	0

fig 3.3 : Tableau d'incidence associé au graphe.

Prolog est un langage qui permet de représenter très facilement les graphes. Un arc exprime une relation entre deux sommets. Il est donc assez naturel de représenter un graphe en prolog par des faits de la forme *arc(X,Y)*, énonçant l'existence d'un arc entre les sommets **X** et **Y**. Ainsi, seuls les arcs existants sont représentés, cela facilite le traitement et la gestion de l'information en mémoire.

```
arc ( a , c ).   arc ( c , d ).
arc ( b , a ).   arc ( c , e ).
arc ( b , c ).   arc ( d , a ).
arc ( b , e ).   arc ( d , d ).
arc ( e , a ).   arc ( e , d ).
```

fig 3.4 : Description du graphe de la figure 3.2 par le fait *arc*.

Cette notation est propice à toutes les opérations liées aux graphes comme la recherche des chemins, des boucles, d'arbres, etc. (*)

Par exemple, le prédicat ***chemin(X,Y)*** sera évalué à vrai s'il existe un chemin entre *X* et *Y*. Le problème à traiter se décompose en deux cas qui donneront naissance à deux clauses :

Une première clause traite du cas où le chemin est élémentaire : Il y a un chemin entre deux points s'il existe un arc qui les relie.

La seconde clause traite le cas où le chemin n'est pas élémentaire, il y a un chemin de *X* à *Z* et un chemin de *Z* à *Y*. Le programme s'écrit :

```
chemin (X,Y) :- arc(X,Y).
chemin(X,Y) :-
        arc(X,Z),
        chemin(Z,Y).
```

(*) Utilisées principalement lors de l'affectation de la causalité, du calcul des chemins causaux et du dessin du bond-graph à l'écran.

Selon que les arguments sont libres (majuscules) ou instanciés (minuscules), le programme ***chemin(X,Y)*** donne des résultats différents.

Nous touchons là à une des forces majeures de prolog, qui est l'utilisation d'un même programme pour diverses tâches.

Si le premier argument est laissé libre, ***chemin(X,d)*** va donner en résultat, l'ensemble des sommets ***X*** du graphe à partir desquels il existe un chemin jusqu'à ***d***.(*)

Si le second argument est laissé libre, ***chemin(a,X)*** donne alors tous les sommets ***X*** qui sont accessibles à partir de ***a***.(**)

Enfin, si les deux arguments sont libres, Prolog va donner tous les chemins existants dans le graphe, ***chemin(X,Y)***.

III.2.2. Codage bond-graph sous Archer

III.2.2.1. Le bond-graph et le graphe

Comme nous l'avons souligné, le bond-graph acausal peut être considéré comme un graphe orienté. Un sommet représente soit une jonction, soit un élément 1-port ou multi-port et un arc caractérise un lien orienté selon la demi-flèche, il relie deux sommets.

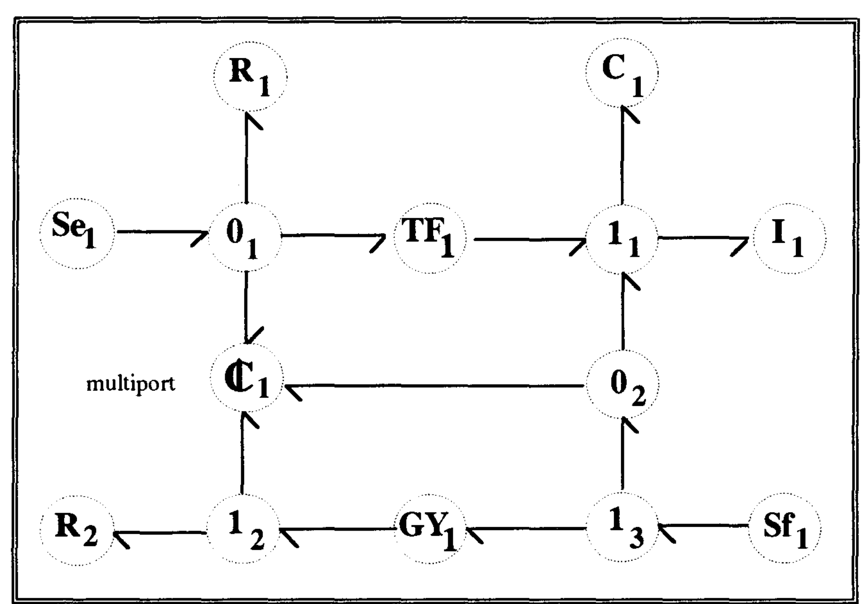


fig 3.5 : Analogie entre un bond-graph et un graphe orienté.

(*) La forme des entrées-sorties pour le prédicat *chemin* est (s,e).

La notion de modèle d'entrée-sortie est une caractérisque très importante, propre à Turbo-Prolog et liée à sa nature de langage compilé. A chaque paramètre d'un prédicat, est associé un ou plusieurs attributs qui indiquent s'il s'agit soit d'un paramètre d'entrée (valeur instanciée à l'appel du prédicat), soit d'un paramètre de sortie (valeur retournée par le prédicat).

(**) le prédicat *chemin* est pris dans sa forme (e,s).

Nous pouvons donc décrire un bond-graph par le fait ***arc(X,Y)*** présenté précédemment. Ainsi le bond-graph de la figure 3.5 est décrit de la manière suivante :

arc(Se1 , 01). arc(TF1 , 11). arc (13 , GY1). arc(02 ,C 1).
arc(01 , TF1). arc(11 , I1). arc(01 ,C 1). arc(Sf1 , 13).
arc(01 , R1). arc(12 , R2). arc(TF1 , 11). arc(02 ,11).
arc(02 ,C 1). arc(GY1 , 12). arc(11 , C1).

fig 3.6 : Description du graphe de la figure 3.5 par le fait *arc*.

Il est possible d'optimiser l'écriture des faits par les remarques suivantes liées à la construction et à la définition même du bond-graph.

- En effet, les arcs, reliant deux jonctions **J1** et **J2**, sont orientés suivant le sens de la puissance dans le bond-graph. **J1** et **J2** sont choisis parmi les jonctions **0** et **1** :

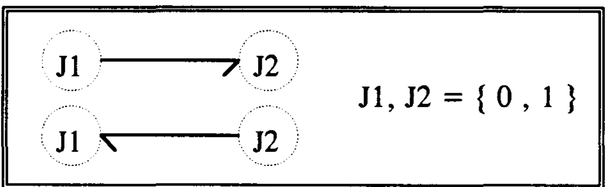


fig 3.7 : Analogie des jonctions et des sommets.

- Les jonctions **TF** et **GY** sont des sommets n'acceptant que deux arcs, un entrant et un sortant :

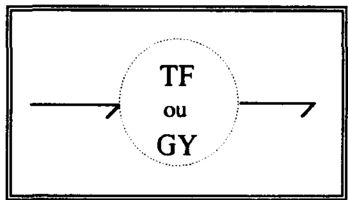


fig 3.8 : Analogie du transducteur avec un sommet à deux arcs.

- Les éléments 1-port et multi-ports (R, C, I) sont considérés comme des sommets n'acceptant qu'un ou plusieurs arcs entrants :

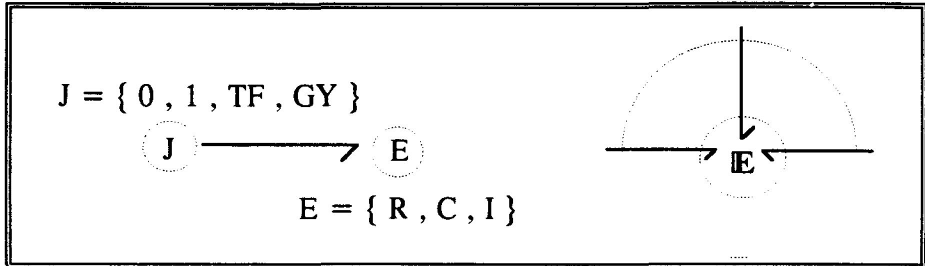


fig 3.9 : Analogie du multi-port avec un sommet.

- Un élément 1-port est toujours relié à un seul sommet jonction.

- Les éléments sources 1-port (Se et Sf) sont des sommets n'acceptant qu'un seul arc sortant :

J = { 0 , 1 , TF , GY }

E ⟶ J

E = { Se , Sf }

fig 3.10 : Analogie d'une source avec un sommet ayant un arc incident.

Conclusion :

Le bond-graph est avant tout un graphe dont l'écriture peut être réorganisée en 2 classes. (*)

La première regroupe les sommets jonctions avec la liste des éléments qui lui sont attachés. Le multi-port sera englobé dans cette classe.

La seconde traduit la structure des sommets jonctions par le sens de circulation de la puissance.

La causalité sera considérée comme une information supplémentaire sur le parcours de la variable effort ou flux.

III.2.2.1. Bond-graph acausal

- Une jonction est représentée par un prédicat à trois paramètres :

jonction (champ1 , champ2 , champ3)(**)

Le ***champ1*** représente la jonction $\mathbf{J_k}$ qui peut être de type 0, 1, TF, GY ou multiport (I, C, R, IC, CR ...) ; **k** est **l'identificateur de la jonction**, il est utile pour le traitement informatique.

Le ***champ2*** est la **liste** des éléments 1-port qui lui sont attachés avec **S** une source de type Se ou Sf et **E** un élément de type I, C ou R.

Le ***champ3*** caractérise un entier **p** égal à 0, 1 ou 2 qui correspond à une convention permettant d'identifier le type du "**bond**" simple ou multiple liant deux jonctions. Lorsque **p = 0** cela signifie que J est une jonction simple.

(*) Voir la présentation du langage bond-graph au chapitre II.

(**) Pour simplifier la notation, nous représentons parfois ce fait par ***j(champ1, champ2, champ3)***.

- Un lien entre deux jonctions **J1** et **J2**, est symbolisé par un prédicat à deux paramètres :

lien (champ1 , champ2) (*)

Le ***champ1*** indique la jonction **J1** de départ et le ***champ2*** la jonction **J2** d'arrivée. **J1** et **J2** peuvent être des multi-ports.

Remarque : Pour une question de facilité, nous écrirons ***j*** et ***l*** à la place de ***jonction*** et ***lien***.

Exemple : la jonction J_k est représentée par le fait $j(J_k , [S_1 , S_2 , E_1 , E_2 , ...], p)$ avec $p = 0$, elle est reliée à la jonction J_h par le fait $l(J_k , J_h)$.

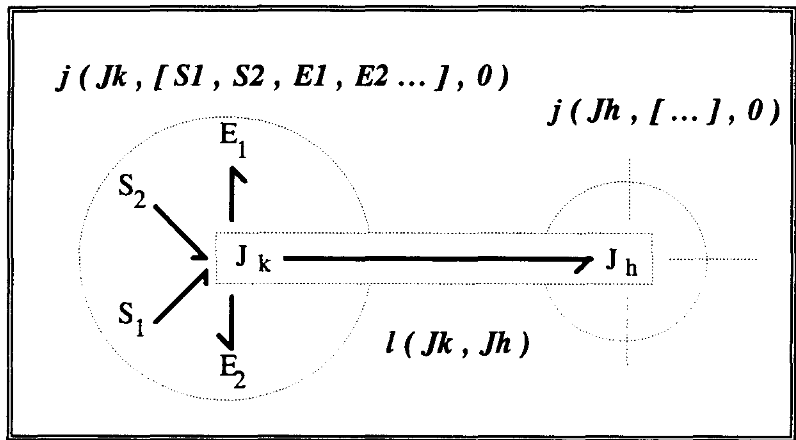


fig 3.11 : Prédicats pour la représentation symbolique des éléments bond-graphs.

Cette séparation, entre les éléments un-port représentés dans la liste d'une jonction et les liens inter-jonctions, permet de mettre en évidence l'ossature du bond-graph. A partir des faits ***lien(_)*** et ***jonction(_)***, il est possible d'effectuer toutes sortes de manipulations sur le bond-graph comme le parcours de graphes nécessaires aux dessins du bond-graph et l'affectation de la causalité, ou entre les bond-graphs pour le couplage à travers un enrichissement du code présenté.

Ainsi, le bond-graph de la figure 3.3 se traduit par les faits suivants :

j(01 , [R1 , Se1])	*j (C1 , [])*	*l(01 , TF1)*	*l (12 , C1)*
j(11 , [C1 , I1])	*j (TF1 , [])*	*l(TF1 , 11)*	*l (GY1 ,12)*
j(02 , [])	*j (GY1 , [])*	*l (02 , 11)*	*l (13 , GY1)*
j(13 , [Sf1])		*l (02 , C1)*	*l (13 , 02)*
j(12 , [R2])		*l (01 , C1)*	

fig 3.12 : Description du bond-graph de la fig 3.5 par les faits *j* et *l*.

(*) Ce fait peut-être représenté par *l(champ1, champ2)*.

III.2.2.2. Bond-graph causal

Le bond-graph acausal est traité par le module qui a en charge l'affectation de la causalité en maximisant le nombre d'éléments bond-graphs en causalités intégrales. [AZMANI 90]. La représentation sous la forme de prédicats jonctions (*j*) et liens (*l*) combinée au moteur d'inférence de Prolog, facilite la résolution des conflits causaux. A l'issue de ce traitement, un ensemble de nouveaux prédicats est généré afin de traduire le trait causal sur les liens.

Ce prédicat ***numéro_effort(champ1 , champ2 , champ3)*** représente la circulation de la variable effort entre le ***champ2*** et le ***champ3***.

Le ***champ1*** est un entier qui indique le numéro du lien joignant les entités de départ (***champ2***) et d'arrivée (***champ3***).

Exemple : ***n_e (N, A, B)***

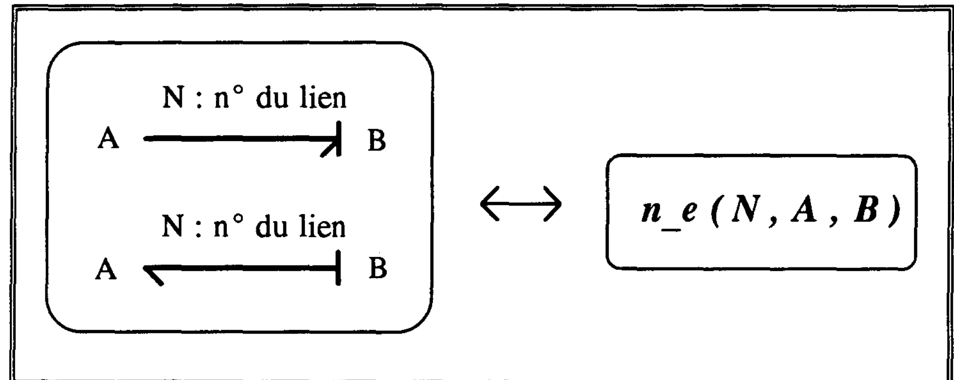


fig 3.13 : Prédicat utilisé pour la causalité sur un lien.

III.2.3. Les Limites du langage Prolog

Le langage Turbo-Prolog est néanmoins limité pour les grands projets. Archer subit actuellement une transformation par les choix d'un langage de programmation et d'un environnement adaptés. Cependant la philosophie reste inchangée. L'aspect objet d'ARCHER sera renforcé par l'utilisation d'un langage orienté objet à part entière.

En effet, les travaux réalisés actuellement autour d'ARCHER ont été conçus dans une conception objet, notamment le travail présenté dans notre mémoire, de manière à faciliter la portabilité des programmes vers des environnements plus conviviaux.

Ainsi la notion de bloc, présentée au chapitre 2, pourra facilement se généraliser à n'importe quel élément du bond-graph constituant ainsi des objets qui peuvent s'auto-gérer.

L'adaptation d'ARCHER à un environnement tel que Windows ne met pas en cause certains modules exécutables réalisés en Turbo-Prolog.

Les modules peuvent être actionnés indépendamment les uns des autres. Connaissant les codes sous lesquels l'information est représentée, nous pouvons les entrer dans un fichier à l'aide d'un éditeur texte classique en respectant la syntaxe des prédicats Prologs. On active ensuite les exécutables voulus.

L'environnement d'ARCHER est un programme qui a en charge la gestion des différents modules présentés au chapitre I. Initialement, cet environnement a été écrit en Turbo-Prolog.

La version développée actuellement sous Windows modifie dans un premier temps uniquement les modules qui faisaient appel aux ressources DOS. C'est-à-dire que l'on crée un environnement dans un langage fonctionnant sous Windows (Pascal, C++...) qui manipule directement les éxécutables des modules existants.

Dans un deuxième temps nous réécrirons, au fur-et-à mesure de l'évolution du projet, les modules à travers un langage similaire à l'environnement d'ARCHER pour exploiter à fond les possibilités de fenêtrage qu'offre Windows.

Cette méthode de travail permet de faire évoluer le projet en utilisant et en s'appuyant sur les travaux déjà effectués, tout en gardant une version opérationnelle du projet.

III.3. Construction d'un compilateur pour ARCHER

Initialement Turbo-Prolog a été conçu pour sa capacité à analyser toutes sortes de langages naturels. La programmation logique permet, en effet, une approche simple des problèmes de transformations de structures d'arbres que constitue, par exemple, la compilation d'un langage de programmation de haut niveau. On passera d'un arbre plat (la chaîne d'entrée) à un arbre complexe représentant la structure sémantique du programme compilé.

Dans cette section, nous allons rappeler les notions qui nous seront nécessaires pour l'implémentation d'un "langage de description de système physique", afin de créer un code bond-graph compréhensible par les modules d'ARCHER.

"Implémenter un langage", veut dire créer un outil permettant d'utiliser ce langage. Cet outil est constitué par plusieurs modules qui, dans leurs principes, doivent être indépendants du matériel utilisé [NOYELLE 88].

Cet outil est appelé un compilateur et il se compose :

- D'un analyseur lexical, dans lequel on découpe le texte du langage source (lu de gauche à droite à partir d'un fichier) en une liste de symboles nommés lexèmes,

- D'un analyseur syntaxique, établissant la structure du langage à travers des règles de syntaxe ou grammaire, et générant en même temps une forme intermédiaire de langage, de manière à ce que le générateur puisse créer du code efficace.

- D'un analyseur sémantique et d'un générateur de code, donnant un sens aux instructions du langage intermédiaire et fabriquant le code final. Il est souvent accompagné d'un optimiseur qui tente de simplifier la forme intermédiaire sans en changer la sémantique.

Remarque : L'analyse lexicale est une phase commune à tous les langages analysés, alors que les deux suivantes dépendent, d'une part de la grammaire et d'autre part, de la nature du code à générer et sont donc différentes d'un cas à un autre.

Il est très intéressant d'utiliser Prolog pour décrire l'ensemble de ce processus, en raison de la structure même du compilateur.

En effet, Prolog peut être utilisé comme formalisme d'un analyseur syntaxique et du composant qui calcule la représentation sémantique de la phrase analysée, aussi bien que comme formalisme de cette représentation formelle.

Nous allons montrer la fabrication d'un compilateur développé sous Prolog à travers le langage bond-graph défini au chapitre II.

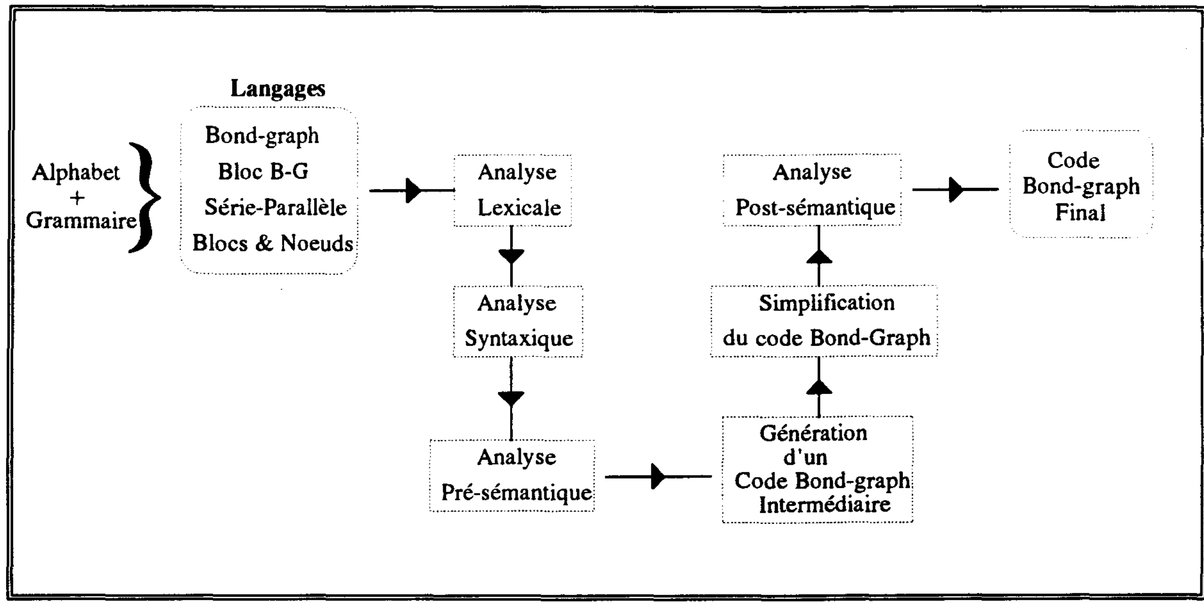


fig 3.14 : Organisation du compilateur d'ARCHER.

III.3.1. Analyseur lexical

C'est un module dont le rôle est d'isoler les éléments syntaxiques ou lexèmes composant un programme écrit dans un langage donné, et de supprimer certaines parties non significatives (pour le compilateur) comme les commentaires, les espaces multiples et autres caractères de mise en page.

Un analyseur lexical peut se présenter sous la forme d'un sous-programme rendant le lexème à chaque appel ou sous la forme d'un pré-processeur, créant une chaîne de lexèmes débarrassée des caractères indésirables (commentaires, espaces, tabulations...) et qui sera utilisée comme source par l'analyseur syntaxique.

Voir en **Annexe 1**, un rappel sur la grammaire et les langages.

III.3.1.1. Décomposition d'une ligne

Pour obtenir les différents lexèmes constituant une ligne, nous allons créer une fonction qui permet d'extraire tous les lexèmes d'une ligne et de les traiter un par un.

Cette fonction est le prédicat ***lexical***, qui par récursivité permet de découper la chaîne de caractère Ligne en lexèmes. A chaque appel on extrait le lexème et on le traite par le prédicat ***traiter_lexème*** qui l'analyse en le comparant aux éléments d'un dictionnaire des types de lexèmes prédéfinis et autorisés dans le langage.(*)

lexical(' ').	Arrêt si la chaîne est vide
lexical(Ligne) :-	Appel principal du prédicat
extraire_lexème(Ligne, Lexème, Reste_de_la _ligne),	Extraction d'1 lexème
traiter_lexème(Lexème),	Traitement d'un lexème
lexical(Reste_de_la_ligne).	Appel récursif

Par exemple dans la ligne ' **m1 + R1 + C1 / R2 + a1** ', tous les lexèmes présents font partie du langage étudié.

(*) En Turbo-Prolog, l'outil privilégié pour l'analyse lexicale est le prédicat ***fronttoken(X,T,Z)*** qui permet de diviser, automatiquement, une chaîne en une liste de "tokens" ou lexèmes. Ce prédicat fonctionne de la manière suivante si nous l'employons avec le modèle (e,s,s). La première entité lexicale de la chaîne ***X*** est retournée dans ***T***, alors que le reste de la chaîne est instancié à ***Z***.

Par contre dans la ligne suivante ' **I1 * R1 POMPE + C1 Se1** ', les lexèmes '**POMPE**' et '**Se1**' ne font pas partie du dictionnaire des éléments autorisés, dans le cas où le système étudié est électrique. De plus le symbole '*' ne fait pas partie des lexèmes opérateurs autorisés.

III.3.1.2. Traitement du lexème

La nature d'un lexème est caractérisée par un <métanom> défini par une grammaire autour d'un alphabet. Les règles de combinaisons entre les métanoms produisent la syntaxe du langage.

La Notation de Backus-Naur (NBN) [NOYELLE 88] permet d'écrire, d'une part, la grammaire des métanoms utilisée par le langage et d'autre part, la grammaire du langage lui-même comme nous le verrons plus loin dans l'analyse syntaxique.

Par exemple pour le langage bond-graph, nous avons besoin des métanoms suivants :

<jonction> :: = **<type_jonction> <nombre_entier>**
<type_jonction> :: = **0 | 1 | TF | GY | MTF | MGY | R | C | I**
<élément> :: = **<type_éléments> <nombre entier>**
<point> ::= **<lettre_minuscule> <nombre_entier>**
<type_élément> :: = **R | C | I | Se | Sf |**
<lettre_minuscule> ::= **a | b | ... | z**
<nombre_entier> :: = **<chiffre> | <nombre entier> <chiffre>**
<chiffre> :: = **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |**
<domaine> :: = **e | el | mt | mr | h | bg | ...**
<séparateur> :: = **/ | : | ; | = | ...**

Les grammaires du langage bond-graph et bloc bond-graph seront explicitées dans le chapitre IV.

Le traitement d'un lexème consiste à vérifier sa présence dans une liste pré-établie. En ce qui nous concerne, cette liste est caractérisée par un ensemble de prédicats qui ont la forme suivante :

liste_élément (champ1 , champ2 , champ3 , champ4 , champ5)

Le ***champ1*** représente le domaine physique.
Le ***champ2*** est une liste qui regroupe les symboles physiques (pré-définis ou créés par l'utilisateur) représentant l'élément physique du domaine considéré.
Le ***champ3*** caractérise le symbole universel qui sera utilisé dans les équations.
Le ***champ4*** est l'exposant ***e*** (1 ou -1) appliqué au symbole du champ3 : ***(champ3)***^{e}
Le ***champ5*** est le symbole Bond-graph associé à l'élément physique.

Exemple :

En mécanique, le ressort peut avoir plusieurs appellations.

liste_élément (mécanique, ([Re, Ressort, Rs, Spring ...], K, -1, C)

L'amortisseur

liste_élément (mécanique, ([Amort, Am, A, Damper ...], b, 1, R)

Source de tension

liste_élément (électrique, ([E , SE, U, Tension, Voltage ...], E, 1, Se)

Réservoir

liste_élément (hydraulique, ([Réservoir, Rés, tank ...], C, 1, C)

Pendant le traitement d'un lexème, plusieurs possibilités sont à considérer :

Le lexème appartient à la liste précédente, dans ce cas on poursuit l'analyse. Le lexème n'appartient pas à cette liste, un message signale à l'utilisateur l'erreur et lui propose soit de la corriger, soit d'ajouter l'élément trouvé à la liste, soit d'arrêter l'analyse.

1. La correction consiste à la proposition d'une liste d'éléments déterminés à partir des éléments de la liste pré-établie qui se rapproche orthographiquement de l'élément incorrect. Par exemple '**Re1**' proche de '**Ressort1**'.

2. L'ajout d'un élément dans la liste pré-établie, consiste à demander à l'utilisateur d'introduire l'élément bond-graph associé au symbole physique de l'élément physique. Par exemple ajouter '**Re**' qui est le symbole d'un ressort et dont l'élément bond-graph associé est '**C**'.

3. Quand l'utilisateur décide d'arrêter, le programme retourne à l'éditeur de texte en positionnant le curseur sur l'élément (lexème) qui a provoqué cette interruption.

Remarque : Lorsque la description se fait en langage bond-graph, la liste des éléments est connue à l'avance puisque les symboles sont uniques {R, C, I, Se, Sf, TF, GY}, le prédicat ***liste_élément*** est occulté. De même pour les opérateurs { + , / , (,) , = }

En réalité, le traitement d'un lexème, passe par une phase préalable, qui consiste à décomposer le lexème en 2 parties : la première partie caractérise le symbole physique utilisé, la seconde représente l'indice permettant d'identifier sans ambiguité cet élément.

Ceci se fait par l'intermédiaire d'un prédicat qui décompose le lexème à analyser.(*)

décompose_lexème(Lexème en alphanumérique, Partie Alphabétique, Partie numérique) ***(e,s,s)***

Par exemple ***décompose_lexème('Ress12', Alpha, Num)*** va donner '**Ress**' pour la sortie ***Alpha*** et '**12**' pour la sortie *Num*.

La première partie alphabétique du lexème fait l'objet du traitement explicité précédemment (comparaison au dictionnaire).

La partie indice permet d'identifier les éléments de mêmes types et de signaler éventuellement une ambiguïté lorsque plusieurs représentations symboliques d'un même élément possèdent le même indice. En effet, ARCHER produit des modèles mathématiques en expressions formelles utilisant les symboles 'universels' associés à cet élément (déclarés dans le prédicat ***liste_élément***) et l'indice utilisé par l'utilisateur.

Exemple :

Re1	C1	1/K1
Ress1	C1	1/K1
Ressort1	C1	1/K1

Dans ce cas, on signale à l'utilisateur qu'il faut soit différencier ces indices, soit les tolérer. Cette étape peut-être qualifiée de pré-sémantique, car elle analyse la cohérence des indices d'un même élément.

Remarque : Dans le cas où une erreur se produit, il suffit de prévoir le cas d'échec en ajoutant une clause à la règle activée avec un message. Ainsi, pour traiter 1 lexème, nous proposons :

traitement_lexèmef(Lexème) :-
    décompose_lexème(Lexème, Alpha, _), (**)
    dictionnaire(Alpha). Comparaison de la partie Alpha au dictionnaire des éléments autorisés.
traitement_lexème(Lexème) :- En cas d'échec du prédicat
    écrire("Erreur, le symbole ***Lexème*** **n'est pas autorisé").**

Remarque : En réalité, les erreurs sont numérotées et stockées dans un fichier "ARCHER.err". Nous donnons, en Annexe, quelques exemples de messages rencontrés.

(*) En Turbo-Prolog, il existe une fonction pré-définie *frontstr* qui facilite ce genre de traitement.

(**) Le caractère '_' signale une variable non déterminée, c'est-à-dire qu'elle ne sera pas instanciée.

III.3.2. Analyseur syntaxique

C'est un programme qui vérifie la syntaxe d'une phrase à savoir la fonction et la disposition des mots (lexèmes) déduits de l'analyse lexicale. L'analyse est spécifiée par des règles de syntaxe, dont l'ensemble forme la grammaire du langage. La reconnaissance des lexèmes se fera d'une manière prédictive, parallèlement à l'analyse syntaxique.

En effet, le processus lexico-syntaxique se fera suivant le schéma de la figure 3.15. A partir d'un texte, nous allons effectuer le traitement ligne par ligne. Pour chaque ligne on commence par extraire un lexème qui sera analysé lexicalement et suivi d'une analyse syntaxique basée sur un automate à état fini. Le but de cette analyse est de vérifier si la position du lexème, par rapport aux autres, est correcte.

Lorsqu'une erreur est rencontrée, on s'arrête pour retourner à l'éditeur de texte, sinon on passe à la ligne suivante. L'analyse lexico-syntaxique se termine avec succès si la fin du texte est atteinte sans erreur.

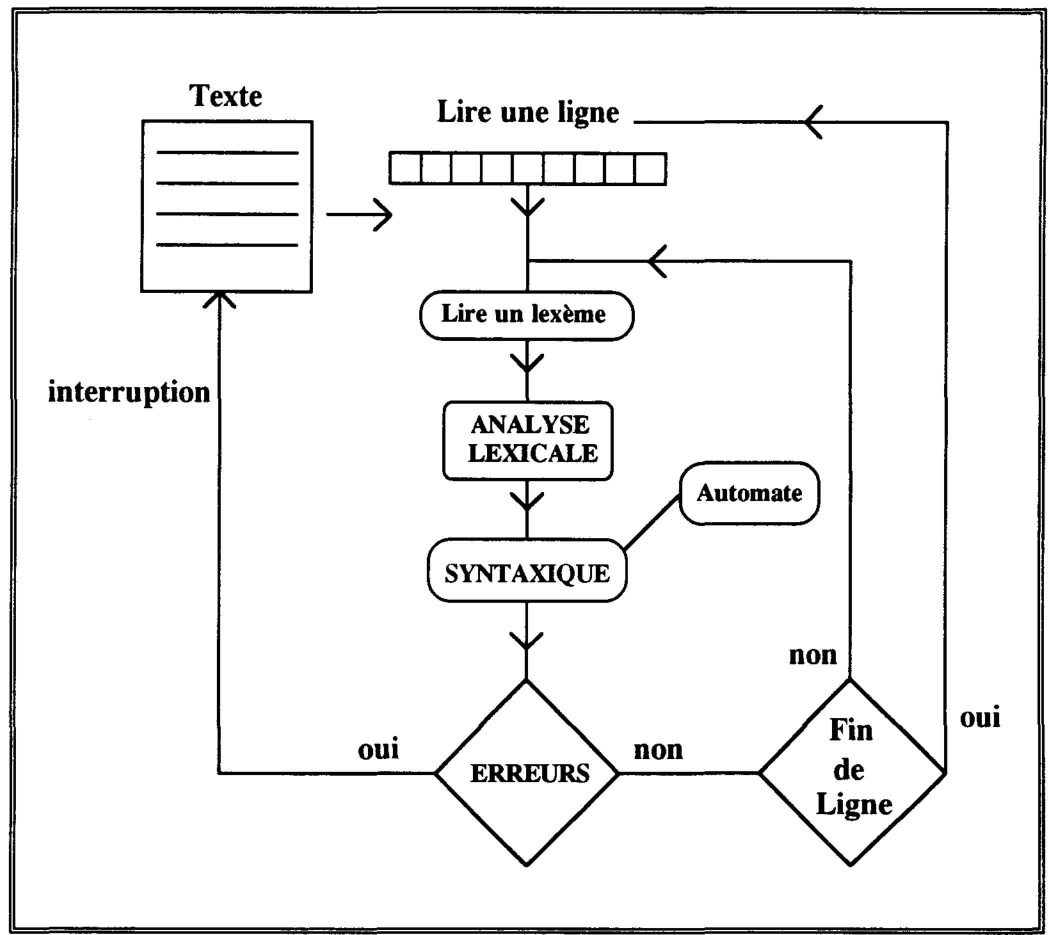


fig 3.15 : Schéma du processus lexico-syntaxique.

Le prédicat qui permet ce processus est une amélioration et une extension du prédicat ***lexical*** présenté plus haut.

lexico_syntaxique(' ').	Arrêt si la chaîne est vide
lexico_syntaxique(Ligne) :-	
extraire_lexème(Ligne, Lexème, Reste_de_la _ligne),	Extraire le lexème de la ligne
traiter_lexème(Lexème),	Appartenance du lexème au dictionaire
automate_lexème(Lexème),	Parcours de l'automate
lexico_syntaxique(Reste_de_la_ligne).	Appel récursif

On peut remarquer que l'algorithme précédent (***lexico-syntaxique***) se base sur une méthode d'**analyse ascendante.** Le principe est de partir d'une ligne ou d'une phrase et de la décomposer en entité élémentaire (Lexème) et de vérifier sa syntaxe selon les règles de production jusqu'à atteindre la racine de la grammaire choisie.

Par exemple la phrase **"01 R1 Se1"** de la partie jonction du langage bond-graph se décompose suivant l'arbre syntaxique de la fig 3.16.

On aurait pu choisir une méthode d'**analyse descendante** qui consiste à trouver la suite de règles qui mènent la racine à la phrase à analyser. Mais dans notre cas, nous considérons la première méthode plus naturelle car elle est facilement réalisable sous Turbo-Prolog.

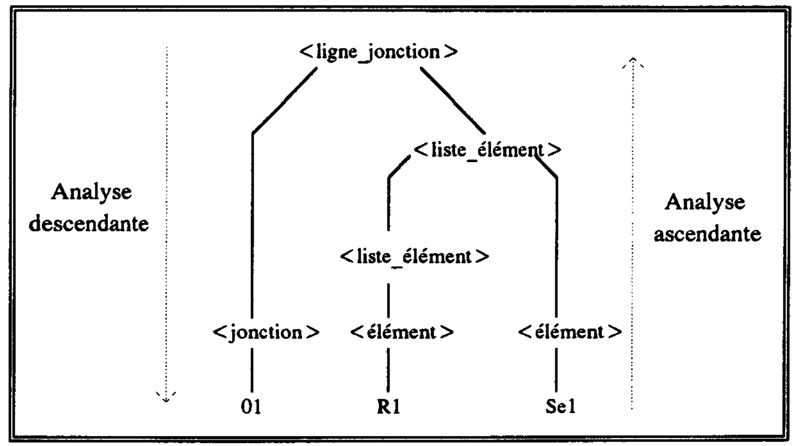


fig 3.16 : Arbre syntaxique de la partie "jonction".

Dans la famille des analyses ascendantes, il existe une méthode d'analyse qui permet l'étude syntaxique de nombreux langages et qui, par sa structure, est facilement implantable en Prolog. C'est la méthode de l'analyseur **LR(k)** (**L**eft to right input scans, **R**ighmost derivation in reverse), c'est-à-dire qu'il analyse une phrase de gauche à droite en s'appuyant sur ce qui a déjà été vu de la phrase et sur les **k** éléments sources suivant le dernier traité. Il est basé sur l'utilisation d'un automate.(*)

(*) L'automate LR(k) est aussi utilisé dans le traitement du langage naturel [J-H. JAYEZ 82]

III.3.2.1. Automate à états finis.

Un **automate** est un dispositif possédant un nombre fixe d'états, entre lesquels il peut évoluer, et une ou plusieurs entrées et sorties. La sortie d'un état ne se fait que si l'information reçue correspond à une des transitions (signaux) de sortie.

L'état initial est représenté par une flêche entrante et l'état terminal ou final par une flêche sortante.

Par exemple, l'automate de la figure 3.17 est construit autour de 5 états activés selon les transitions a, b, c, d, e, f et l'état courant. Les états 1 et 2 sont des états de départs, et les états 3 et 5 des états de sorties.

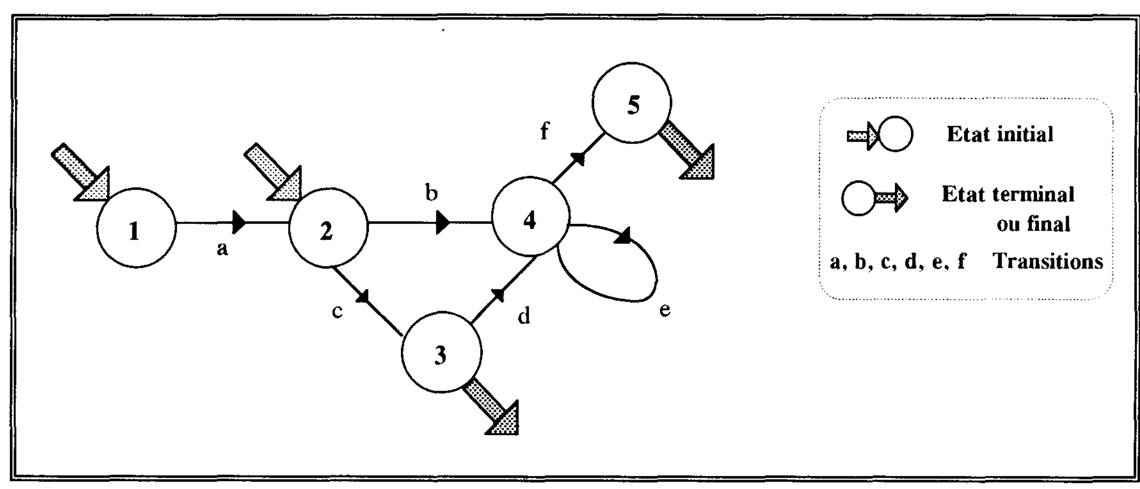


fig 3.17 : Représentation graphique d'un automate.

Pour établir le nombre, l'ordre et la structure des états composant l'automate, nous avons besoin de la grammaire du langage à analyser.

La liste des états est dérivée de la grammaire, de telle façon que chaque état soit un reflet de la structure de la partie déjà vue de la phrase à analyser. Chaque transition caractérise un métanom de la grammaire.

Dans le cas qui nous intéresse, un analyseur **LR(1)** suffit à décrire la plupart des grammaires rencontrées. L'entrée sera le texte source et le signal sera la lecture du lexème. Le nouvel état atteint dépend à la fois du **1er** lexème suivant et de l'état antérieur de l'automate.

Exemple : L'automate de la figure 3.18 représente la syntaxe de la partie jonction vue dans la figure 3.16 à la différence prés que nous avons inclu les séparateurs comme le montre la grammaire NBN suivante.

<ligne_jonction> ::= **<jonction> <séparateur> <liste_elément>**

<liste_élément> ::= **<liste_élément> <séparateur> <élément>** | ε

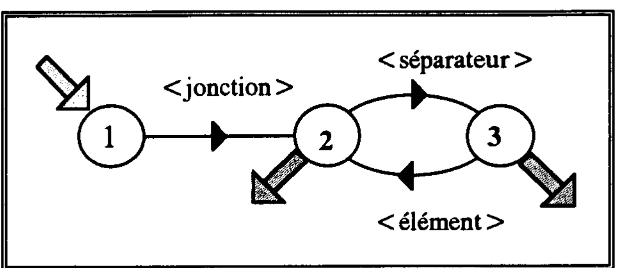


fig 3.18 : Automate de la partie jonction.

⇒Fonctionnement de l'automate

On utilise les diagrammes d'états pour reconnaître si une suite de lexèmes est syntaxiquement correcte.

- On commence à partir de l'état courant initial D et on explore la liste de lexèmes de gauche à droite.
- Pour chaque lexème lu, on part de l'état courant suivant l'arc de transition et étiqueté par ce lexème, ce qui mène à un nouvel état qui devient l'état courant. Une transition entre chaque état sera représentée en général par un métanom. Dans certains cas on accède à des transitions nulles qui ne conduisent à aucune lecture de lexème.
- Si la phrase est correcte, on doit pouvoir, à la fin, s'arrêter dans un état final.

Pour traiter un Automate, on distingue deux méthodes :

- La première consiste à parcourir les étapes d'un automate sans garder une trace sur les états déjà rencontrés. Dans ce cas, nous disons que l'automate n'a aucun souvenir.

- La seconde associe à l'automate une pile qui va mémoriser un certain nombre d'états et/ou de transitions qui précédent l'état courant. Elle est particulièrement utilisée lorsqu'on emploie des langages parenthèsés.

Prenons un langage basé sur les expressions mathématiques avec l'alphabet :

{ A , B , ... , Z , '+' , '/' , '(' , ')' }

où les signes '+' et '/' sont des opérations autorisées sur les lettres [A , ... , Z] et où les parenthèses '(' et ')' servent à regrouper des expressions.(*)

(*) Cet exemple n'est pas anodin, il nous servira à définir la grammaire du langage série et parallèle.

Il existe plusieurs grammaires pour décrire ce langage, en voici une dans laquelle nous allons utiliser le métanom **<expression$_i$>** qui caractérise une combinaison de lettre avec les signe '+' et '/' et les parenthèses et dont l'indice **i** sert à définir les différents niveaux à l'intérieur des expressions.

<expression1>	**:: = <expression2> | <expression1> '+' <expression2>**
<expression2>	**:: = <expression3> | <expression2> '/' <expression3>**
<expression3>	**:: = (<expression1>) | <lettre>**
<lettre>	**:: = A | B | ... | Z**
<signe>	**:: = + | /**

Il existe une méthode permettant de construire l'automate à pile d'une telle grammaire, elle est assez complexe et aboutit souvent à un automate possédant un nombre important d'états [NOYELLE 88].

Nous préférons construire un automate d'une façon plus intuitive à travers l'analyse heuristique de quelques exemples.

A + B / C   A / (C + D) + B   ((A + B) / (C + D) + E / F / G + H .....

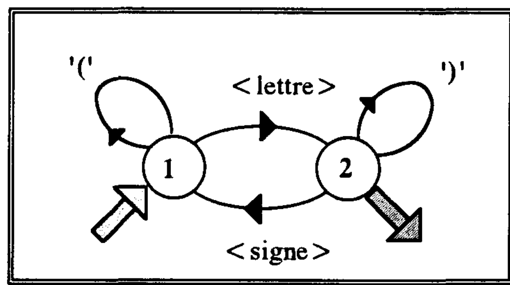


fig 3.19 : Automate basé sur les expressions mathématiques

Nous mémorisons les transitions sur les parenthèses ouvrantes (**Po**) et fermantes (**Pf**), en vérifiant, parallèlement à l'analyse de l'expression, les contraintes suivantes :

- La première parenthèse doit être une parenthèse ouvrante.

- A tout instant, nous devons avoir : le **nombre de *Po* > = nombre de *Pf***

Exemple :

"(--(----)--)--)---"   **nbre_*Po*** = 2 et **nbre_*Pf*** = 3   il manque une ***Po***
ou il y a une ***Pf*** en trop

- A la fin de l'expression nous devons obtenir **nombre de *Po* = nombre de *Pf*.**

⇒Problèmes de déterminisme

S'il n'existe aucune ambiguïté à la sortie de chaque état, c'est-à-dire que les signaux sont tous différents, alors dans ce cas l'automate est **déterministe** (figure 3.20) ; dans le cas contraire il est **non-déterministe** (figure 3.21).

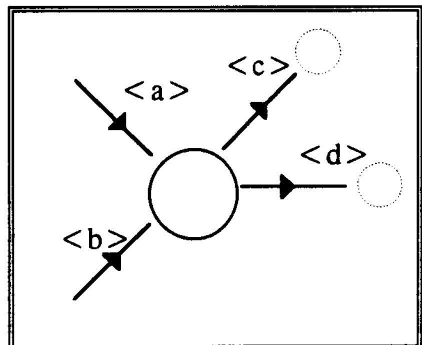


fig 3.20 : Automate déterministe.

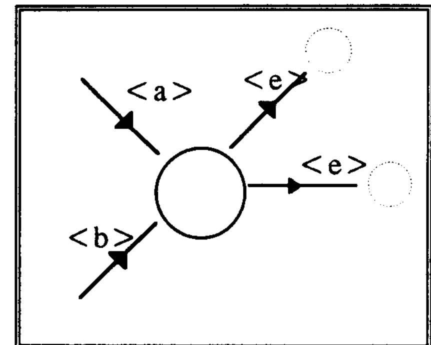


fig 3.21 : Automate non-déterministe.

Par exemple la grammaire du langage série-parallèle dérive de la grammaire des expressions mathématiques. Une description commence par un point en série avec le circuit à décrire et se termine par un point. (*)

<ligne>	**:: = <point> + <expression1> + <point>**
<expression1>	**:: = <expression2> | <expression1> '+' <expression2> | <point>**
<expression2>	**:: = <expression3> | <expression2> '/' <expression3>**
<expression3>	**:: = (<expression1>) | <éléments>**

L'automate d'une telle grammaire est représenté par la figures suivante :

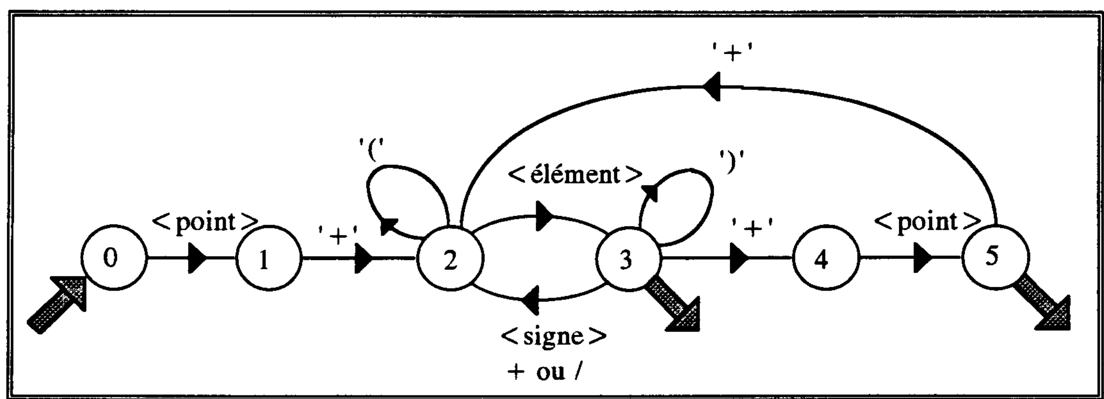


fig 3.22 Automate du langage série et parallèle.

L'état **3** est non déterministe car la rencontre d'un signe '+' peut conduire à l'état **2** ou **4**.

(*) Cela est surtout vrai pour le domaine électrique.

Pour ne pas alourdir l'automate, nous n'avons pas cherché à le rendre déterministe. Cependant cette situation est facilement solvable, il sufit pour cela de tester l'élément qui vient après le signe "+". Cette opération n'est pas coûteuse en temps puisqu'elle est **2-décidable** : pour sortir d'une situation non déterministe, on ne peut se décider qu'au bout de 2 lectures successives.

Dans le cas du langage série-parallèle basé sur les éléments physiques, il est préférable d'utiliser un automate du type LR(2).

III.3.2.2. Erreurs syntaxiques

Lorsqu'on rencontre un lexème qui ne correspond pas à la transition attendue et qui arrête la progression dans l'automate, on dit que c'est une erreur syntaxique.

Lorsqu'une erreur intervient, le compilateur a le choix entre deux approches : soit s'arrêter à la première erreur rencontrée ; soit poursuivre l'analyse syntaxique du reste du texte, de façon à détecter toutes les éventuelles autres erreurs. Ce deuxième cas peut s'avérer coûteux en temps et en mémoire.

Dans tous les cas, il doit s'efforcer d'indiquer exactement où l'erreur a été détectée, et le type de l'erreur (si possible par un message clair), ensuite il doit se re-synchroniser sur le texte source. Cela se fait à l'aide des compteurs de lignes et de colonnes, afin de repérer la place exacte de l'interruption.

L'analyseur **LR** permet de diagnostiquer ce genre d'erreur par l'adjonction à l'automate d'un **'état-puits'** qui permet de maîtriser toute interruption liée à un blocage de l'automate.

Par exemple la figure 3.23 représente l'automate de la figure 3.18 enrichie d'un état-puits qui sera caractérisé par un état grisé.

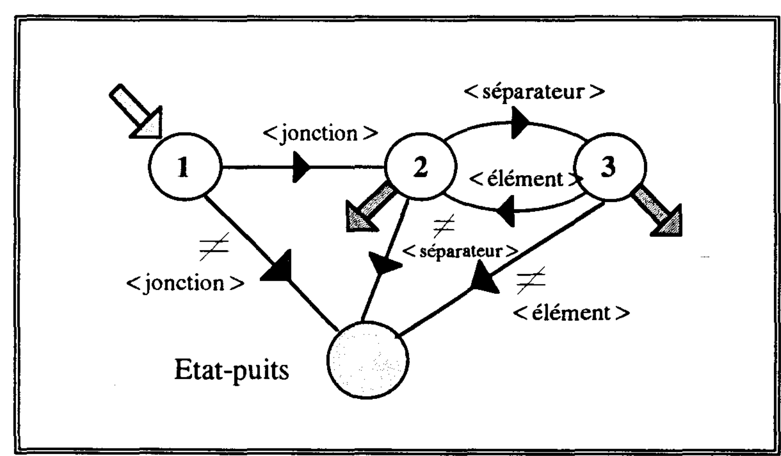


fig 3.23 : Automates de la partie jonction avec un état-puits.

Avec un tel analyseur, il est possible de corriger l'erreur, car l'automate "sait" ce qui aurait dû se trouver à l'endroit erroné. On peut ajouter ce qui devait se trouver, mais faut-il supprimer l'élément courant (remplacement) ou le conserver (ajout pur) ? Dans les deux cas on risque d'ajouter des erreurs parasites.

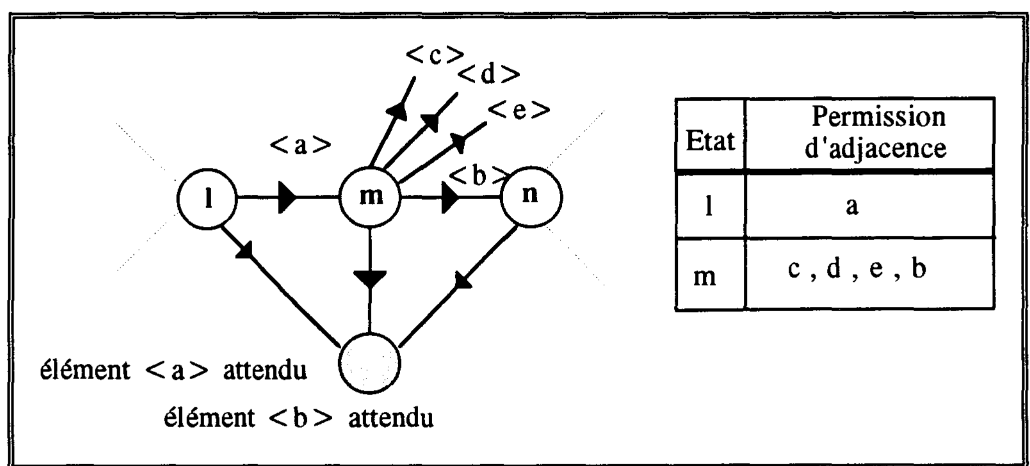
Exemple :

pour le langage bloc série parallèle, nous pouvons rencontrer ce type de phrase : **"... R1 + + C1 ..."**.

Il y a la répétition du signe '+' qui peut signifier, soit une erreur de frappe qui peut être facilement rattrapée en supprimant un plus, soit l'absence d'un élément ou d'un point : **"... R1 + C1 ..."** ou **"... R1 + ? + C1 ..."**

De même dans la phrase **"... L2 / + C2 ..."** on peut corriger l'erreur par l'élimination d'un des deux signes ce qui a pour conséquence de changer le comportement de **L2** par rapport à **C2**. Dans un cas ils sont en parallèle et dans l'autre en série. On peut aussi signaler, comme plus haut, l'absence d'un élément :
"... L2 / ? + C2 ...".

Dans le cas général, on introduit la notion de **permission d'adjacence**, qui consiste à associer à chaque état de l'automate du langage la liste des métanoms autorisés juste après lui comme le montre la figure 3.24. Ainsi, chaque fois que l'on prend l'élément suivant du texte source, on regarde s'il figure dans le tableau des adjacences autorisées ; dans le cas contraire, on détecte une erreur.



Etat	Permission d'adjacence
l	a
m	c , d , e , b

fig 3.24 : Permission d'adjacence dans un automate.

Si la détection des erreurs syntaxiques est un problème maîtrisé, il n'en est pas de même pour une correction entièrement automatique et pour un diagnostic précis qui permet à l'utilisateur de cerner l'erreur. De plus on a souvent le choix entre plusieurs diagnostics qui peuvent aboutir à plusieurs messages et compliquer la correction de l'utilisateur.

Nous préférons adopter la méthode qui consiste à corriger au-fur-et-à-mesure les erreurs dans le texte en suivant l'automate.

En cas d'échec, on tombe dans un état-puits et le message de l'erreur signale le métanom attendu à la place du lexème courant.

III.3.2.3. Automate en prolog

Comme nous venons de le voir précédemment, le traitement syntaxique du lexème, se fait à travers un automate **déterministe** à états finis qui peut être programmé de plusieurs façons en Turbo-Prolog. La méthode que nous proposons consiste à décomposer la structure du programme en deux parties.

- La première partie sert à décrire l'automate par des faits qui peuvent être regroupés dans une data-base et sauvegardés dans un fichier.

- La deuxième partie a en charge le fonctionnement de l'automate proprement dit, c'est-à-dire qu'il établit si une phrase est licite par rapport à une grammaire exprimée sous forme de transitions. Si le passage d'un état initial à un état final réussit, la ligne anlysée fait partie du langage reconnu par la grammaire.

Cette structure est générique car elle permet de réutiliser le même traitement syntaxique sur un autre langage en chargeant dans la data-base courante l'automate associé à la grammaire.

En **Annexe 2**, nous définissons les informations nécessaires à la description d'un automate.

III.3.3. Analyse sémantique et génération de code

L'analyse sémantique consiste à vérifier si le texte analysé a une signification. En effet, les phrases analysées peuvent être syntaxiquement correctes, mais prises dans leur globalité, elles peuvent montrer un non sens qui risque de générer un mauvais code. C'est à ce niveau que seront traitées les erreurs dites de "sémantique statique", non repérables par une grammaire et donc non détectables lors de l'analyse syntaxique.

Faut-il effectuer cette analyse avant ou après la génération de code ?

Par exemple, pour les langages de programmation, la sémantique vient avant la génération de code. Souvent, ces 2 phases sont confondues. Cependant, lors de l'exécution du code, il peut apparaître des erreurs non détectables avant sa génération.

En ce qui nous concerne, nous distinguons trois phases pour l'analyse sémantique :

Phase 1 *Analyse pré-sémantique*

Il s'agit essentiellement des erreurs de descriptions ou des erreurs de frappes qui dépendent du langage de description utilisée.

Phase 2 *Génération de code*

Il s'agit de transformer la description de l'utilisateur en un code bond-graph acausal tel que nous l'avons établi. (en suivant la sémantique interne du langage)

Le but d'un compilateur est de générer un langage plus simple, en l'occurrence le code bond-graph. Pour obtenir ce code, on se base sur la grammaire du langage en associant à chaque règle syntaxique, une description correspondante [E. IRONS 61].

On appelle ces descriptions des actions sémantiques car elles représentent en quelque sorte la sémantique de la traduction en indiquant ce qui doit être généré à chaque réduction des règles de production.

Phase 3 *Analyse post-sémantique*

Elle se fait essentiellement en quatre étapes :

<u>Etape 1</u> Elle consiste à identifier les erreurs topologiques liées à la connexité du graphe (bond-graph).

<u>Etape 2</u> Elle simplifie le code généré selon les règles de la méthodologie bond-graph.

<u>Etape 3</u> Elle analyse la propagation de la puissance à travers les jonctions du bond-graph.

<u>Etape 4</u> Elle effectue une mini expertise sur les éléments dynamiques attachés aux fonctions afin d'anticiper d'éventuels conflits causaux lors de l'affectation de la causalité.

Remarque : Pour les deux dernières étapes, il est préférable de travailler sur un modèle bond-graph simplifié.

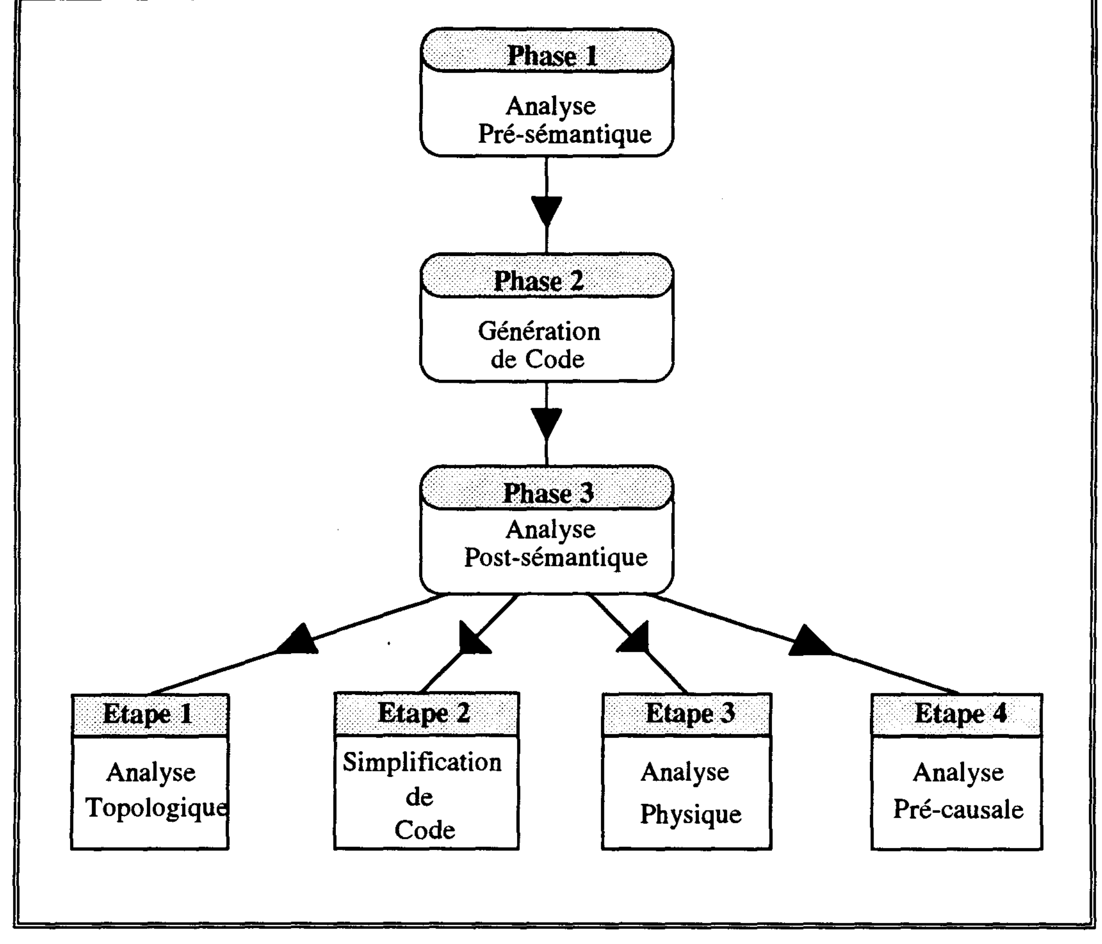


fig 3.25 : Schéma des différentes phases de l'analyseur

Comme les deux premières phases dépendent plus du langage utilisé, nous allons les retrouver dans la partie qui traite des langages de description au chapitre IV.

La troisième phase est plus générale et ne s'appuie que sur le code généré. Nous avons choisi de la présenter avant les deux premières.

III.3.4. Analyse post-sémantique

Les étapes 1, 3 et 4 peuvent conduire à des anomalies que l'on peut considérer soit comme des erreurs, soit comme des situations choisies volontairement par l'utilisateur.

A chaque fois, une anomalie est signalée mais l'utilisateur reste toujours maître de la situation. Il peut suivre les solutions proposées ou passer outre et construire à sa guise le bond-graph voulu. Ce principe sera suivi dans les autres modules.

III.3.4.1. Erreurs topologiques

Nous pouvons vérifier la connexité du bond-graph grâce au fait lien qui relie les jonctions entre elles, celles-ci étant connectées automatiquement aux éléments qui l'entourent.

Définition : Rappelons qu'un graphe est dit **connexe** si :
Pour tout couple de sommets i et j (i < > ou = j), il existe une chaîne joignant i et j. [MINOUX & Al 79]

Pour cela, nous allons utiliser l'algorithme proposé au paragraphe III.2.1.3 en remplaçant le fait chemin par le fait ***chaîne(X,Y)*** qui cherche l'existence de toutes les arêtes liées entre 2 sommets **X** et **Y**.

```
chaîne (X,Y) :-  arête(X,Y).
chaîne(X,Y) :-   arête(X,Z),
                 chaîne(Z,Y).
chaîne(X,Y) :-   write("Bond-graph non connexe car pas de chaîne entre les jonctions X et Y")
```

Une ***arête(X,Y)*** est soit un ***lien(X,Y)*** ou un ***lien(Y,X)***. Si la chaîne n'existe pas alors le graphe n'est pas connexe.

```
arêtes(X,Y) :-   lien(X,Y).
arêtes(X,Y) :-   lien(Y,X).
```

Il nous faut fabriquer tous les couples de sommets possibles à partir de la liste de toutes les jonctions présentes dans le bond-graph à travers le fait ***présent***.(*)

On appelle le fait ***présent(L)*** et on exécute le prédicat ***sommet(L)*** qui va lire les jonctions de la liste L et générer les couples en question.

```
sommet([ X | Q ]) :-
        faire_couple(X,Q),
        sommet(Q).

faire_couple(X,[X1 | Q]) :-
        assert(couple(X,X1)),
        faire_couple(Q).
faire_couple(X,[]) :- assert(couple(X,X)).
```

(*) Ne pas confondre le fait ***existant*** et le fait ***présent***.

Les faits ***couples*** sont stockés dans une base de données dynamique. Le fait ***connexe*** appelle successivement (à l'aide du fail) tous les ***couples*** et vérifie l'existence d'une chaîne entre les 2 jonctions.

```
connexe :-
        couple(X,Y),
        chaîne(X,Y),
        fail,!.
connexe :- écrire("Le bond-graph est connexe").
```

III.3.4.2. Simplification

Avant d'obtenir le code bond-graph final, il est nécessaire d'opérer certaines opérations de simplification sur le code 'bond-graph intermédiaire' qui va être généré par les langages séries/parallèles et blocs/noeuds que nous allons présenter plus loin.

Pour le langage bond-graph, le code doit refléter exactement la structure voulue par l'utilisateur, la simplification ne s'impose pas toujours.

Le principe de la simplification est facilité par la forme des données bond-graphs (prédicats ***jonction*** et ***lien***) et l'inférence naturelle de Prolog.

Il suffit de décrire les règles de simplification citées au chapitre II. Prolog va se charger de trouver toutes les configurations jonctions et liens possibles en relation avec les règles, leur appliquer les transformations nécessaires, créer de nouveaux faits et effacer les anciens. Ainsi de proche en proche, on atteint l'état où toutes les règles sont résolues.

Cela se fait à l'aide du prédicat ***simplification*** qui réussit lorsque tous les prédicats ***règle(i)*** sont vérifiés.

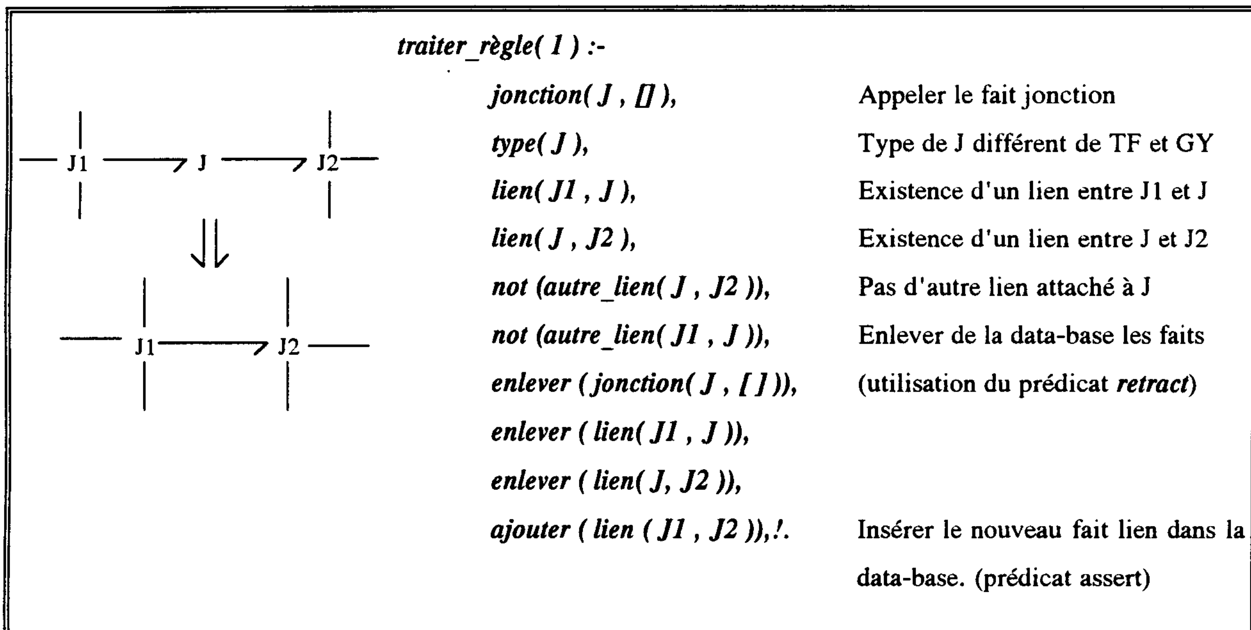
```
simplification :-
        règle(1),
        règle(2),
        règle(3),
        règle(4),
        écrire("Simplification du bond-graph terminée"),!.
```

Pour toutes les règles nous avons la structure suivante :

règle(i) :-	Faire le traitement correspondant à la règle i.
traiter_règle(i),	Recommencer si le traitement a réussi.
règle(i),!.	En cas d'échec on passe à la règle i +1 suivante, la
règle(i).	Règle i étant mise en attente.

Pour chaque règle, nous allons expliciter la procédure qui, à travers le prédicat ***traiter_règle(i)***, permet de simplifier le bond-graph.

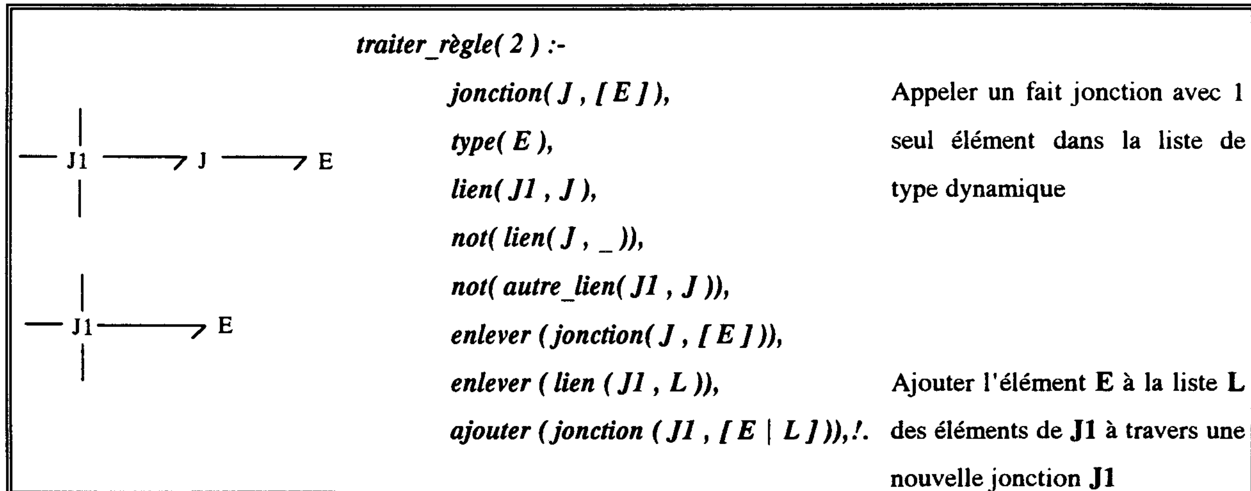
REGLE 1



traiter_règle(1) :-	
jonction(J , []),	Appeler le fait jonction
type(J),	Type de J différent de TF et GY
lien(J1 , J),	Existence d'un lien entre J1 et J
lien(J , J2),	Existence d'un lien entre J et J2
not (autre_lien(J , J2)),	Pas d'autre lien attaché à J
not (autre_lien(J1 , J)),	Enlever de la data-base les faits
enlever (jonction(J , [ ])),	(utilisation du prédicat *retract*)
enlever (lien(J1 , J)),	
enlever (lien(J, J2)),	
ajouter (lien (J1 , J2)),!.	Insérer le nouveau fait lien dans la data-base. (prédicat assert)

Pour la règle 3, le traitement fonctionne sur le même principe que la règle 2, à la différence que les liens entre **J1**, **J** et **E** sont inversés et que **E** est un élément source.

REGLE 2 & 3

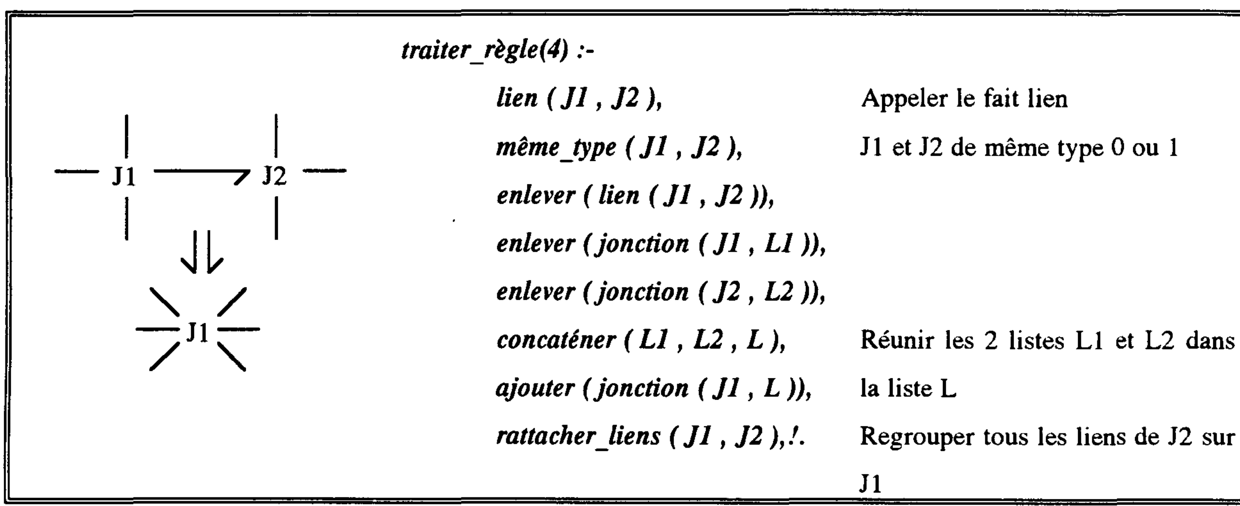


traiter_règle(2) :-	
jonction(J , [E]),	Appeler un fait jonction avec 1 seul élément dans la liste de type dynamique
type(E),	
lien(J1 , J),	
not(lien(J , _)),	
not(autre_lien(J1 , J)),	
enlever (jonction(J , [E])),	
enlever (lien (J1 , L)),	Ajouter l'élément **E** à la liste **L**
ajouter (jonction (J1 , [E | L])),!.	des éléments de **J1** à travers une nouvelle jonction **J1**

le prédicat ***autre_lien*** est commun aux règles 1, 2 et 3. Il sert à vérifier si la jonction **J** est attachée à une autre jonction.

```
autre_lien(J1,J) :-                          autre_lien(J,J2) :-
        lien(J3,J),                                  lien(J,J4),
        J3 < > J1,                                   J4 < > J2,!.
```

REGLE 4




```
traiter_règle(4) :-
        lien ( J1 , J2 ),                       Appeler le fait lien
        même_type ( J1 , J2 ),                  J1 et J2 de même type 0 ou 1
        enlever ( lien ( J1 , J2 )),
        enlever ( jonction ( J1 , L1 )),
        enlever ( jonction ( J2 , L2 )),
        concaténer ( L1 , L2 , L ),             Réunir les 2 listes L1 et L2 dans
        ajouter ( jonction ( J1 , L )),         la liste L
        rattacher_liens ( J1 , J2 ),!.          Regrouper tous les liens de J2 sur
                                                J1
```

Le prédicat ***rattacher_liens (J1,J2)*** sert à lier tous les liens attachés à la jonction **J2** vers la jonction **J1**, puisque **J2** a été simplifiée.

```
rattacher_liens (J1,J2) :-
        retract (lien (J,J2)),          regrouper les liens sortants de la jonction J2
        J< >J1,
        ajouter (lien (J,J1)),
        ratttacher_liens(J1,J2)),!.
rattacher_liens (J1,J2) :-
        retract (lien (J2,J)),          regrouper les liens entrants de la jonction J2
        J< >J1,
        ajouter (lien (J1,J)),
        ratttacher_liens(J1,J2)),!.
```

Au final, nous obtenons un bond-graph acausal simplifié qui se traduit au niveau du graphe par une minimisation du nombre d'arcs et de noeuds. Le code peut-être manipulé plus facilement par les autres modules d'ARCHER et en particulier par le module d'affectation de la causalité dont le traitement est fonction de la taille du graphe.

III.3.4.3. Validation du sens physique

Il peut arriver que dans le code bond-graph, généré après la simplification, certaines jonctions (**0** ou **1**) aient tous leurs liens :

- entrant avec une liste d'éléments constitués de sources (Se, Sf) ou vide.

- ou sortant avec une liste d'éléments dynamiques (R, C, I) ou vide comme le montre la figure 3.26.

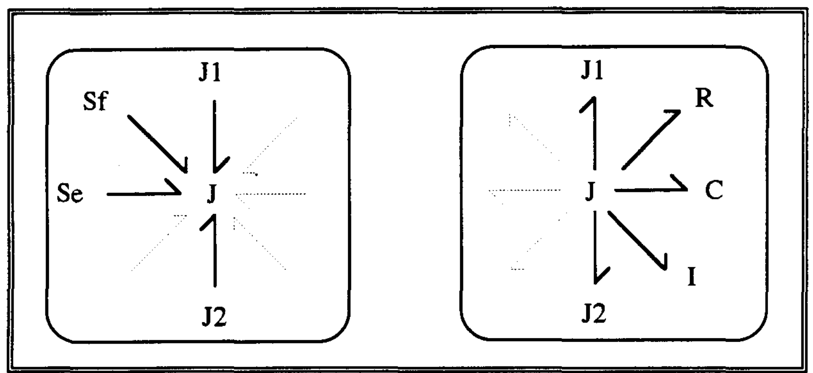


fig 3.26 : Situation extrême sur un jonction J de type 0 ou 1.

En effet, comme nous l'avons suggéré au chapitre II, nous pouvons signaler le cas où le sens des liens dans le bond-graph bloquent le parcours de la puissance.

Cette situation peut apparaître par exemple en électronique de puissance où une mauvaise description du circuit triphasé fait apparaître une jonction **0** là où il faudrait un élément R 3-ports.

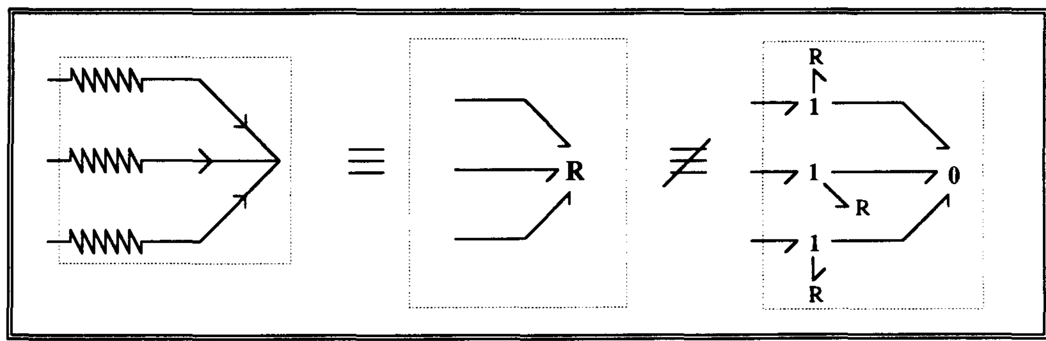


fig 3.27 : Partie bond-graph d'un schéma électrique triphasé.

Cette situation peut donc être synonyme d'une erreur qui met en cause le sens physique du modèle étudié.

On essaye d'avoir pour tout le bond-graph, une configuration qui facilite le parcours de la puissance à travers les jonctions en ayant à chaque fois les cas suivants :

- au moins un lien entrant et au moins un lien sortant sur une jonction J adjacente à plus de 2 jonctions (figure 3.28a),

- au moins une source (Se ou Sf) sur une jonction J avec des liens sortants et des éléments dynamiques (figure 3.28b),

- au moins un élément dynamique sur une jonction J avec des liens entrants et des sources (figure 3.28c),

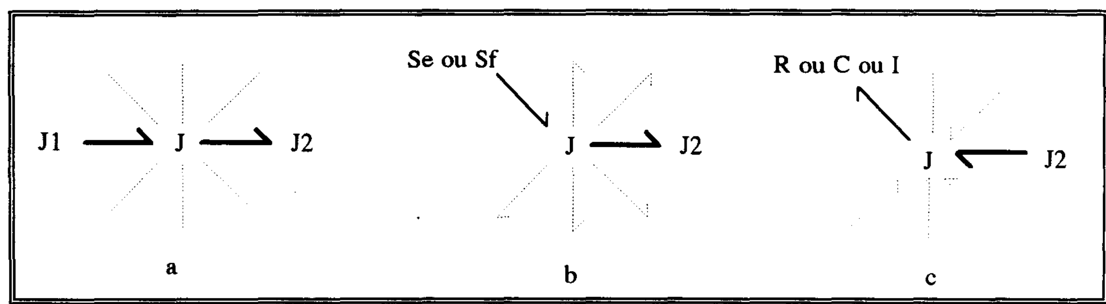


fig 3.28 : Situation privilégiée pour une jonction J.

Pour cela nous allons nous définir trois états de la jonction à l'aide d'un prédicat ***état(J,E)*** dans lequel **E** est une variable qui prend trois formes :

E=i : Toutes les demi-flêches autour de la jonction **J** sont entrantes.

E=o : Toutes les demi-flêches autour de la jonction **J** sont sortantes.

E=io : Il y a au moins une demi-flêche entrante et une flêche sortante de la jonction J comme le montre la figure ci-dessus.

L'état qui permet une totale circulation de la puissance, d'une jonction à une autre, est l'état '**io**', car nous avons une relation avec une puissance entrante (**Pi**) et sortante (**Po**) : Σ **Pi** = Σ **Po**.

Les autres états ont tendance à figer cette relation : Σ **Pi** = Σ **Po** = **0**.

La procédure peut se dérouler parallèlement à la génération de code. La méthode est la suivante :

- Au début, **E** est initialisé à vide '**v**'.

- Pour chaque jonction **J** créée, on génère le fait ***état(J,E)*** selon les règles présentés dans le tableau de la figure 3.29.

Cas de figures	Appeler le fait *jonction(J , Liste_éléments)*	Opérations
	Etat ' i '	
Source j J J1 j	L = Liste_éléments non vide L ne contient que des sources Se ou Sf J a tous ses liens entrants donc pas de liens sortants J est dans un état i	***not(lien (J, _))*** ***état (J , i)***
j j J1 J j j	L = Liste vide J a tous ses liens entrants J est dans un état i	***not(lien (J, _))*** ***état (J , i)***
	Etat ' o '	
Elément Dynamique j J J1 j	L ne contient que des Eléments dynamiques R, C, I J a tous ses liens sortants J est dans un état o	***not (lien (_ , J))*** ***état (J , o)***
j j J J1 j j	J a tous ses liens sortants J est dans un état o	***not (lien (_ , J))*** ***état (J , o)***
	Etat ' io '	
Elément Dynamique j J J1 Source j	L contient au moins une source et au moins un élément dynamique J a des liens entrants et / ou sortants J est dans un état io	***état (J , io)***
J1 J J2	J possède au moins un lien entrant et au moins un lien sortant J est dans un état io	***lien (_ , J)*** ***lien (J , _)*** ***état (J , io)***

fig 3.29 : Règles de génération des états i et o.

La ou les jonctions qui possèdant les états **'i'** (***état(J,i)***) ou **'o'** (***état(J,o)***) sont signalées à l'utilisateur

Remarque : A la fin de l'expertise nous pouvons trouver des jonctions **TF** et **GY** dans un état **i** ou **o** comme le montre la figure 3.30.

Cela traduit une erreur dans la description des transducteurs qui doit être signalée à l'utilisateur. Les jonctions **TF** et **GY** doivent être obligatoirement dans un état **io.**

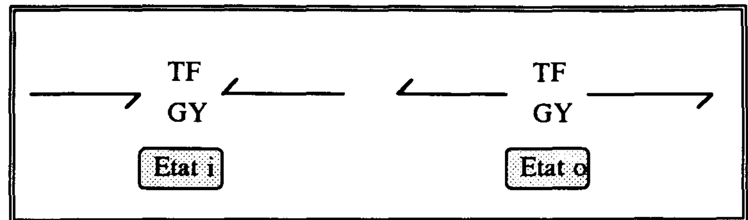


fig 3.30 : Etat non toléré pour un transducteur.

Exemple :

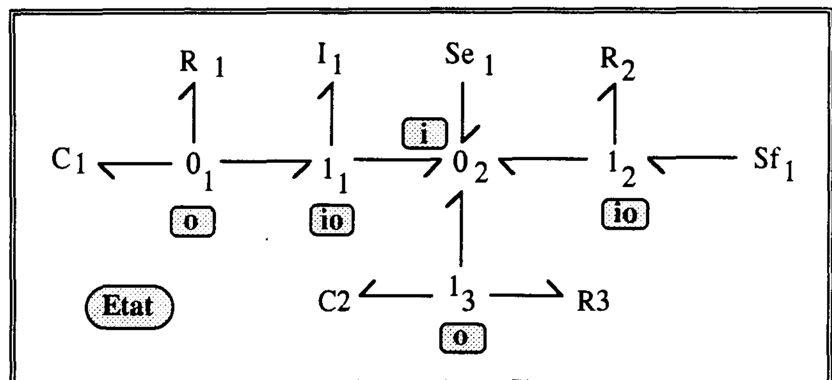


fig 3.31 : bond-graph avec des états i, o, io.

Les jonctions **01** et **12** sont respectivement dans l'état **'o'** et **'i'**. L'utilisateur a plusieurs solutions pour les forcer à **'io'**, avec la contrainte qui consiste à n'enlever aucun élément. Pour débloquer une situation **'i'** ou **'o'**, nous proposons les solutions suivantes :

Proposition 1

Inverser le sens du lien entre deux jonctions de manière à obtenir deux jonctions dans un état **'io'**. Les jonctions dans un état **'o'** adjacentes à des jonctions dans un état **'i'** seront traitées en priorité. Par exemple pour résoudre l'état **'i'** de la jonction **02** nous inversons le lien qui joint **02** à une jonction dans un état **'o'** en l'occurrence **13**.

Proposition 2

Ajouter une source (**Se** ou **Sf**) lorsque la jonction est dans un état **'o'**.

Par exemple on ajoute une source d'efforts sur la jonction **01**.

Proposition 3

Ajouter un élément dynamique lorsque la jonction est dans un état **'i'**.

Par exemple une résistance **R** sur la jonction **02**.

Proposition 4

Ne rien faire et tolérer les états **'i'** et **'o'**.

Remarque : Au final, pour toute jonction J, nous ne devons trouver aucun fait ***état(J,E)*** avec **E=v**.

En effet cette variable vide signifierait que la jonction n'est attachée à aucun élément, ce qui est possible, ou à aucune jonction, ce qui est impossible, car pour définir la 'partie lien' il faut obligatoirement un minimum de deux jonctions par ligne (voir la grammaire du langage). De plus nous avons, au préalable, vérifié dans l'étape 1 la connexité du bond-graph.

Ces remarques, pour le moins intuitives, sont dans un premier temps, le résultat empirique d'exemples pris dans la littérature bond-graph. Elles nous ont amené à chercher une justification théorique qui trouve sa résolution dans le concept qui met en relation le bond-graph et la théorie des matroïdes [BIRKETT 89].

III.3.4.4. Analyse pré-causale

Bien qu'ARCHER possède un module d'affectation de la causalité qui permet de traiter les conflits causaux, nous pouvons prévoir certains de ces conflits dès la génération du code bond-graph acausal par une analyse sémantique un peu plus poussée sur un bond-graph acausal.

En effet il est facile de construire, parallèlement à l'analyse syntaxique, des informations capables de vérifier certaines règles élémentaires, dans le but de détecter ces conflits. Il suffit de dénombrer le nombre d'éléments R, C, I, Se et Sf respectivement sur chaque jonction **0** et **1** lors de l'analyse syntaxique et de les stocker dans le fait :

nombre_élément (champ1 , champ2 , champ3)

Le **Champ1** représente le type de l'élément **E** (R, C , I, Se, Sf);
Le **Champ2** est la jonction **J** (**0** ou **1**) courante.
Le **Champ3** est le nombre d'élément de type **E** présent sur la jonction **J**.

Comme dans les analyses précédentes, le principe est d'interpeler l'utilisateur sur une éventuelle erreur.

Conflits sur les sources

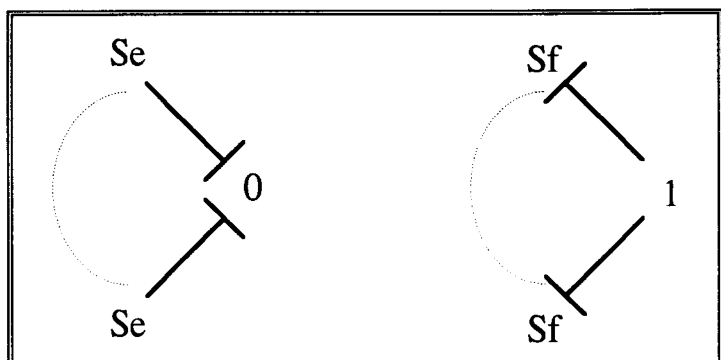


fig 3.32 : Conflits possibles avec un élément source.

Règle 1 :

S'il y a plus d'une source d'efforts sur une jonction **0**.

nombre_élément (Se , 0 , N) et N > 1

Règle 2 :

Si il y a plus d'une source de flux sur une jonction **1**.

nombre_élément (Sf , 1 , N) et N > 1

Conflits sur les éléments dynamiques

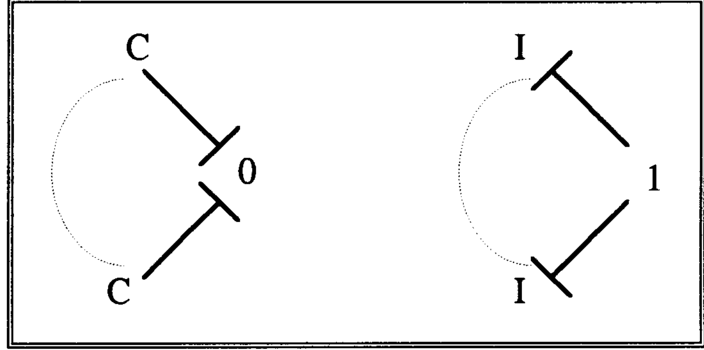


fig 3.33 : Conflits avec des éléments capacitifs et inductifs.

Règle 3 :

S'il y a plus d'un élément de type capacité sur une jonction **0**.

nombre_élément (C , 0 , N) et N > 1

Règle 4 :

S'il y a plus d'un élément de type inductance sur une jonction**1**.

nombre_élément (I , 1 , N) et N > 1

Boucle de causalité

Définition : Une boucle de causalité est une succession de jonctions et de liens formant un cycle, pour lesquels la causalité est orientée dans le même sens sur tous les liens, (sauf en présence d'un GY).

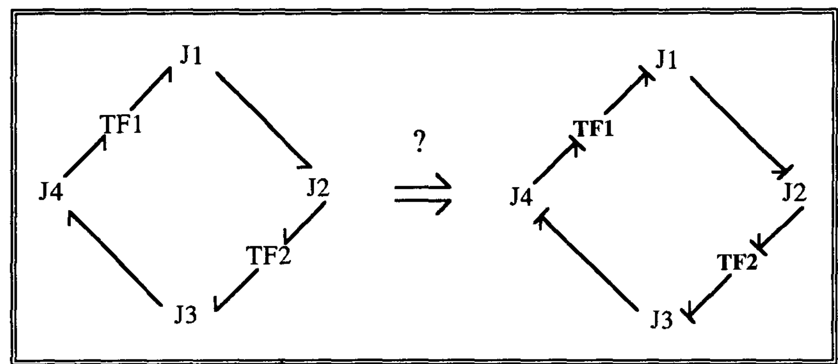


fig 3.34 : Boucle de causalité.

On repère les boucles par une recherche de chemin classique entre une jonction et elle-même. Si une boucle ne contient aucun transducteur **TF** et **GY,** il y a le risque d'avoir, lors de l'affectation de la causalité, une boucle de causalité de gain 1.

En effet nous pouvons éviter la boucle de causalité grâce à un choix judicieux des gains des transducteurs, de telle façon que leur produit soit différent de 1.
[ROSENBERG 79] [WLASOWSKI 91]

III.3.4.5. Tableau de conversion

Pour les métanoms **<éléments>**, une database de conversion est créée au fur et à mesure de la lecture des lexèmes. En effet, comme nous l'avons montré dans le tableau des correspondances de la fig 2.65, on peut associer un élément physique et un symbole universel à un élément bond-graph.

Ces informations sont synthétisées par le fait ***conv(D,S_UNI,S_BG)*** où **D** représente le domaine physique, **S_UNI** le symbole universel et **S_BG** son équivalent bond-graph.

Lors de l'analyse syntaxique, on construit au fur et à mesure les faits ***conv*** lors de l'identification des lexèmes par le fait ***identifier***.

identifier(Elmt,Métanom):-	**Elmt** est du type **Métanom** ?
séparer(Elmt,A,NE),	Séparer la partie Alphanumérique **A** de l'indice **NE**.
existe(A,Métanom),	Echec si **A** n'est pas de type **Métanom**
équivalence(A,S_UNI, S_BG,D),	Chercher les symboles équivalents de **A**
concat(S_UNI,NE,NEW_S_UNI),	Concaténer le **S_UNI** avec l'indice
concat(S_BG,NE,NEW_S_BG),	Concaténer le **S_BG** avec l'indice
assert(conv(D,NEW_S_PHY,NEW_S_BG)).	Stocker le fait ***conv***

On construit le code bond-graph du modèle décrit à l'aide des symboles bond-graphs (**S_BG**) avec l'indice adéquat.

Lors de l'affichage du bond-graph et des équations mathématiques, on utilise le symbole universel (**S_UNI**).

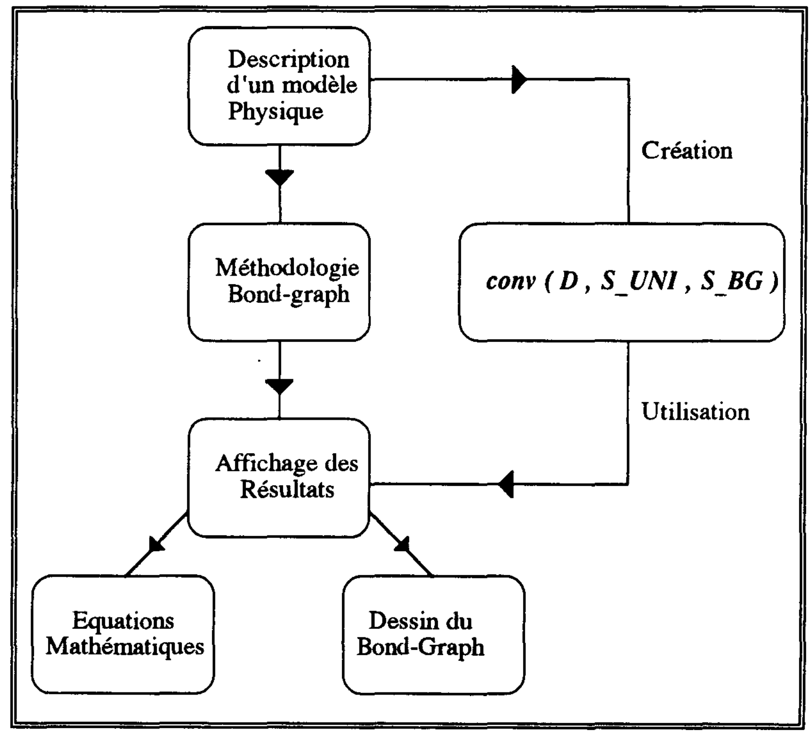


fig 3.35 : Création et utilisation du fait *conv*.

Il y a ainsi uniformité des données dans les différents traitements effectués par ARCHER : description, génération de code et affichage.

Pour le langage bond-graph, nous travaillons évidemment avec des éléments bond-graphs. Les faits ***conv*** seront de la forme : ***conv(bg,R1,R1), conv(bg,Se1,Se1)***...

Exemple :

Nous allons décrire un système mécanique à l'aide du langage utilisateur afin de montrer l'importance et l'utilité du fait ***conv*** au niveau de l'affichage du bond-graph et des équations en formelle.

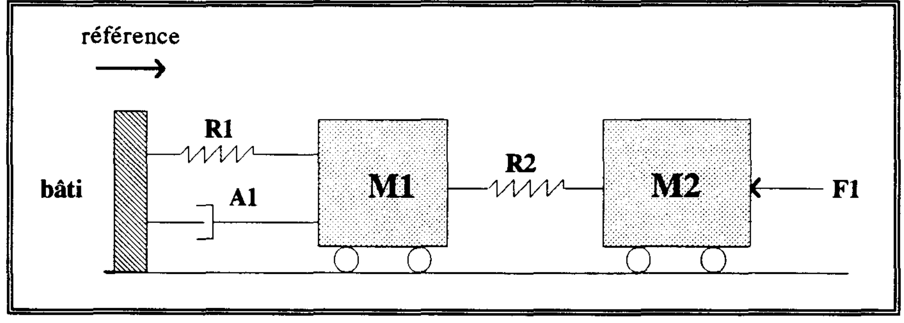


fig 3.36 : Modèle mécanique.

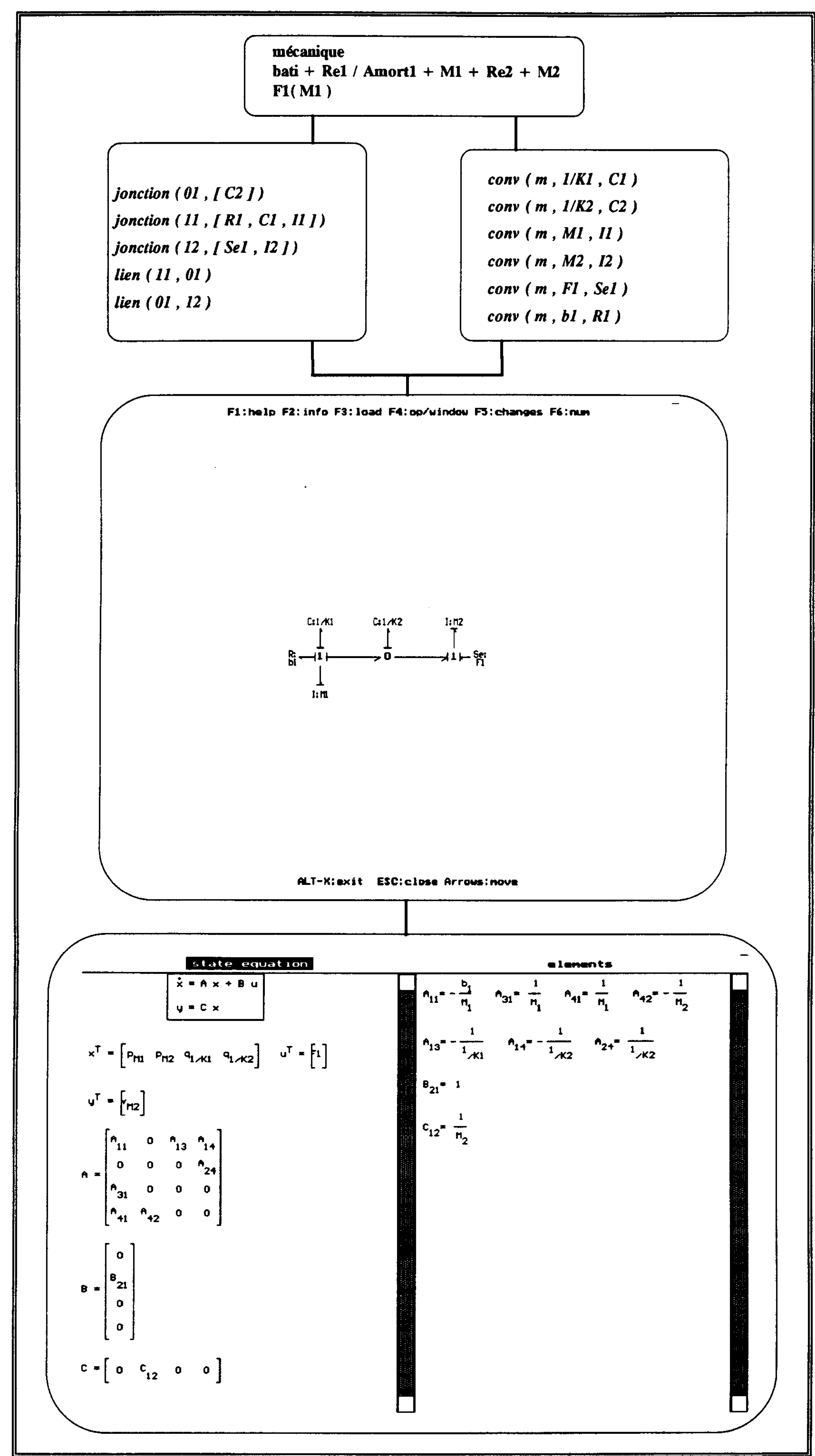


fig 3.37 : Copie d'écran du modèle mécanique de la figure 3.38.

III.4. Conclusion

Dans ce chapitre, nous avons présenté les caractéristiques et les éléments de base qui composent le compilateur d'ARCHER. Nous avons particulièrement insisté sur les parties communes aux différents langages que nous étudierons plus en détail dans le prochain chapitre. Dans les parties communes, nous avons présenté :

- L'analyseur lexical dont l'algorithme de fonctionnement reste valable quelque soit le langage utilisé.

- L'analyseur syntaxique à travers le traitement d'un automate à états finis, associé à une procédure de détection des erreurs à l'aide des états-puits.

- L'analyseur post-sémantique qui se base sur une forme de code bond-graph commune à tous les langages. Cette partie joue un rôle préventif car elle permet à l'utilisateur d'identifier des erreurs liées à la topologie, à la description et au sens physique du modèle en cours de construction.

Dans cette classe de sémantique nous proposons une analyse pré-causale qui permet de déceler, avant l'affectation de la causalité proprement dite, d'éventuels conflits causaux.

Les deux autres phases de l'analyseur sémantique (pré-sémantique et générateur de code), seront traitées, pour chaque langage, dans le prochain chapitre, après avoir spécifié, la syntaxe et la grammaire à travers les différentes transitions associées à chacun des automates mis en jeu dans les langages.

Nous aurions pu nous contenter des 2 phases de l'analyseur sémantique, mais nous avons cherché, à travers la phase post-sémantique, une analyse qualitative du code généré par ARCHER. En effet, l'analyse du parcours de la puissance à travers le sens des liens pour chaque jonction est une méthode originale qui permet, à partir du code bond-graph acausal, d'anticiper certaines erreurs de description du système étudié avant de passer aux autres modules d'ARCHER.

CHAPITRE IV

LANGAGES DE DESCRIPTION ET GENERATION DE CODE BOND-GRAPH

LANGAGES DE DESCRIPTION ET GENERATION DE CODE BOND-GRAPH

IV.1. Introduction

Nous avons souligné dans le chapitre précédent les aspects génériques du compilateur d'ARCHER, à savoir le traitement syntaxique des langages, les analyses post-sémantiques sur le code bond-graph supposé déjà généré, la simplification de celui-ci ; nous nous intéressons à présent à sa génération à travers les différents langages que nous avons proposés au chapitre II.

En premier nous présentons le langage bond-graph réservé, plus particulièrement, au bond-graphiste, qui offre à l'utilisateur la possibilité de décrire, sous une forme texte, le bond-graph issu de 'sa modélisation' d'un modèle physique. Cela sous-entend qu'il utilise tous les avantages de cette méthodologie, c'est-à-dire la possibilité d'englober, dans un même bond-graph, plusieurs domaines de la physique.

Lorsque le modèle est formé d'éléments appartenant au même domaine physique, nous avons vu qu'il est possible de décrire, sous certaines conditions, ce modèle à l'aide du langage série & parallèle. Afin de faciliter la compréhension et la subtilité de la méthode qui conduit à la génération du bond-graph, nous utiliserons un exemple pris dans le domaine électrique.

A partir de ces deux langages, nous pouvons définir des blocs en ajoutant une partie déclarative dans laquelle nous définissons les ports d'entrées **in** et de sorties **out** qui servent à coupler les blocs entre eux. Ces blocs sont sauvegardés sous la forme d'un code bond-graph.

Pour coupler ces blocs, nous présentons le langage blocs et noeuds qui construit une structure intermédiaire dans laquelle va se greffer, à l'aide des ports, les bond-graphs des blocs utilisés dans le modèle étudié.

Nous verrons qu'il est possible d'étendre ces trois formes de description et les faire coexister dans un langage plus général qui permet à son tour de définir un bloc bond-graph formé de plusieurs blocs.

Enfin nous aborderons le couplage lorsque la modélisation de certains blocs nécessitent une représentation sous la forme d'une fonction de transfert ou d'une équation d'état. Nous proposerons différentes méthodes pour les intégrer à ARCHER, notamment pour la phase d'analyse structurelle qui nécessite une représentation graphique de la globalité du système étudié. Cette "unification" est permise par la théorie des digraphes qui permet de représenter une matrice d'état par une combinaison de sommets et d'arcs orientés et pondérés.

IV.2. Description homogène d'un système physique

IV.2.1. Langage bond-graph

Rappelons que ce langage va permettre de décrire rapidement, à partir d'un texte, une représentation graphique d'un bond-graph.

Commençons à présenter la grammaire associée à ce langage construite autour de l'alphabet {a .. z, A .. Z, 0 .. 9, :} et dont les métanoms sont représentés par la forme de backus suivante :

<mot_clef> ::= **'jonction' | 'lien'**
<jonction> ::= **<type_jonction> <nombre_entier>**
<élément> ::= **<type_élément> <nombre_entier>**
<nombre_entier> ::= **<chiffre> | <nombre entier> <chiffre>**
<type_jonction> ::= **0 | 1 | TF | GY | MTF | MGY | R | C | I**
<type_élément> :: = **R | C | I | Se | Sf |**
<chiffre> ::= **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |**

Comme nous l'avons signalé aux chapitres II et III, la description se fait en deux parties déclaratives. Pour la bonne compréhension du langage proposé, nous allons scinder la grammaire en deux sous-grammaires.

Une sous-grammaire **G1** pour la partie commençant par le mot clef "jonction" qui concerne le traitement des éléments associés aux jonctions.

Une sous-grammaire **G2** pour la partie commençant par "lien" qui sert à décrire l'ossature du bond-graph (uniquement les liens entre les jonctions) en suivant le sens des demi-flêches dans les liens.

Une grammaire se définit par un alphabet A, un vocabulaire non-terminal Vn, des règles de production P, une racine R : G(A,Vn,P,R). Nous donnons la définition des 2 sous-grammaires G1 et G2 du langage bond-graph.

G1(A,{<ligne_jonction>,<liste-élément>,<jonction>,<élément> ... },P1,<ligne_jonction>).
G2(A,{<ligne_lien>,<liste_jonction>,<jonction> ... },P2,<ligne_lien>).

Règles de production P1

<ligne_jonction> ::= <jonction> <liste_elément>
<liste_élément> ::= <liste_élément> <élément> | ε

Règles de production P2

<ligne_lien> ::= <jonction> <liste_jonction>
<liste_jonction> ::= <liste_jonction> <jonction> |ε

IV.2.1.1. Analyseur syntaxique

Le principe de fonctionnement de l'analyseur syntaxique va se baser sur l'automate de la figure 4.1, chaque partie s'identifie par un mot clef 'jonction' ou 'lien'.

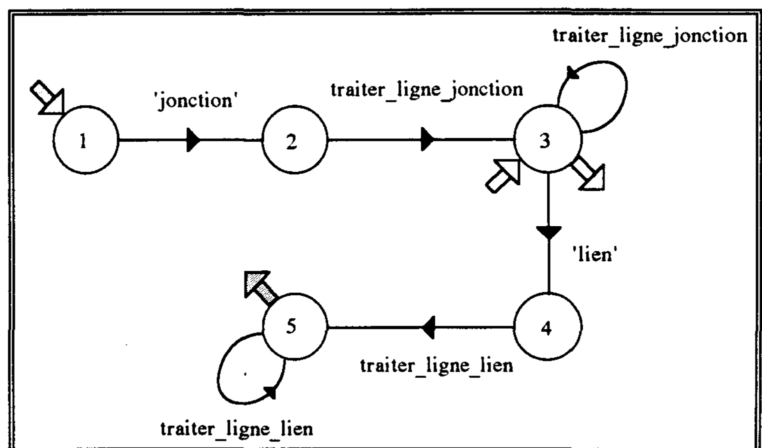


fig 4.1 : Automate de fonctionnement des deux parties déclaratives.

Remarque : Cet automate s'apparente à la structure d'un programme qui active les automates des parties 'jonction' et 'lien' à travers les transitions **traiter_ligne_jonction** et **traiter_ligne_lien.**

Selon la méthodologie bond-graph, la première partie est obligatoire et la deuxième peut ne pas exister. Cela se traduit sur l'automate par l'état 3 qui peut-être final.

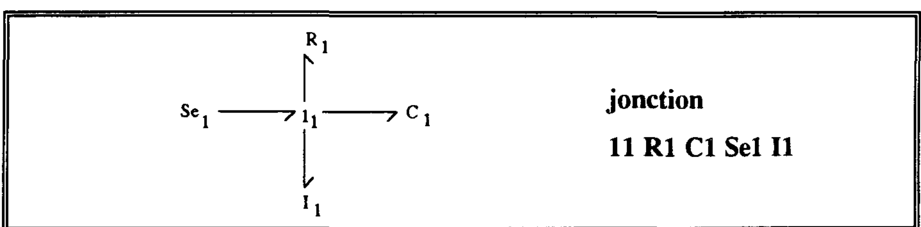


fig 4.2 : Exemple d'un bond-graph sans liens entre les jonctions

Cependant, il est possible de construire un bond-graph uniquement autour des liens sans éléments dynamiques ou sources,(*) dans ce cas l'état 3 devient initial.

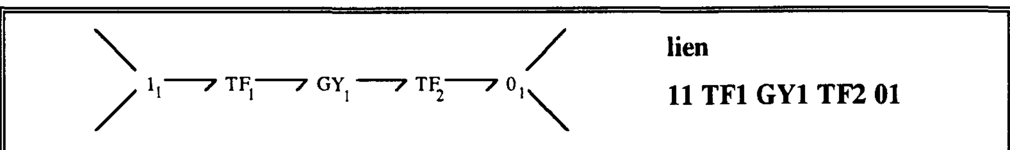


fig 4.3 : bond-graph sans éléments dynamiques ou sources.

(*) Voir le langage bloc bond-graph IV.3.

Ainsi la première partie commence par une ligne 'jonction' et les lignes suivantes se composent d'un lexème de type <jonction> suivi d'une liste de lexèmes identifiés à <élément> :

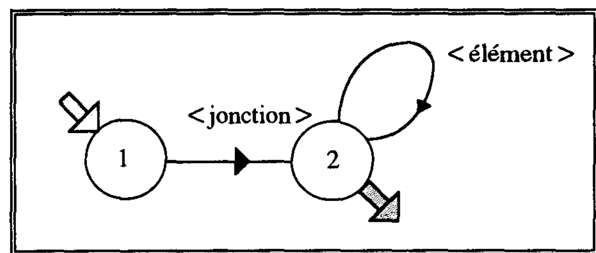


fig 4.4 : Automate pour le traitement d'une ligne jonction.

Nous pouvons avoir des lignes qui contiennent uniquement une jonction, cela indique que cette dernière ne possède pas d'éléments. C'est un cas de figure toléré par ARCHER, mais à éviter car il inclut une redondance inutile dans la description. En effet les jonctions décrites dans la partie 'lien' et que l'on ne rencontre pas dans la première partie, sont considérées comme ne possédant pas d'éléments.

Si la ligne suivante commence par le mot 'lien', nous sommes dans la partie **lien** et la syntaxe se compose d'une liste d'au moins deux lexèmes identifiés à <jonction>. En effet, pour qu'un lien existe il faut un minimum de 2 jonctions adjacentes.

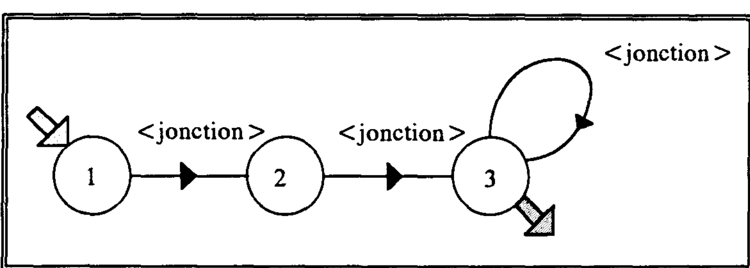


fig 4.5 : Automate pour le traitement d'une ligne lien.

S'il n'y a plus de ligne à analyser, le texte est alors correct. Dans le cas contraire, le compilateur s'arrête avec le message approprié.

La figure 4.6 représente la description texte sous ARCHER d'un modèle bond-graph.

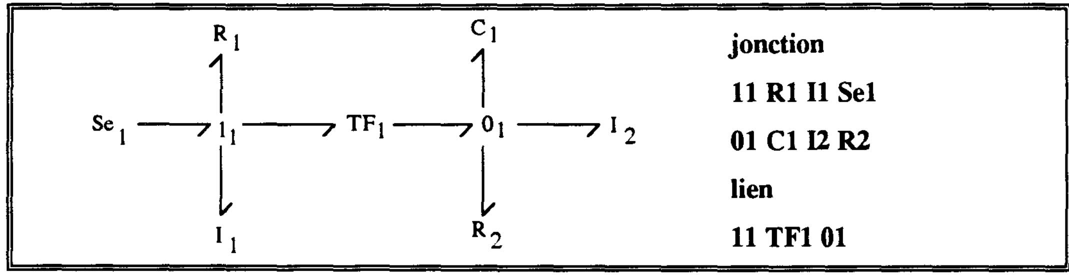


fig 4.6 : Description texte d'un bond-graph.

Cette description ne comporte aucune anomalie au regard des automates précédents.

Au chapitre III nous avons vu le fonctionnement général de l'automate syntaxique sous Prolog. Nous avons tout simplement signalé qu'il faut adapter le fait ***transition*** à l'automate étudié. Ainsi le traitement des deux automates précédents va se faire respectivement selon les transitions suivantes :

Automate de la partie jonction	Automate de la partie lien
transition (1 , jonction , 2)	*transition (1 , jonction , 2)*
transition (2 , élément , 2)	*transition (2 , jonction , 3)*
initial (1)	*transition (3 , jonction , 3)*
final (2)	*initial (1)*
	final (3)

fig 4.7 : Transitions des automates de la partie jonction et lien.

Les erreurs de syntaxes sont repérées par les états puits qui diagnostiquent l'erreur commise. Le fait d'avoir adopté une notation déclarative pour le langage permet de donner une orientation sémantique à l'analyse syntaxique. En effet, tirées du contexte déclaratif, les lignes, prises une à une, peuvent avoir une syntaxe correcte par rapport aux règles de production décrites plus haut, tout en étant placées au mauvais endroit.

IV.2.1.2. Analyse pré-sémantique

L'objectif de cette partie est de repérer des anomalies de description qui ne respectent pas la définition d'un bond-graph. Le traitement de cette partie se fait parallèlement à l'analyseur syntaxique, autrement dit, pendant la vérification syntaxique de la description du modèle étudié, nous enregistrons des informations qui nous permettront de déceler les anomalies suivantes.

⇒ Une jonction (J2) reliée uniquement à une autre jonction (J1) et qui possède une liste vide. Dans ce cas la jonction J2 peut-être éliminée.

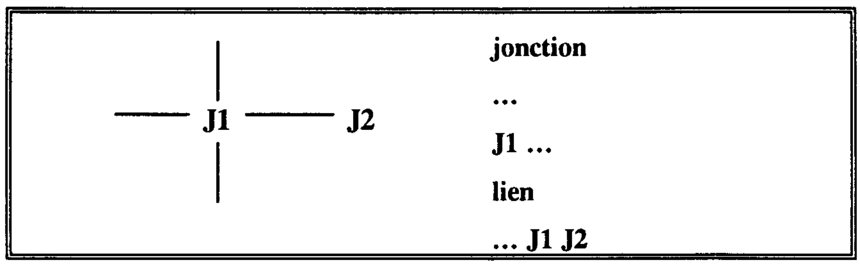


fig 4.8 : Jonction J2 reliée à une seule jonction.

⇒ Une jonction reliée à elle-même.

fig 4.9 : Répétition d'une même jonction.

⇒ Les boucles entre 2 jonctions telles que J1 soit reliée à une jonction J2 elle-même reliée à J1 comme le montre la figure suivante.

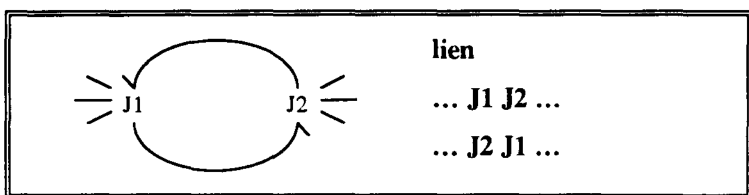


fig 4.10 : Jonctions reliées par deux liens en sens inverse.

Remarque : Ce genre de boucle peut générer des conflits inutiles lors de l'affectation de la causalité.

⇒ Le respect des éléments 2-ports TF, GY, MTF et MGY. Ces jonctions doivent être reliées 2 fois. Le premier port est relié à une jonction et le deuxième à une seule jonction ou un élément.

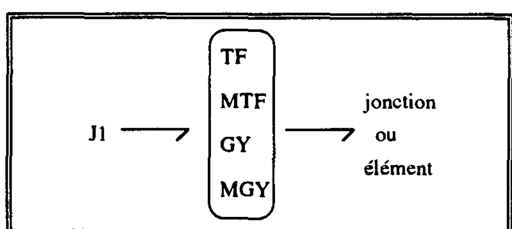


fig 4.11 : Liaison des éléments 2-ports.

⇒ Une jonction qui possède des éléments et qui n'apparaît pas dans la partie 'lien' comme nous pouvons le voir dans la figure suivante.

Les jonctions présentes dans la partie 'lien' sans avoir été définies dans la partie jonction ne sont pas forcément des erreurs. Par exemple la jonction **TF1** de la figure 4.12 n'a pas été définie car aucun élément ne lui est attaché, néanmoins nous aurions pu la déclarer dans la partie jonction par un **TF1** suivi d'une liste vide, dans ce cas l'information est redondante et la description n'est plus minimale. Cette anomalie peut se rapprocher de la connexité du graphe.

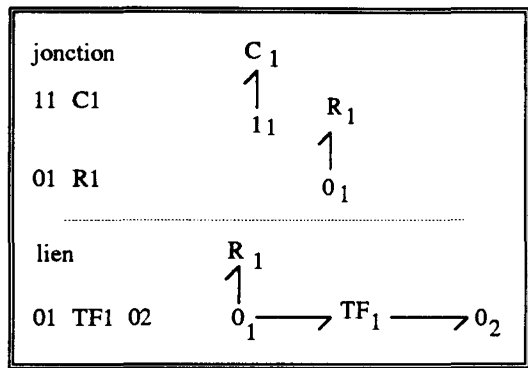


fig 4.12 : Erreur de description.

Pour détecter les erreurs analogues à la figure 4.12, nous allons créer, parallèlement à la lecture de la partie 'jonction', une liste des jonctions présentes à travers une Liste de Jonction **LJ** sauvegardée dans le fait ***existant(LJ)***. Ainsi, nous pourrons vérifier si toutes les jonctions de la liste **LJ** se retrouvent dans les jonctions lues dans la partie 'lien'.

IV.2.1.3. Génération de code

Il s'agit de générer un code sur le modèle d'ARCHER décrit dans le chapitre III. Les faits ***jonction(J,Liste_élément)*** et ***lien(J1,J2)*** caractérisent le parcours de la puissance entre 2 jonctions.

Cette séparation, souhaitée par le codage, nous a incité à créer un langage bond-graph adapté en respectant 2 parties déclaratives. Ce langage va être facilement transcodé par ARCHER.

Pour cette raison, cette génération de code est facile à mettre en oeuvre. Pour chaque ligne jonction on crée le fait ***jonction(J,L)*** avec comme paramètre **J** le premier Lexème de la ligne et le reste comme les éléments de la liste **L** représentant le 2° paramètre de ce fait. Ce processus se répète jusqu'à la fin du fichier texte ou la rencontre du mot clef 'lien'. Dans ce dernier cas on traite chaque ligne lien, composée d'une chaîne de jonctions, de la manière suivante :

- On crée le fait ***lien***(J_1,J_2) avec le premier Lexème et le deuxième,
- le fait ***lien***(J_2,J_3) avec le deuxième et le troisième Lexème ...
- le fait ***lien***(J_{n-1},J_n) avec les lexème n-1 et n ...
- On répète cette action jusqu'à la fin du fichier.

Nous supposons que la ligne à traiter est correcte, c'est-à-dire que les analyses lexicales et syntaxiques ont réussi.

PARTIE JONCTION

traiter_ligne_jonction:-	
lire(Ligne),	Lire une ligne
créer_jonction(Ligne),!.	Appeler le prédicat *créer_jonction*
créer_jonction(Ligne):-	
extraire_lexème(Ligne,Lexème,Reste_ligne),	Extraire le premier lexème qui est une jonction
créer_liste(Reste_ligne,Liste_éléments),	Mettre le reste des lexèmes dans une liste
ajouter(jonction(Lexème,Liste_éléments))!.	Ajouter le fait *jonction* dans la data-base
créer_liste(' ', []).	Renvoyer une liste vide si aucun autre lexème
créer_liste(Reste,[Elmt | Liste_éléments]):-	
extraire_lexème(Reste,Elmt,New_reste),	Manière récursive de créer une liste en respectant
créer_liste(New_reste,Liste_éléments),!.	l'ordre des lexèmes rencontrés

PARTIE LIEN

traiter_ligne_lien:-	
Lire(Ligne),	Lire une ligne
extraire_lexème(Ligne,Lex1,Reste_ligne),	Lire le premier lexème
not(egal(Reste_ligne,' ')),	Vérifier si le reste de la ligne est non vide
créer_lien(Lex1,Reste_ligne),!.	Appeler le prédicat *créer_lien*
créer_lien(_,' ').	
créer_lien(Lex1,Reste_ligne):-	
extraire_lexème(Reste_ligne,Lex2,New_reste),	Extraire un autre lexème
ajouter(lien(Lex1,Lex2)),	Ajouter le fait *lien*
créer_lien(Lex2,New_reste),!.	Appel récursif

IV.2.1.4. Exemple d'application

Considérons le bond-graph décrit dans la figure 4.13 du chapitre, celui-ci est un exemple assez complet qui rend compte des possibilités du langage proposé. Quelle que soit la description texte du bond-graph, nous obtenons le même code bond-graph et a fortiori les mêmes coordonnées donc le même dessin.

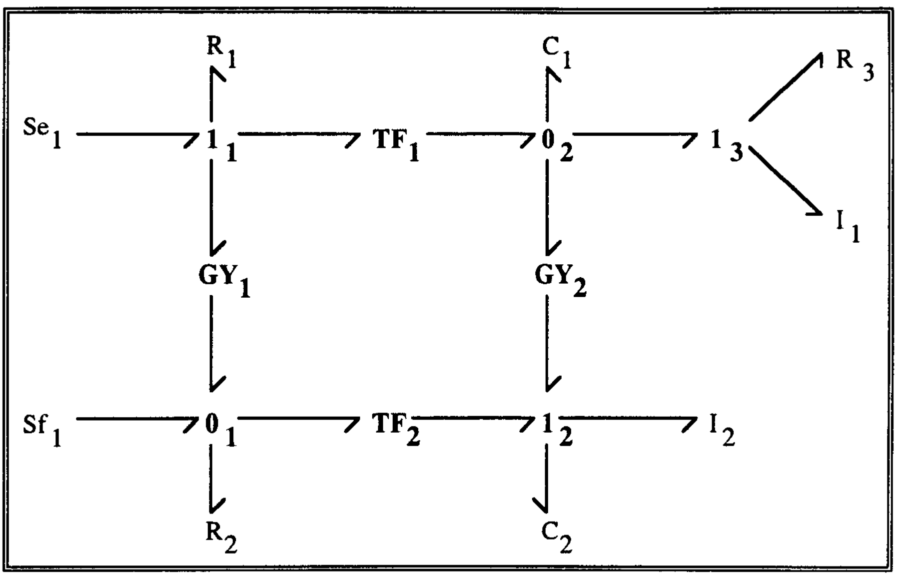


fig 4.13 : bond-graph à décrire

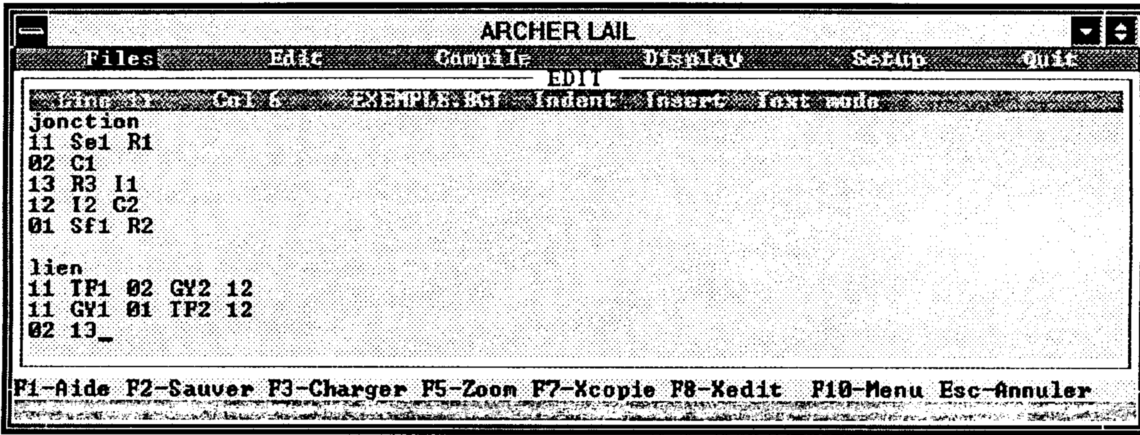


fig 4.14 : Description texte sous l'éditeur d'ARCHER.

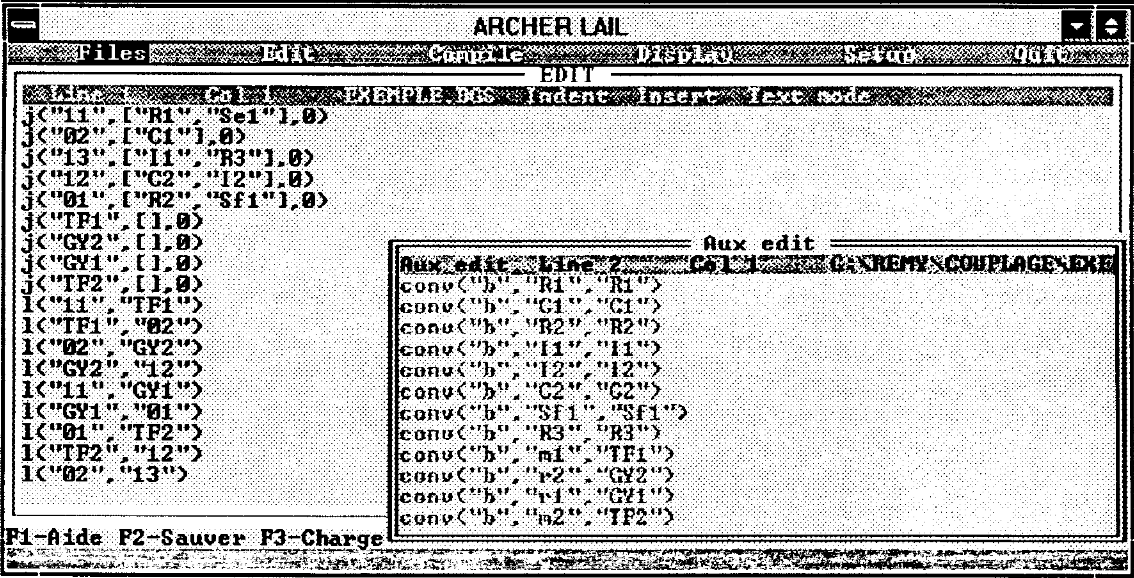


fig 4.15: Code bond-graph et le fichier de conversion de l'exemple de la figure 4.13.

Le module de dessin automatique d'ARCHER fournit le bond-graph suivant :

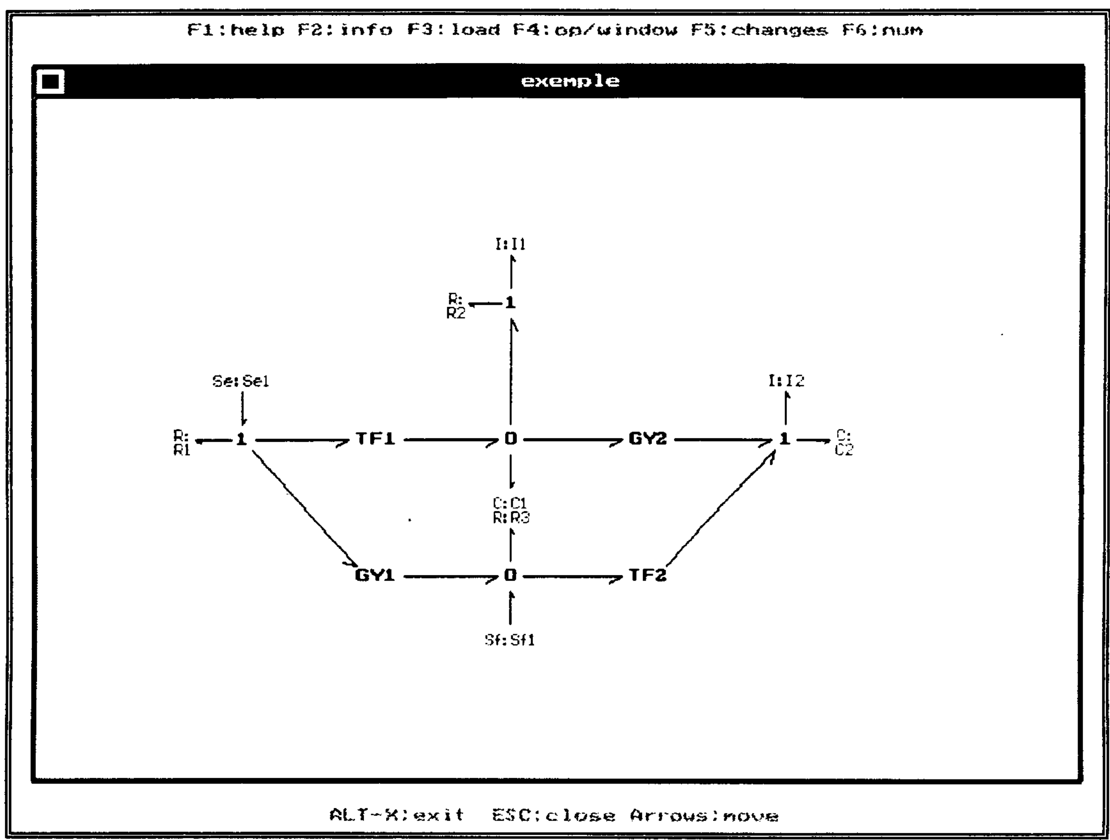


fig 4.16 : Affichage de la description du bond-graph de la figure 4.14.

IV.2.2. Langage Série & Parallèle

Chaque ligne de ce langage représente une expression dans laquelle on retrouve une combinaison des informations précédentes.

Cette expression, comme nous l'avons indiqué dans le chapitre III, ressemble à une expression arithmétique formelle avec les opérateurs '+' pour les éléments en séries et '/' pour les éléments en parallèles. L'opérateur '/' est prioritaire par rapport à l'opérateur série '+'. Nous avons attribué à ce langage, comme dans le cas des expressions arithmétiques, des parenthèses afin de respecter l'architecture et la structure du système.

Globalement, ce langage va utiliser l'alphabet suivant :

{a .. z , A .. Z , 0 .. 9 , +, / , (,)}

Les règles de productions associées à cette grammaire sont les suivantes :

<domaine> ::= 'électrique' | 'électronique' | 'mécanique' | ...

<ligne> ::= <point> <signe+> <expression1> | <expression1>

<expression1> ::= <expression2> | <expression1> <signe+> <expression2> | <expression1> <signe+> <point>

<expression2> ::= <expression3> | <expression2> <signe/> <expression3>

<expression3> ::= <parenthèse_ouv> <expression1> <parenthèse_ferm> | <élément>

<élément> ::= <type_élément> <nombre_entier>

<type_élément> ::= <type_électrique> | <type_mécanique> | <type_électronique> | ...

<type_électrique> ::= R | C | L | TFp | TFs | E | I | ...

<type_mécanique> ::= Am | R | M | F | V

<signe+> ::= '+'

<point> ::= <lettre_minuscule> <nombre_entier> | 'bâti'

<signe/> ::= '/'

<parenthése_ouv> ::= '('

<lettre_minuscule> ::= a | b | ... | z

<parenthèse_ferm> ::= ')'

<nombre_entier> ::= <chiffre> | <nombreentier> <chiffre>

<chiffre> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

IV.2.2.1. Analyseur syntaxique

La description commence par la déclaration du domaine physique **D** dans lequel on travaille, cette information est enregistrée dans le fait ***domaine(D)*** qui permet, lors de la génération de code, de choisir le traitement approprié. Pour chaque ligne, on active traitement à l'aide de la transition **traiter_ligne** comme le montre l'automate suivant :

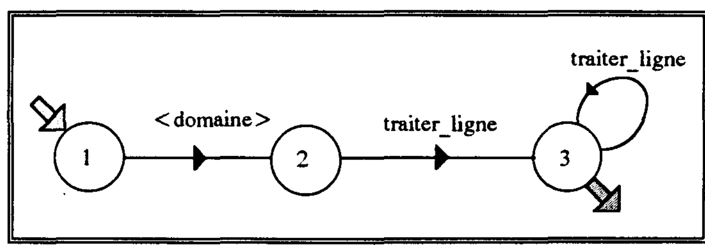


fig 4.17 : Automate pour l'analyse d'une description série & parallèle.

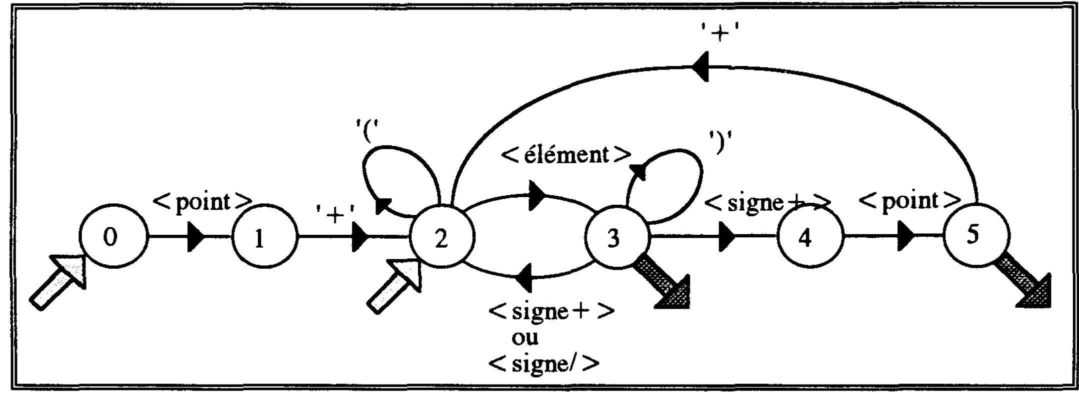


fig 4.18 : Automate pour le traitement d'une ligne en langage série et parallèle.

On peut remarquer que cet automate s'adapte à tous les cas de figures, quel que soit le domaine étudié, l'état **0** est initial et l'état **5** est final, cependant les états **2** et **3** peuvent devenir respectivement initial et terminal dans le cas d'un système mécanique.

Le traitement syntaxique de ce langage consiste à parcourir cet automate selon les faits suivants. Nous rappelons que l'état **3** est non déterministe.

transition(0 , point , 1)	*transition (5 , signe+ , 2)*
transition(1 , signe+ , 2)	*transition_bis (3 , signe+ , 4 , point)*
transition(2 , parenthèse_ouv , 2)	*transition_bis (3 , signe+ , 3 , élément)*
transition(2 , élément , 3)	*transition_bis (3 , signe+ , 3 , parenthèse_ouv)*
transition(3 , signe/ , 2)	*initial (0)*
transition(3 , parenthèse_ferm , 3)	*initial (2)*
transition(3 , signe+ , 4)	*final (3)*
transition (4 , point , 5)	*final (5)*

fig 4.19 : Ensemble des faits représentatifs de l'automate de la figure 4.18.

L'adaptabilité des éléments par rapport aux domaines étudiés caractérise la tâche principale de l'analyseur lexical qui, comme nous l'avons vu au chapitre III, se fait parallèlement à l'analyse syntaxique. Toute description qui ne respecte pas les règles de la grammaire et qui conduit à une interruption du parcours de l'automate, génère une erreur syntaxique. Voici une liste non exhaustive des erreurs traitées par le compilateur :

"Pas de domaine précisé" La première ligne à analyser ne précise pas de domaine.
"Une ligne doit commencer par un point (ex : m1, a1, ...)"
"Il manque N parenthèses fermantes)"
"Il manque N parenthèses ouvrantes ("
"Une ligne doit se terminer par un point"
"le terme X existe déjà plus haut"
"Elément inconnu"

Nous présentons en **Annnexe 3** quelques exemples d'erreurs syntaxiques.

IV.2.2.2. Analyse pré-sémantique

Pour le langage série & parallèle nous pouvons rencontrer, dans la description physique du système étudié, principalement 3 anomalies possibles qui mettent en cause le sens physique du système.

⇒ Lorsqu'une description ne contient aucune source d'énergie (effort ou flux).
Cette anomalie peut être néanmoins tolérée, en particulier lors de la création d'un bloc.(*)

⇒ Lorsque l'on rencontre dans une description plus d'une fois le même élément. Cependant en mécanique, les masses sont considérées comme des points et peuvent être citées plusieurs fois. Il suffit de tester la première occurrence de la masse pour ne pas l'enregistrer deux fois.

⇒ Lorsque dans un système électrique ou électronique, on trouve un transformateur **TFp** (primaire) sans son complément **TFs** (secondaire) ou réciproquement.

IV.2.2.3. Génération de code

Au chapitre II, nous avons donné schématiquement les principes qui conduisent à la création d'un bond-graph à partir d'une description texte. Dans ce paragraphe nous allons insister sur les méthodes informatiques utilisées à travers les différents domaines de la physique. Nous détaillerons en particulier le domaine électrique car il est facilement maîtrisable pour un érudit du langage.

A. Electrique

Dans le domaine électrique, les éléments sont traversés par un flux, l'intensité du courant, donc attachés à des jonctions **1**.

Les opérateurs '+' correspondent à des noeuds de tension, donc à des noeuds d'effort ; ils sont associés à des jonctions **0**.

Les points sont considérés comme des éléments particuliers attachés à des jonctions **0**.

Toute la difficulté réside dans la création des liens entre les jonctions. Pour faciliter la compréhension de la méthode, nous proposons d'étudier l'algorithme de génération à travers un exemple.

Les liens sont créés en respectant, pour l'opération parallèle, la topologie de la figure 2.62 du chapitre II.

(*) Voir le langage bloc bond-graph au paragraphe IV.3.1.

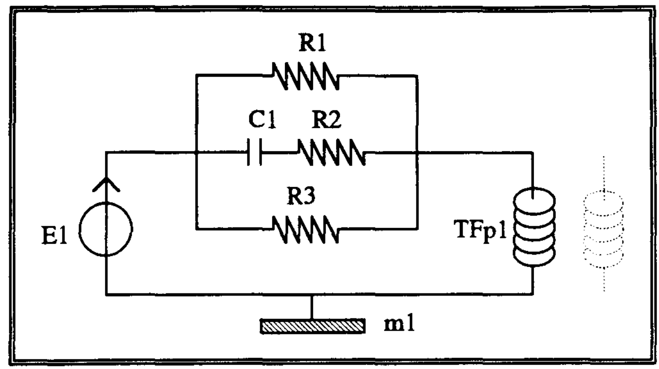


fig 4.20: Circuit électrique.

Par exemple, la description texte du circuit électrique de la figure 4.20 se traite de la manière suivante :

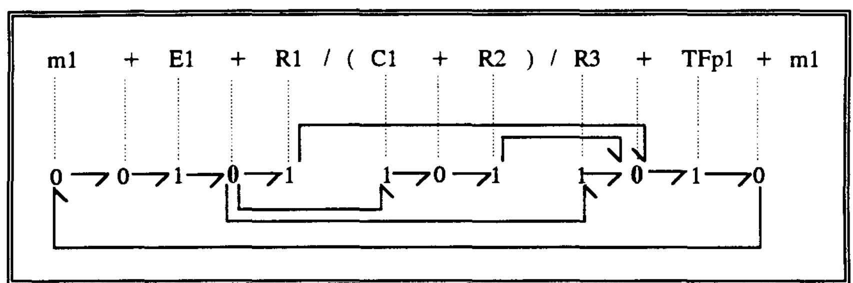


fig 4.21 : Corrélation entre la description et le bond-graph.

Ce bond-graph mis à plat donne après la phase de simplification le bond-graph de la figure suivante :

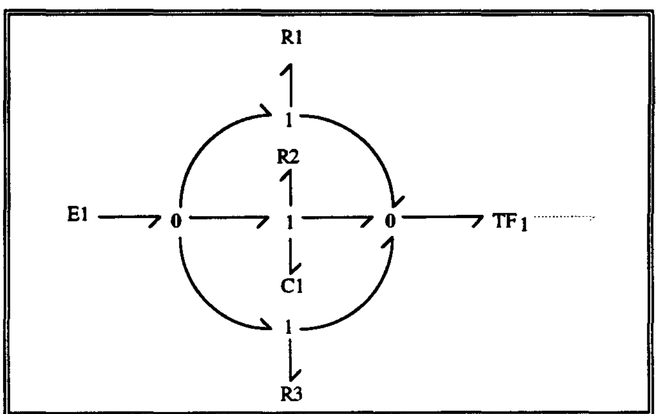


fig 4.22 : Bond-graph simplifié du circuit électrique.

Nous devons garder en mémoire d'une part les jonctions **0** qui précèdent les parties en parallèle pour pouvoir y brancher les jonctions **1** correspondant aux premiers éléments de ces blocs, et d'autre part les jonctions **1** correspondant aux derniers éléments de ces blocs qui restent en attente de branchement. Les faits suivants sont donc conservés en mémoire :

un(entier) : compteur de jonction 1
zéro(entier) : compteur de jonctions 0
niveau(entier) : niveau de parenthésage initialisé à 0 en début de ligne
dernière_jonction(nom) : nom de la dernière jonction d'attache
jonction_attente(nom, entier) : nom d'une jonction en attente de branchement et niveau de parenthèses auquel il se trouve
branchement(nom) : nom d'une jonction "adresse de branchement " ; la data-base de Turbo-Prolog est utilisée à la manière d'une pile LIFO, la dernière adresse de branchement est placée au sommet de la data-base et sera la première à sortir.

L'algorithme de construction du modèle bond-graph est présenté en annexe 3.

B. Electronique

La procédure utilise les mêmes règles que le domaine électrique en y ajoutant le traitement des éléments de commutations (switchs, diodes...).(*)

'élément de commutation'

A.8.1. On crée une jonction **1** avec une liste vide et une jonction **MTF** avec dans la liste l'élément de commutation **EC** correspondant.
A.8.2. On attache la jonction **1** et la jonction **MTF**.

1 ⇀ MTF ⇀ EC

A.8.3. On crée un lien entre la dernière jonction d'attache et la jonction **1** dans le sens.

dernière jonction → jonction **1** courante

A.8.4. On met la jonction **1** en attente de branchement avec le niveau courant.

Exemple :

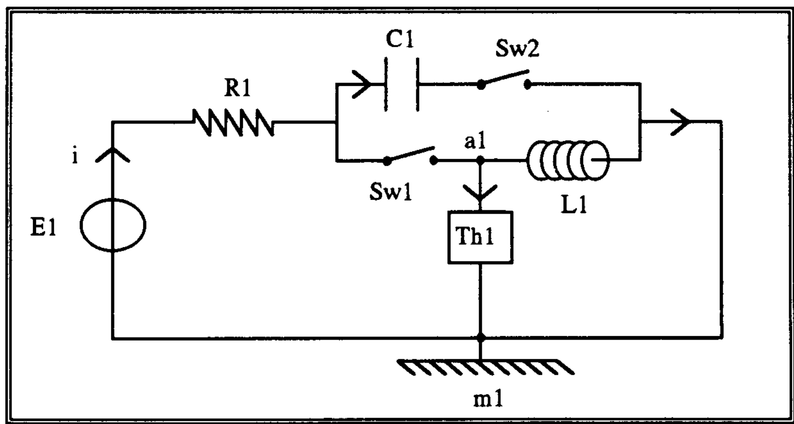
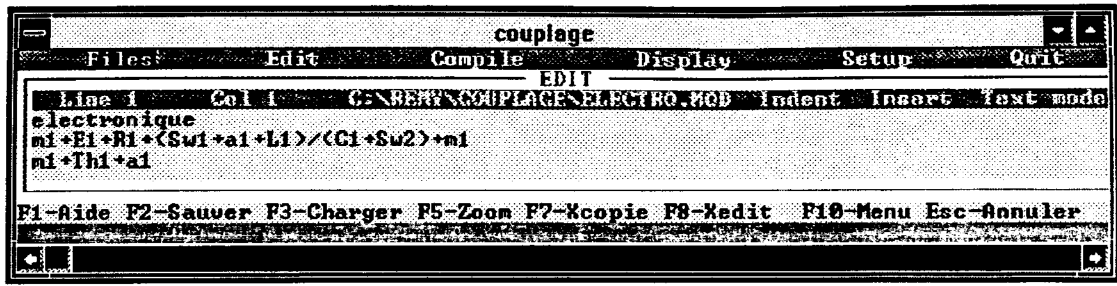


fig 4.23 : Circuit électronique avec deux switchs et un thyristor.

(*) Voir le 3.2.2 du chapitre II.



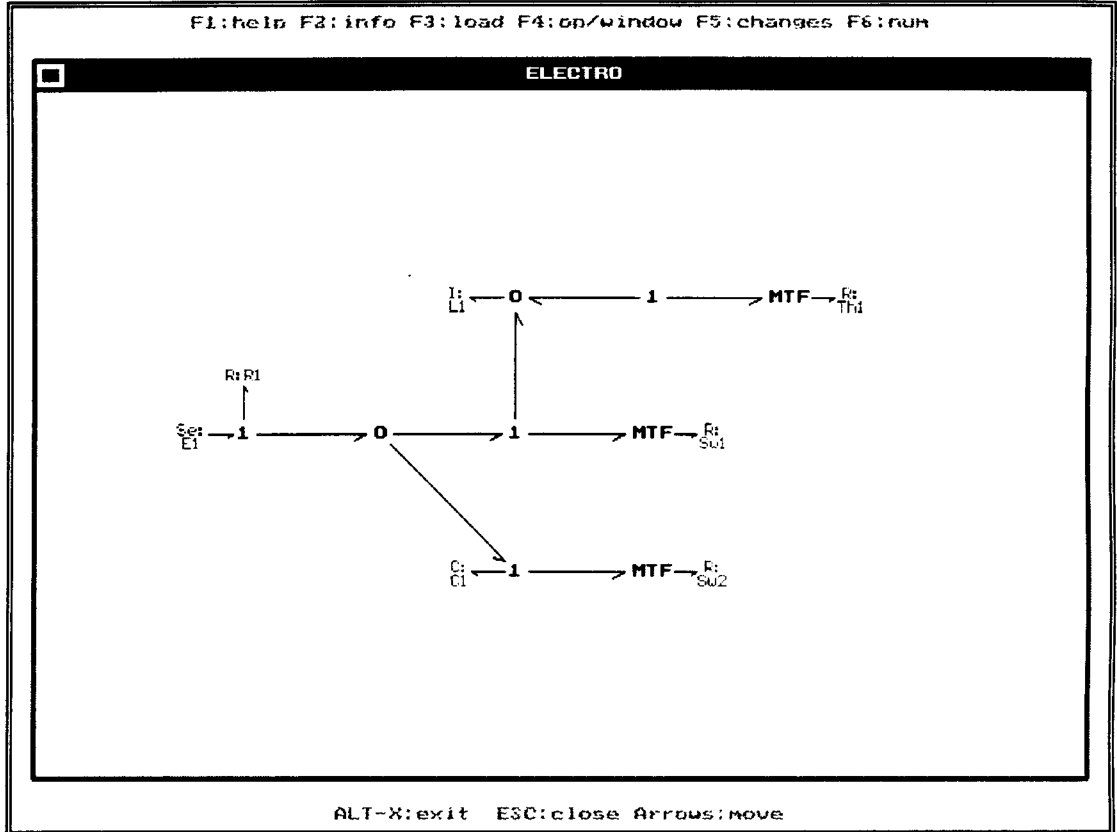


fig 4.24 : Dessin du bond-graph de la description du modèle électronique.

C. Mécanique

Le domaine mécanique (à une dimension) étant dual du domaine électrique, nous allons profiter de cette propriété pour utiliser un dérivé de la méthode.

Ainsi, pour les systèmes mécaniques, le noeud de vitesse se caractérise par une jonction **1**, et les notions séries et parallèles sont inverses de celles utilisées en électriques. Deux éléments en **série (+)** sont soumis à la même **force** (effort), deux éléments en **parallèles (/)**, ont la même **vitesse relative** (flux). La description se fait en suivant le sens de la référence.

La procédure est donc la même, il suffit de suivre les mêmes phases en permutant les jonctions **0** par les jonctions **1** avec quelques nuances :

- Les **masses** sont attachées à des noeuds de vitesses autrement dit des jonctions **1**,

- Les **points** sont considérés comme des masses nulles, donc attachés aussi à des jonctions 1.

- Les éléments **ressorts** et les **amortisseurs** sont associés à des jonctions **0**.

- Lorsqu'une force **F** (une vitesse **V**) est appliquée à une masse **M** ou un point, on attache à la jonction **1** de la masse une source d'effort **Se** (une source de flux **Sf**) ou à la jonction **1** associé au point.

- Dans le cas d'un frottement **Fr**, on attache à la jonction **1** un élément dissipatif **R** qui physiquement pourrait être représenté par un amortisseur attaché à la masse **M**.

Exemple :

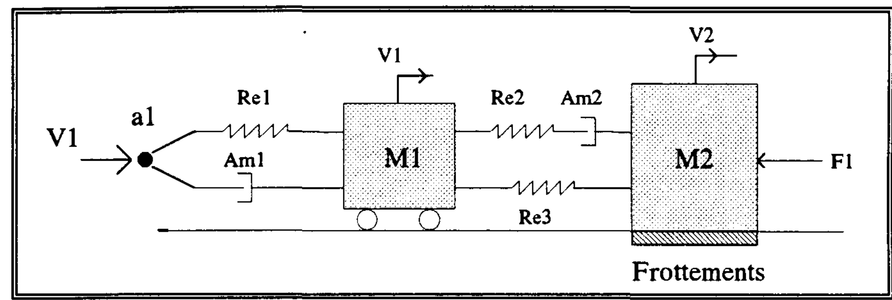
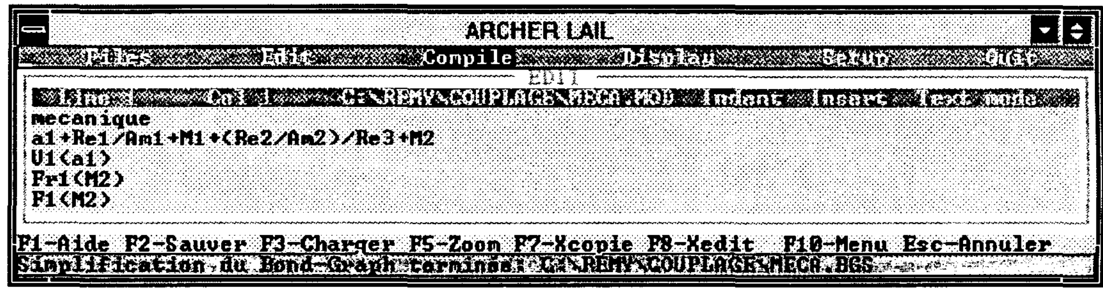


fig 4.25 : Modèle mécanique avec une force F1 et un frottement Fr1 sur la masse M2 et 1 source de vitesse appliqué sur le point a1.



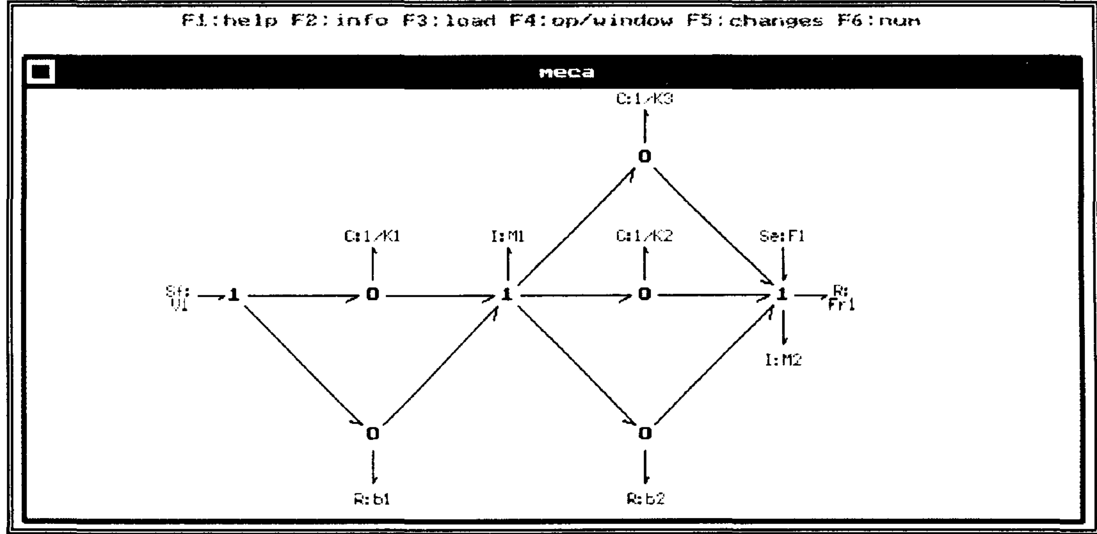
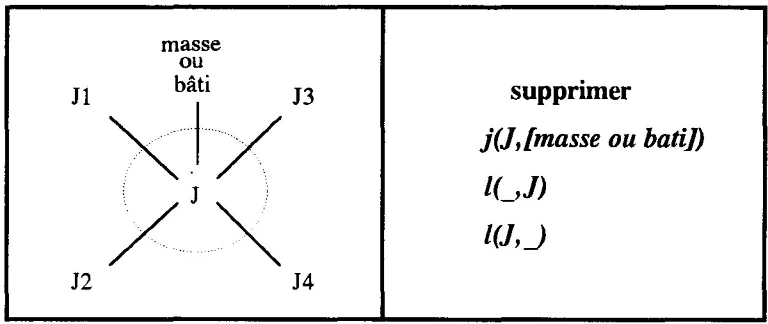


fig 4.26 : Bond-graph simplifié du système mécanique.

IV.2.2.4. Phase de simplification

Pour simplifier le bond-graph d'une manière optimale, il convient de scinder l'opération en trois étapes :

- Dans la première étape, les jonctions associées à des points de références (m) ou à des bâtis sont éliminées, ainsi que les liens qui leurs sont attachés. Les jonctions associées à des points ne sont pas encore éliminées



- La seconde consiste à effectuer les habituelles simplifications sur le reste du bond-graph.(*)

- Dans la dernière étape, les jonctions associées à des points s'éliminent dans le cas suivant : (voir règle 1)

j(J,[point]) *l(J1,J)* *l(J,J2)* *not(autre_lien(J1,J))* *not(autre_lien(J,J2))*	point J1 → J → J2	J1 → J2	**supprimer** *j(J,[point])* *l(J1,J)* *l(J,J2)* **ajouter** *l(J1,J2)*

Pour les autres jonctions, le point est éliminé de la liste.

$$jonction(J,[point]) \Rightarrow jonction(J,[])$$

Remarque : Les points ne sont pas éliminés directement lors de la génération de code car, lors de la fabrication d'un bloc en langage série & parallèle, certains points peuvent être déclarés comme des ports ***in*** ou ***out***. Les autres suivent le traitement décrit plus haut.(**)

(*) Voir chapitre II, simplification et chapitre III.

(**) Voir le paragraphe suivant 'création d'un bloc'.

IV.3. Création d'un bloc bond-graph

Définition : Le "bloc" est un objet qui se compose :

- D'un identificateur unique (nom alphanumérique).
- D'une sous-partie représentée par un type de modélisation.
- De connexions propres aux blocs appelées 'ports' possédant chacun une nature propre.

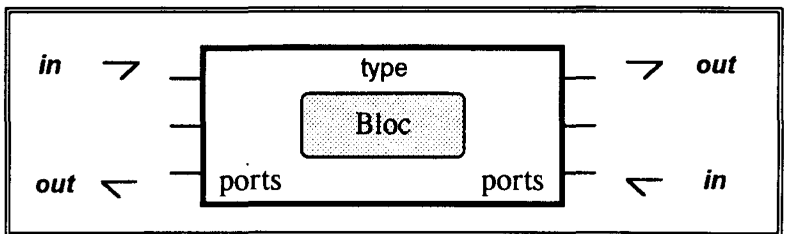


fig 4.27 : Organisation d'un bloc.

Les informations intrinsèques au bloc sont présentées sous la forme de prédicats. Chaque bloc a un nom (N) sans numérotation afin d'éviter toute ambiguïté à sa création et un type de modélisation (T). Lors du couplage les blocs utilisés sont affectés d'un **indice** correspondant à leur multiplicité.

Les données relatives à un bloc sont stockées dans 2 fichiers.

- Le premier fichier (*.bgs) contient le code bond-graph simplifié sous forme de prédicats ***jonction*** et ***lien*** générés à partir d'une description du bloc à l'aide du langage bond-graph ou série & parallèle. Cette partie devient "le corps du bloc".

- Le second fichier (*.blk) sert à spécifier les ports d'entrées (***in***) et de sorties (***out***) "autrement dit "les attaches du bloc", ainsi que le nom et le type du bloc par les trois faits suivants :

b(N,T) : **N** est le nom du modèle et **T** le type de représentation. ('bg' pour un bond-graph).

in(I,J,D) et ***out(I,J,D)*** : représentent les ports d'entrées et de sorties avec **I** le numéro du port, **J** la jonction liée au port, **D** le domaine du port.

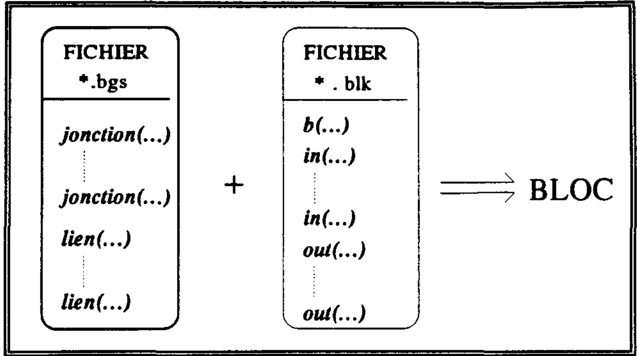


fig 4.28 : Prédicats utilisés pour la représention d'un bloc en mémoire.

Nous allons nous intéresser à la fabrication des prédicats du fichier *.blk dans le cas d'un bloc en langage bond-graph et d'un bloc langage en série & parallèle.(*)

IV.3.1. Bloc en langage bond-graph

Ce langage permet d'affecter des ports d'entrées et de sorties sur n'importe quelle jonction d'un modèle bond-graph pour former un bloc. Celui-ci peut désormais ne contenir aucune source d'énergie car il peut, par la suite, être couplé à d'autres blocs afin de constituer un modèle bond-graph plus complexe.

IV.3.1.1. Analyse syntaxique

Pour définir un bloc, il suffit d'ajouter, à la structure des parties 'jonction' et 'lien' du bond-graph, une partie 'port' contenant l'information des ports attachés aux jonctions avec les métanoms suivant :

<mot_clef> ::= | 'port'
<port> ::= <in_out> <nombre_entier>
<in_out> ::= 'in' | 'out' — **e : electrique, el : électronique, h : hydraulique**
<domaine> ::= e | el | h | mt | mr — **mt, mr : mécanique de translation, de rotation**

La grammaire utilisée est la suivante :

G3(A,{<ligne_port>,<liste_port>,<info_port>...},P3,<ligne_port>)

Elle utilise les règles de production P3 suivantes :

<ligne_port> ::= <jonction> <liste_port>
<liste_port> ::= <liste_port> <info_port> | ε
<info_port> ::= <port> ':' <domaine>

(*) Voir le chapitre II : 'blocs et ports'.

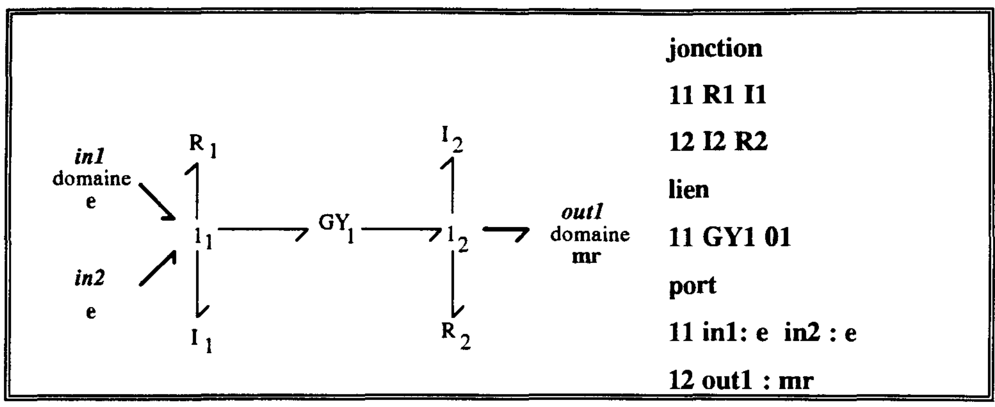


fig 4.29 : Exemple d'un bloc bond-graph.

Il suffit d'ajouter 3 états à l'automate du langage bond-graph de la figure 4.1, afin de traiter la partie port qui s'identifie à l'aide du mot clef 'port'.

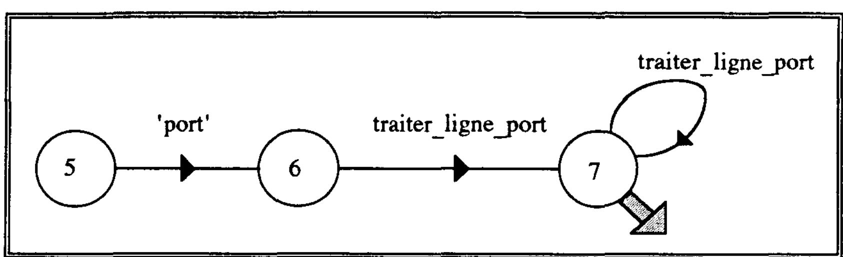


fig 4.30 : Automate de la partie port.

La transition **traiter_ligne_port** émule l'automate de la figure 4.31 construit autour des règles de production P3.

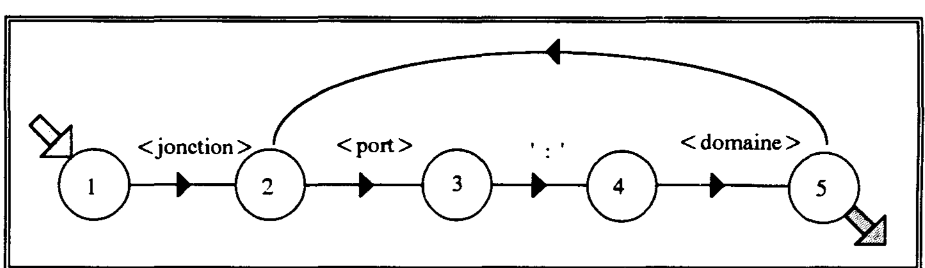


fig 4.31 : Automate pour le traitement d'une ligne port.

L'automate se traduit par les faits suivants :

<u>**Automate de la partie port**</u>

transition (1 , jonction , 2)

transition (2 , port , 3)

transition (3 , : , 4)

transition (4 , domaine , 5)

initial (1) final (5) nulle (5 , 2)

fig 4.32 : Transitions de l'automate de la partie port.

IV.3.1.2. Analyse pré-sémantique

Elle reprend les mêmes fonctions que pour le langage bond-graph seul, augmentée de quelques nouvelles analyses sur la partie réservée à l'affectation des ports.

Lorsque nous associons un domaine physique à un port, nous signalons les différences de domaines sur une même jonction **0** ou **1**.

Pour les jonctions **TF** et **GY** on autorise un seul port en entrée et un seul port en sortie avec des domaines physiques différents (ou égaux), puisque le rôle des transducteurs est justement de se comporter comme le médiateur de la puissance entre deux domaines différents.

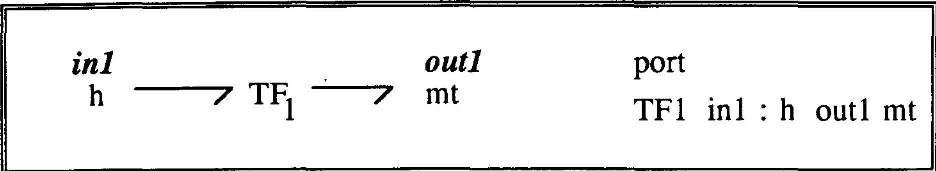


fig 4.33 : Bloc formé d'un transformateur.

IV.3.1.3. Génération de code

A la procédure qui génère le code bond-graph, nous ajoutons l'algorithme suivant qui crée les faits ***in*** et ***out***.

'J $P_i : D_i$ $P_j : D_j$ ...' J ∈{ 0 , 1 , TF , GY]

Lire le premier lexème **LX0** qui représente la jonction.

Extraire les trois lexèmes suivants : **LX1**, **LX2**, **LX3**.

> **LX1** correspond à un port $\mathbf{P_i}$ où **P** peut représenter le mot 'in' et 'out' et **i** un indice numérique.
>
> **LX2** au séparateur ':'
>
> **LX3** le domaine $\mathbf{D_i}$ associé au port $\mathbf{P_i}$.

Décomposer le lexème **LX1** en une composante alphanumérique **P** et numérique **I.** (*)

> ***décompose_lexème(LX1,P,I)***
>
> Si **P** est un port 'in' on crée le fait ***in(I,LX0,LX3).***
>
> Si **P** est un port 'out' on crée ***out(I,LX0,LX3).***

Recommencer avec trois autres lexèmes.

Traiter la ligne suivante.

(*) Voir prédicat ***décompose_lexème*** au chapitre III.

Nous pouvons appliquer ce traitement au bloc bond-graph de la figure 4.28.

'11 in1 : e in2 : e'	***in(1,11,e)*** ***in(2,11,e)***

'12 out1 : mr'	***out(1,12,mr)***

Ces faits sont sauvegardés dans un fichier *.blk.

IV.3.1.4. Exemple d'un bloc hydro_mécanique

Prenons la pompe hydraulique de la figure 4.34 qui possède trois ports, une entrée hydraulique, et deux sorties, une hydraulique et l'autre en mécanique de translation.

Les surfaces **A1** et **A2** de part et d'autre du cylindre ne sont pas égales, il en résulte que les pressions **P1** et **P2** ainsi que les débits **Q1** et **Q2** ne le sont pas non plus.

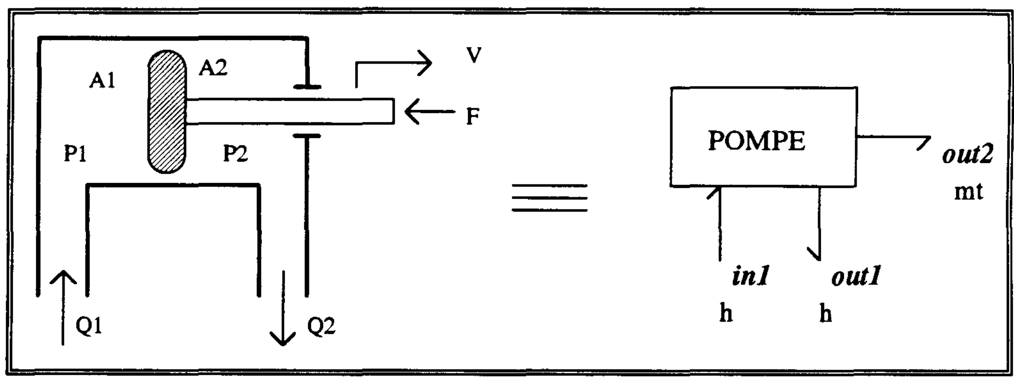


fig 4.34 : Schéma physique et représentation bloc de la pompe.

Pour modéliser un tel bloc en bond-graph on peut négliger en première approximation les frictions et les inerties mécaniques du piston, ainsi que les effets de compressibilité hydraulique. On obtient le modèle bloc bond-graph de la figure 4.35 avec la description texte suivante.

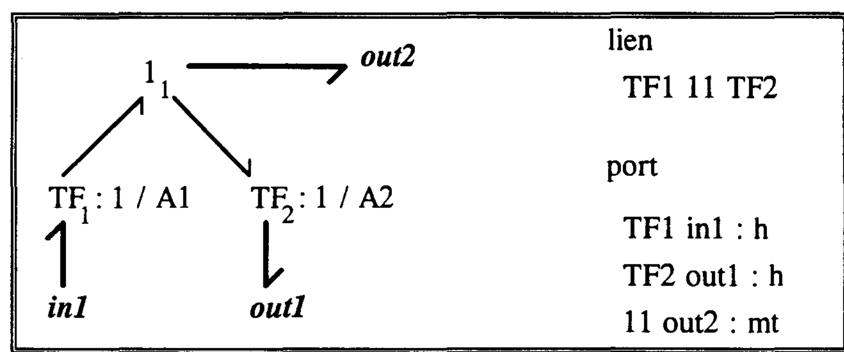


fig 4.35 : Bloc bond-graph et description de la pompe.

En poussant la modélisation plus loin, nous pouvons affiner le modèle en ajoutant quelques liens et des éléments dynamiques au schéma précédent, soit une inertie **I1** et une résistance **R1** pour prendre en compte la masse et l'action du piston, deux capacités **C1** et **C2** pour les effets de compliance au niveau des conduits 1 et 2, le tout combiné à une résistance **R2** pour l'interaction résultante. Nous obtenons le bloc bond-graph et la description texte suivants. Après compilation, nous obtenons les codes de la figure 4.38 stockés dans une base de données.

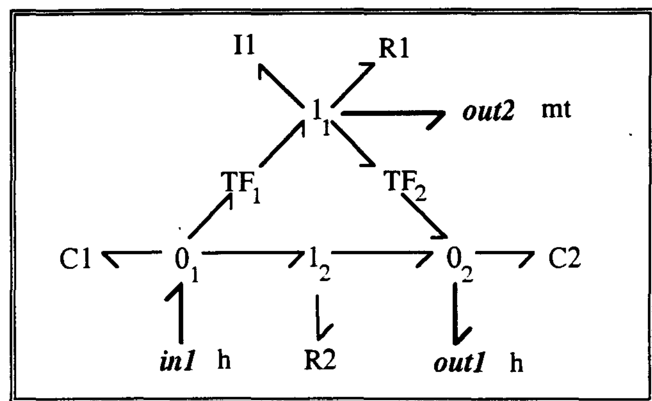


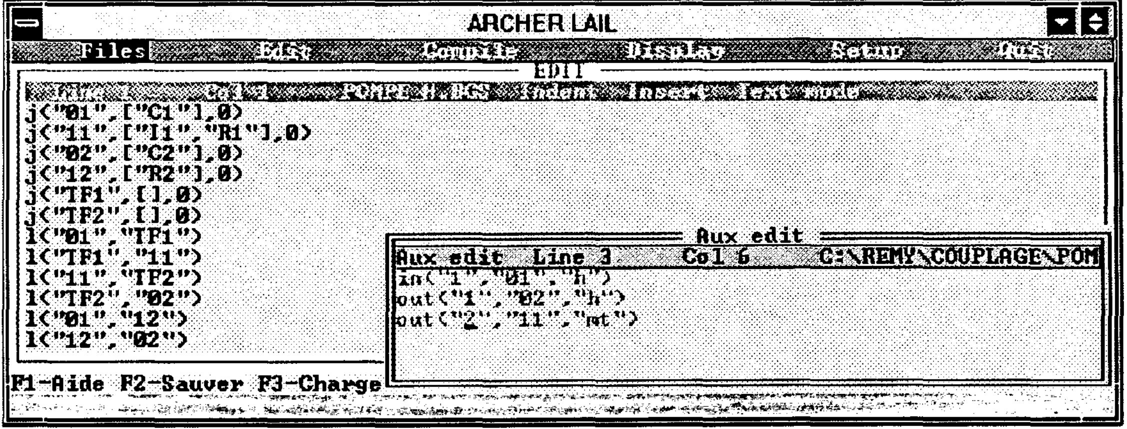
fig 4.36 : Bloc bond-graph affiné de la pompe.

```
ARCHER LAIL
Files  Edit  Compile  Display  Setup  Quit
EDIT
jonction
01 C1
11 R1 I1
02 C2
12 R2

lien
01 TF1 11 TF2 02
01 12 02

port
01 in1:h
11 out2:mt
02 out1:h_
F1-Aide F2-Sauver F3-Charger F5-Zoom F7-Xcopie F8-Xedit F10-Menu Esc-Annuler
PAS D'ERREUR DE SYNTAXE. CONSTRUCTION DU BOND-GRAPH EN COURS ...
```

fig 4.37 : Description texte du bloc pompe.



```
ARCHER LAIL
Files  Edit  Compile  Display  Setup  Quit
EDIT
j("01",["C1"],0)
j("11",["I1","R1"],0)
j("02",["C2"],0)
j("12",["R2"],0)
j("TF1",[],0)
j("TF2",[],0)
l("01","TF1")
l("TF1","11")
l("11","TF2")
l("TF2","02")
l("01","12")
l("12","02")
F1-Aide F2-Sauver F3-Charge

Aux edit
Aux edit  Line 3  Col 6  C:\REMY\COUPLAGE\POM
in("1","01","h")
out("1","02","h")
out("2","11","mt")
```

fig 4.38 : Code du bloc bond-graph : fichier *.bgs et *.blk.

Pour visualiser les ports du bloc bond-graph à l'écran, nous utilisons toujours le module graphique avec quelques modifications.

En effet, les ports sont considérés comme des jonctions provisoires que l'on nomme **Pi** pour les ports en ***in*** et **Po** pour les ports en ***out***, affectés de l'indice correspondant comme le montre la figure 4.39.

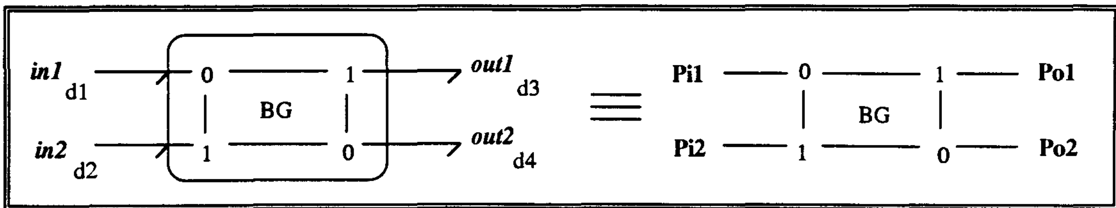


fig 4.39 : Représentation des ports in et out lors de l'affichage.

Nous avons créé un module qui traduit ces informations en un texte, sauvegardé dans un fichier *.hlp. Celui-ci peut être consulté à volonté par l'utilisateur et être enrichi par d'autres informations sur le bloc lui-même (utilisation avec d'autres blocs, contexte de validité...). C'est en quelque sorte la carte d'identité du bloc.

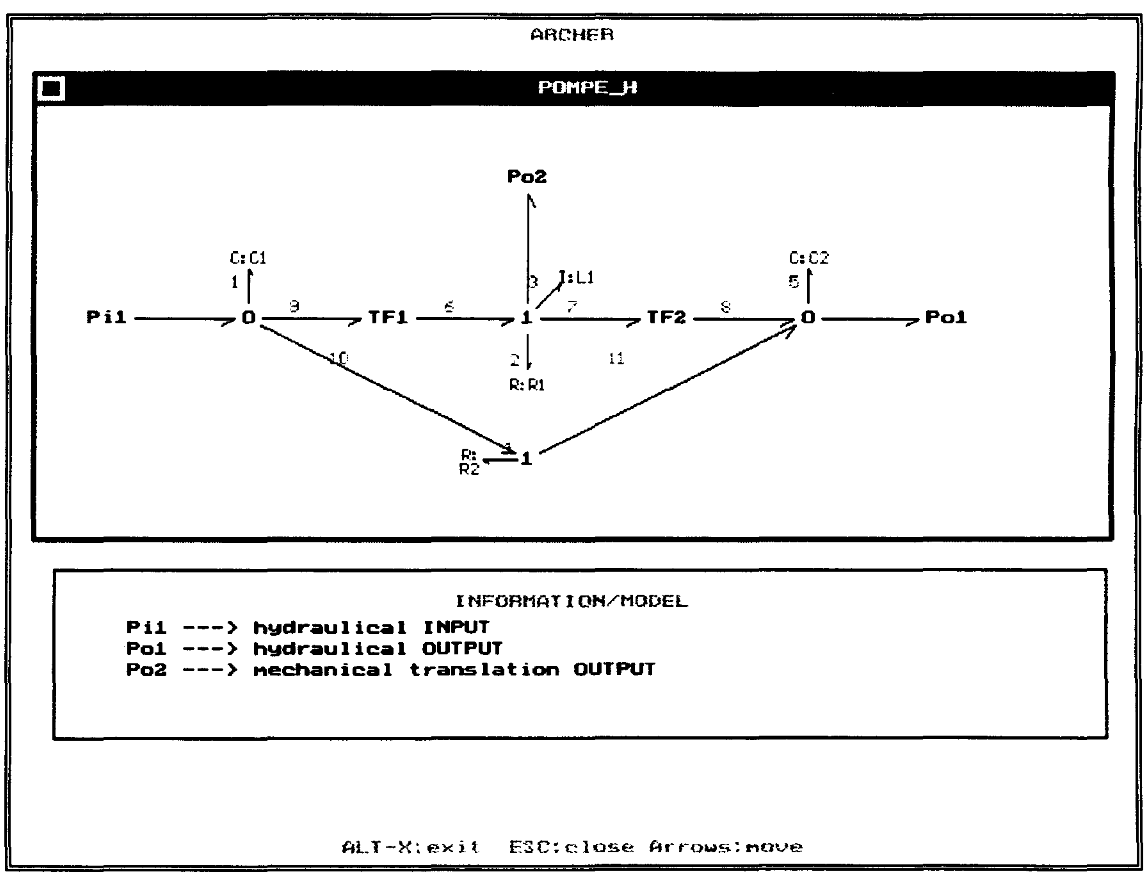


fig 4.40 : Dessin du bloc POMPE.

IV.3.2. Bloc en Langage série & parallèle

Il est possible de créer un bloc avec le langage série & parallèle en ajoutant à la description des points de connexion aux endroits voulus par l'utilisateur. A l'aide de la partie 'port' il suffit d'affecter les ports ***in*** et ***out***, aux points de connexions, sans ajouter cette fois ci le domaine physique puisque par définition il est connu dés le début de la description.(*)

IV.3.2.1. Analyse syntaxique

La grammaire de cette partie est la suivante

G2(A,{<ligne_port>,<liste_port>,<port>},P2,<ligne_port>)

Les règles de production P2 sont les suivantes :

<lIgne_port> ::= <point> '=' <liste_port>

<liste_port> ::= <liste_port> <port> |ε

On retrouve l'habituel automate de fonctionnement de la figure 4.17 et l'automate suivant qui analyse une ligne de la partie port lorsque la transition **traiter_ligne_port** est activée.

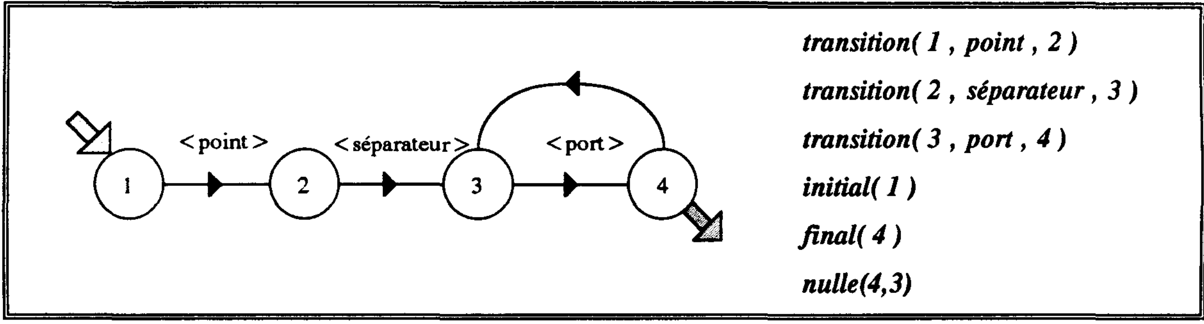


fig 4.41 : Automate pour le traitement d'une ligne de la partie port.

IV.3.2.2. Génération de code

La procédure est calquée sur celle de la partie port du langage bond-graph, à la différence que la jonction sur laquelle le port va se greffer est remplacée par le point de connexion. Nous avons vu comment la description série & parallèle génère un code bond-graph non simplifié dans lequel les points de connexion sont attachés à des jonctions **0** ou **1**, selon le domaine physique : ***jonction(J , [point])***

(*) Voir paragraphe II.3.5.1.

Les ports attachés à un point sont donc attachés à la jonction liée à ce même point, on peut alors générer les faits ***in*** et ***out*** correspondants. Chaque port est défini par défaut selon le domaine physique sollicité lors de la description.

```
                         'point = Pi Pj Pk ...'

Créer les faits in et out à partir de cette procédure :
Rappeler domaine courant à l'aide du fait domaine(D)
Pour chaque ligne
        Lire le premier Lexème : LX1
        Rappeler un fait jonction( J , [LX1] )
        Mettre la jonction dans la liste L du fait jonction_port(L)
        Lire le lexème suivant : LX2 < > '='
        Pour chaque port Pi
                décompose_lexème( Pi , P , i )
                Si Pi = 'in'
                        créer le fait in ( i , J , D )
                Si Pi = 'out'
                        créer out ( i , J , D )
```

A présent il nous reste à simplifier, dans le code bond-graph, les points définis lors de la description et qui n'ont pas été utilisés dans la partie 'port' tout en gardant les jonctions des points qui servent à définir les ports. On remplace la liste de ces jonctions par une liste vide.

jonction(J , [point]) ⇒ jonction(J , [ ])

IV.3.2.3. Exemple d'un bloc électrique

Nous proposons de reprendre l'exemple du bloc électrique de la figure 2.75 que nous appelons RLC.

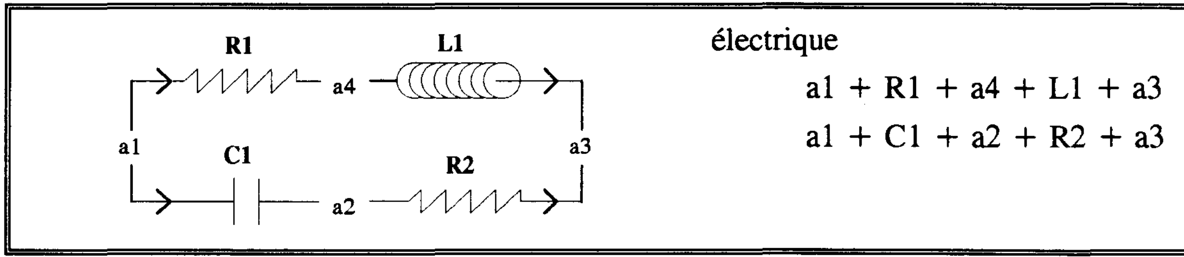


fig 4.42 : Bloc RLC.

A partir de cette description, nous obtenons le code bond-graph non simplifié suivant :

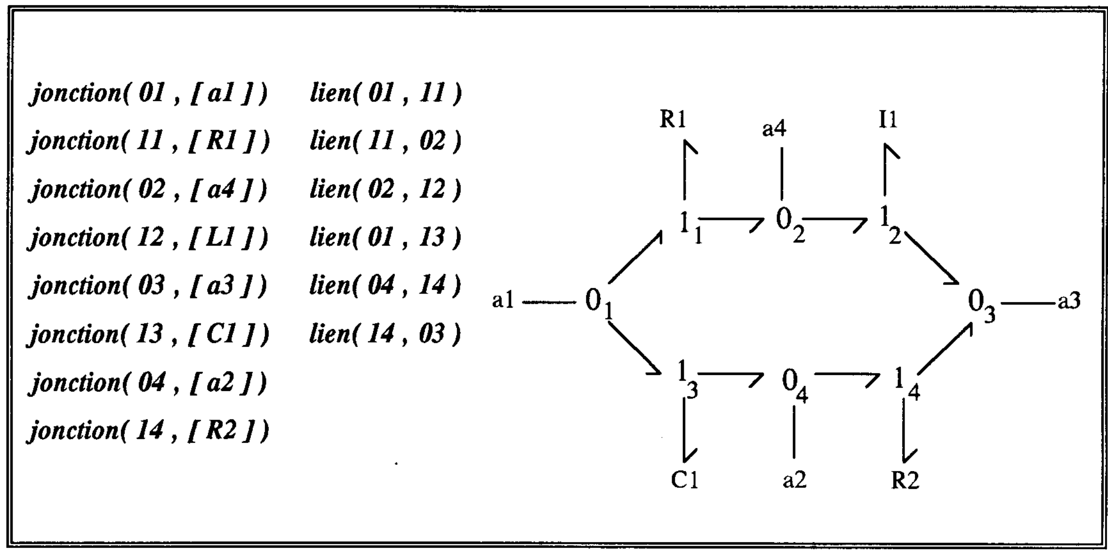


fig 4.43 : Code et dessin du bond-graph non simplifié.

Le domaine physique étant électrique, nous avons le fait ***domaine(e)***.

A partir de cette description et selon la déclaration des points dans la partie port, nous pouvons former plusieurs blocs RLC.

Exemple : Un bloc avec une entrée et une sortie

port
a1 = in1
a3 = out1

- Le point **a1** est porté par la jonction **01**.
- Le port **in1** est affecté à ce point à fortiori la jonction **01** avec comme nature physique le domaine électrique (e).
- On fabrique le fait ***in(1 , 01 , e)***.
- De même pour le point **a3** nous avons ***out(3 , 03 , e)***.
- On garde une trace de ces jonctions grâce au fait ***jonction_port([01 03])***
- On simplifie le bond-graph de façon à garder les jonctions **01** et **03**.

Nous obtenons le bloc-bond-graph suivant. (en arrière plan nous avons le bond-graph non simplifié de la figure précédente.

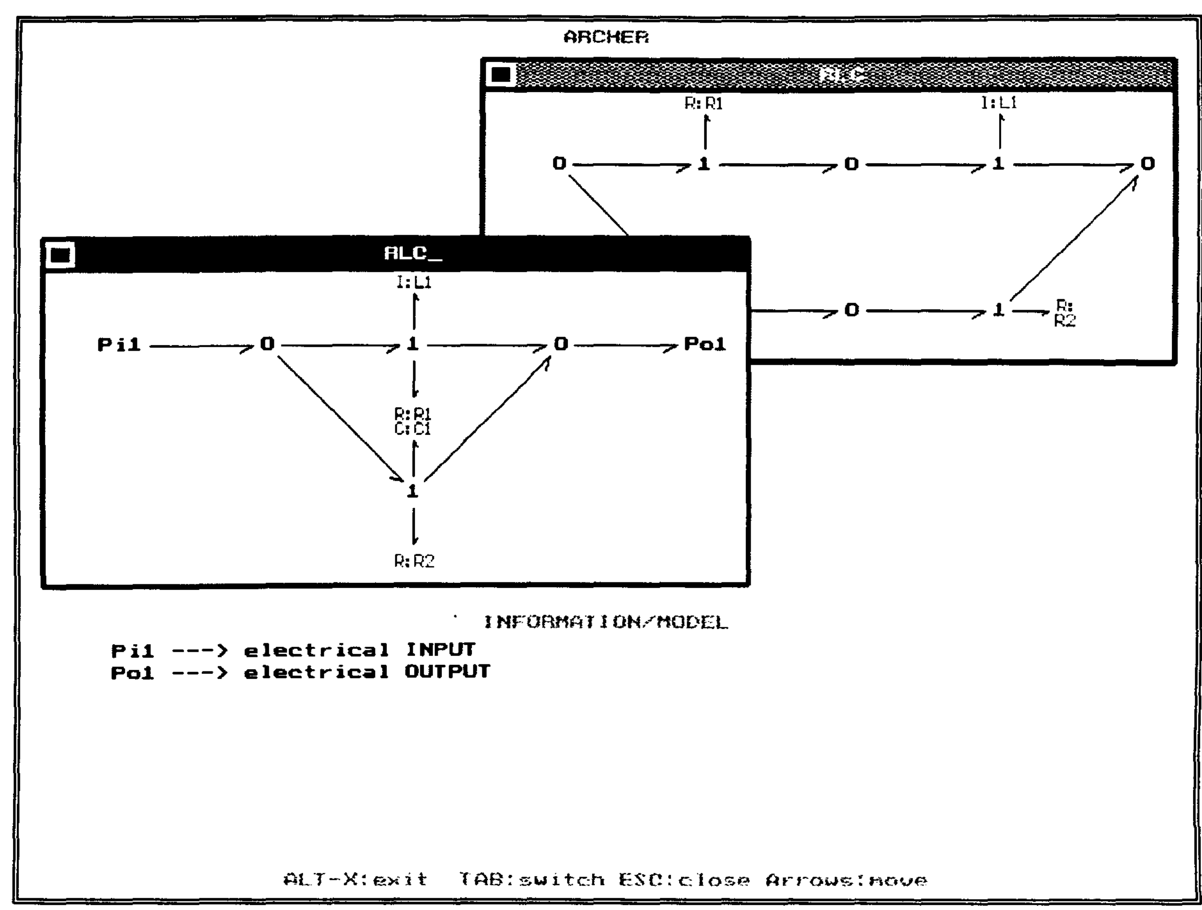


fig 4.44 : Bloc RLC avec 2 ports.

IV.4. Langage Blocs & noeuds

Ce langage génère un bond-graph unique à partir d'une description texte d'un modèle formé par le couplage de blocs préalablement définis. Nous rappelons que ce langage se compose d'une déclaration des blocs et d'une description toplogique de la structure composée de noeuds (**Ne** ou **Nf**) et de blocs (**Bi**).

Pour faciliter la compréhension des différentes analyses, nous illustrons la démarche à travers un exemple de modèle composé de quatres blocs :

Un bloc de type **ALPHA** à 1 sortie de nature α1.
Un bloc de type **BETA** à 1 entrée et 2 sorties de nature respectives β1, β2 et β3.
Deux blocs de type **GAMMA** à 2 entrées de nature γ1 et γ2.

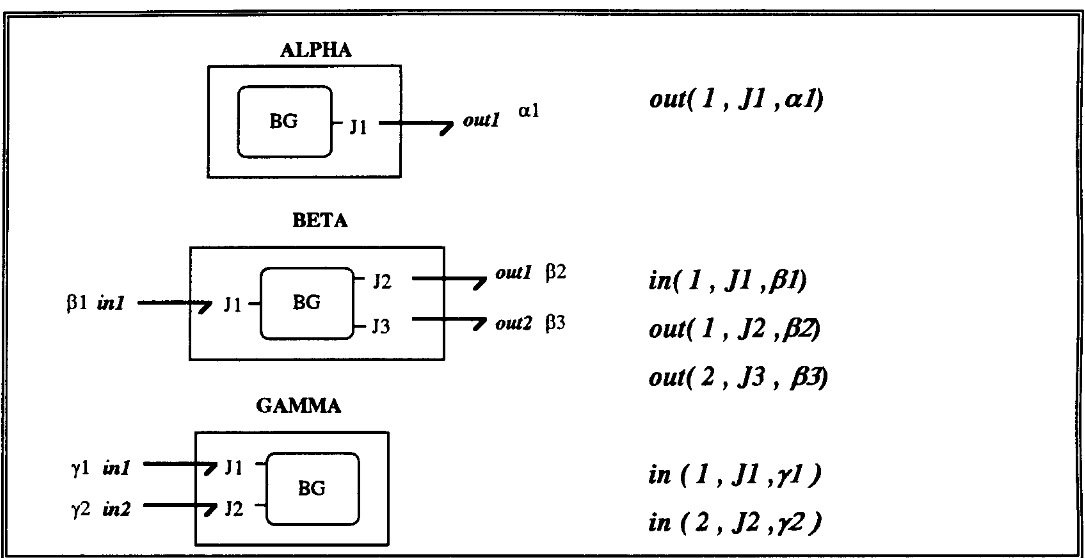


fig 4.45 : Exemple de blocs pré-définis.

L'utilisateur imagine une structure formée à partir des blocs définis plus haut. Par exemple le bloc **Alpha1 (B1)** est couplé au bloc **Béta1 (B2)** par un noeud d'effort (**Ne1**), celui ci est couplé à 2 blocs **Gamma1 (B3)** et **Gamma2 (B4)** par un noeud de flux (**Nf1**). Nous obtenons la figure 4.46.

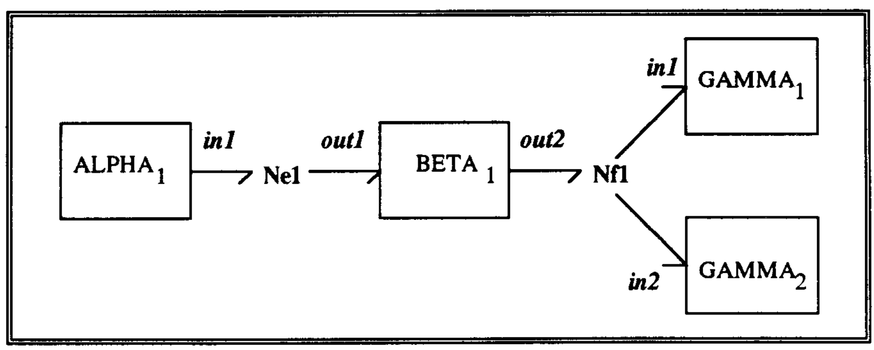


fig 4.46 : Modèle constuit à partir des blocs.

Remarque : Pour l'instant, seuls les ports d'attaches nous importent, autrement dit la connaissance du fichier *.blk de chaque bloc. La constitution bond-graph interne des blocs intervient plus tard.

IV.4.1. Analyse syntaxique

IV.4.1.1. Grammaire du langage

Le langage se définit par une partie déclaration et une partie description. A la manière du langage bond-graph, nous pouvons séparer la grammaire en deux sous-grammaires basées autour du vocabulaire non terminal suivant.

<bloc>	**:: = 'B' <nombre entier>**	**<nom>**	**:: = <liste_alpha>**
<noeud>	**:: = <nod> <nombre entier>**	**<liste_alpha>**	**::= <lettre> <liste_alpha>**
<nod>	**::='Ne' | 'Nf'**	**<caractère>**	**::= A | B | ... | Z**

Les sous-grammaires sont composées des règles de production suivantes :

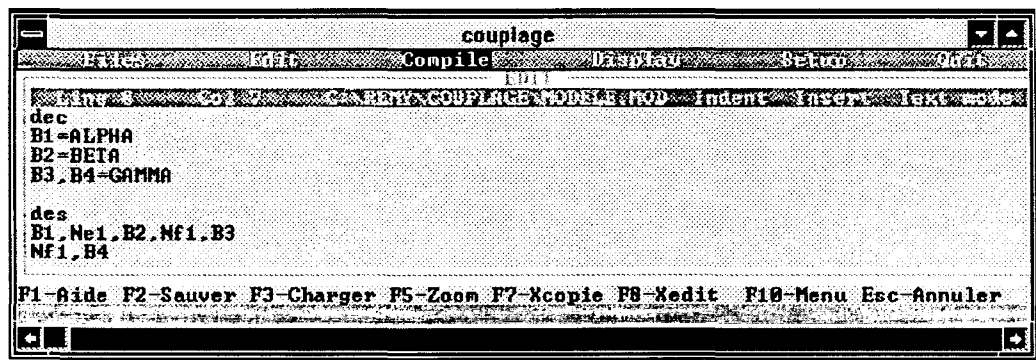
Partie déclaration

<ligne_dec> ::= <liste_bloc>' =' <nom>
<liste_bloc> ::= <bloc> ',' <liste_bloc> | ε

Partie description

<ligne_des> ::= <liste_n_b> | <liste_n_b> <noeud> | ε
<liste_b_n> | <liste_b_n> <bloc> | ε
<liste_n_b> ::= <noeud> <bloc> <liste_n_b> | ε
<liste_b_n> ::= <bloc> <noeud> <liste_b_n> | ε

Dans la partie description, un noeud peut-être répété plusieurs fois à la manière des points du langage série & parallèle. Nous entrons la description de la figure 4.46 à l'aide de l'éditeur d'Archer :



Nous lançons la compilation, l'analyse des différentes lignes se fait selon l'automate suivant.

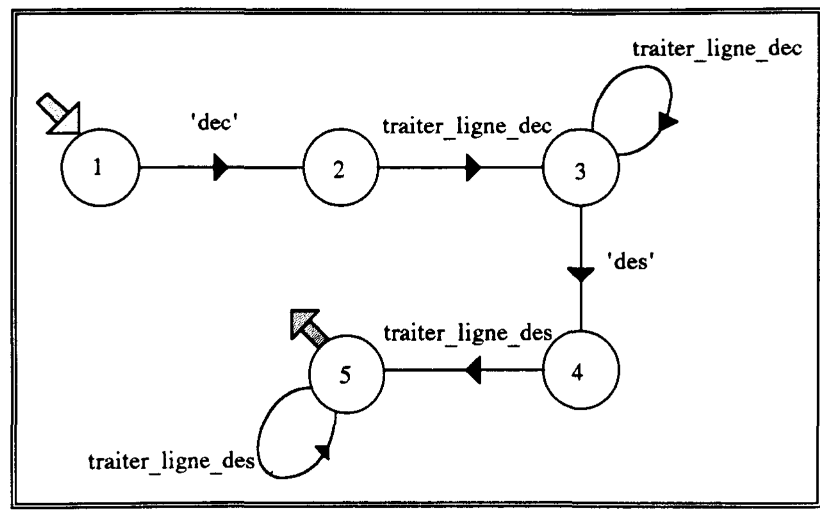


fig 4.47 : Automate pour le traitement du langage blocs & noeuds.

Comme pour les autres langages, chaque transition active un automate pour l'analyse syntaxique de la ligne courante.

L'automate associé à la transition **traiter_ligne_dec** s'occupe de l'analyse syntaxique de la partie "déclaration" des blocs.

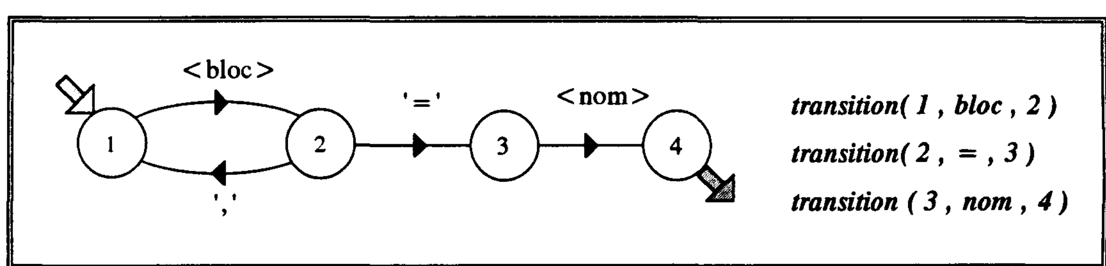


fig 4.48 : Automate pour le traitement d'une ligne de déclaration.

De même pour la transition **traiter_ligne_des**, nous avons l'automate de la partie description. Pour chaque cas, nous donnons le code traduisant leur structure en mémoire.

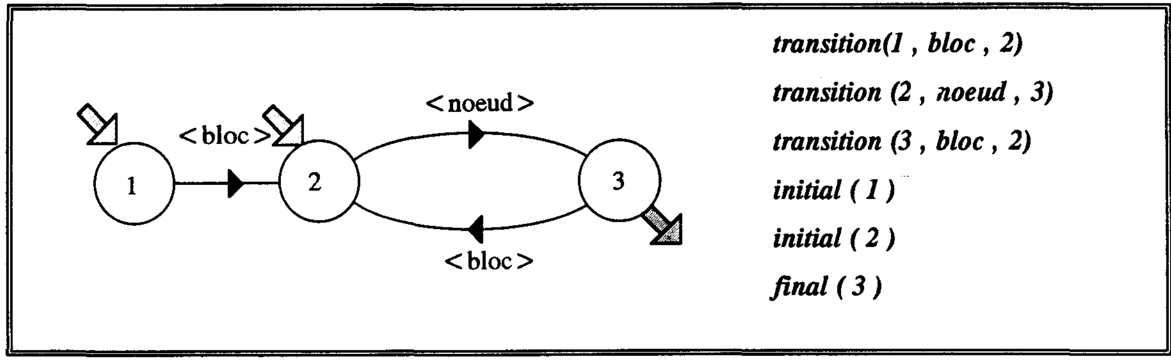


fig 449 : Automate pour le traitement d'une ligne de description.

IV.4.1.2. Erreurs syntaxiques :

Le but n'est pas de donner une liste exhaustive des erreurs qu'il est possible de faire, mais de nous donner les outils et les moyens pour les détecter. Libre au programmeur d'enrichir les diagnostics et les messages associés.

Comme pour les langages vus précédemment, on distingue les erreurs syntaxiques simples qui correspondent aux états puits de l'automate, par exemple dans la phrase de la partie déclaration suivante 'B1 ALPHA' il manque un signe '=' aprés B1.

Remarque : Le cas suivant n'est pas considéré comme une erreur de description :

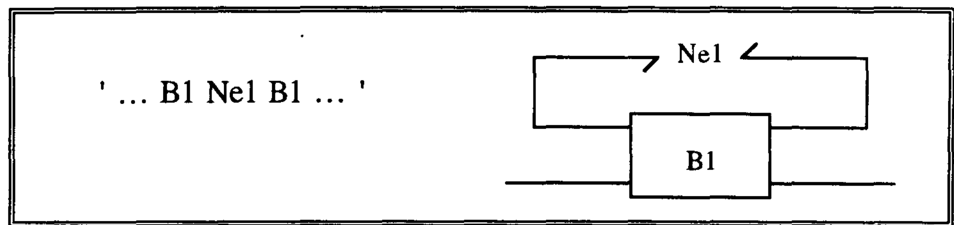
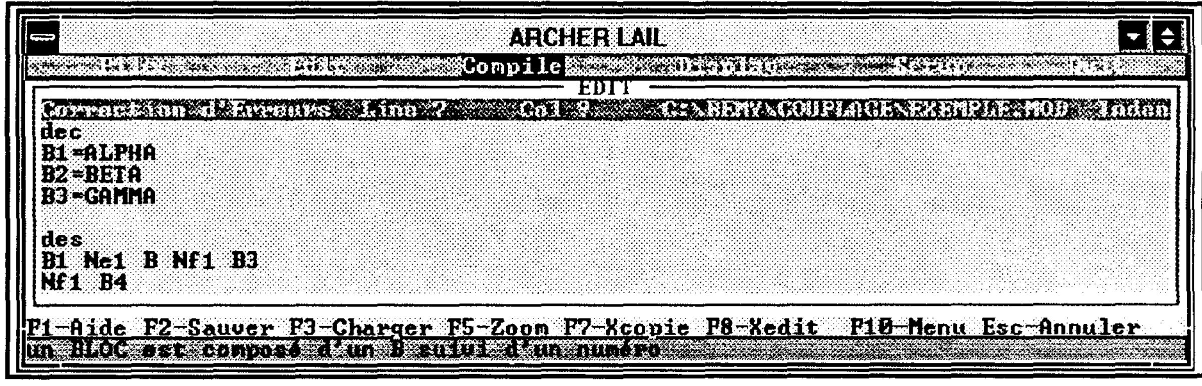
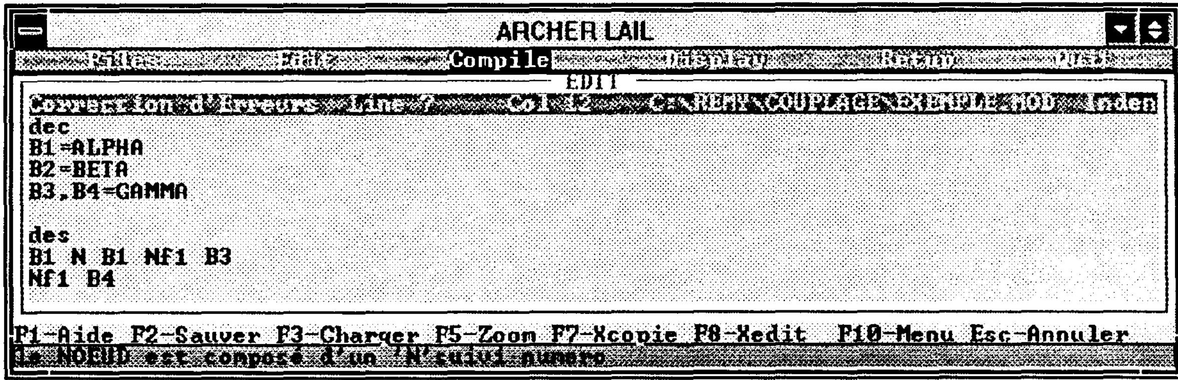


fig 4.50 : Noeud reliant les ports d'un même bloc.

En effet une des sorties peut être connectée à l'une des entrées d'un même bloc.

Voici quelques erreurs classiques détectées lors de l'analyse syntaxique.





IV.4.2. Analyse pré-sémantique

Lorsque l'analyse syntaxique du texte est correcte, il reste à identifier, pour chaque bloc, le numéro du port ***in*** ou ***out*** attaché au noeud à travers la phase de "sélection des ports".

Cette phase demande une réorganisation de l'information en un code intermédiaire composé de nouveaux faits qui sont générés à partir de la déclaration, des fichiers *.blk de chaque bloc et de la description elle-même.

Ce code intermédiaire va permettre de gérer facilement la sélection des ports qui s'effectue parallèlement à l'analyse sémantique. De plus, il nous renseigne sur des erreurs qui ne peuvent être détectées lors de l'analyse syntaxique.

IV.4.2.1. Partie déclaration

A partir de la déclaration des blocs, nous avons accés à plusieurs informations que nous allons classer dans les faits suivants :

présent(L) : L est la liste des types de blocs (sans indice de multiplicité) présents dans la partie déclaration.

Par exemple il y a trois types de blocs présents dans la description de notre modèle de la figure 4.46 : ALPHA, BETA, GAMMA.

dec
B1 = ALPHA
B2 = BETA ***présent([ALPHA BETA GAMMA])***
B3, B4 = GAMMA

fig 4.51 : Fait relatif à la partie déclaration de notre exemple.

équivalent(N, Nn, Bn) : **N** est le nom du modèle, **Nn** est le modèle avec son indice de multiplicité, **Bn** le bloc qui lui est affecté.

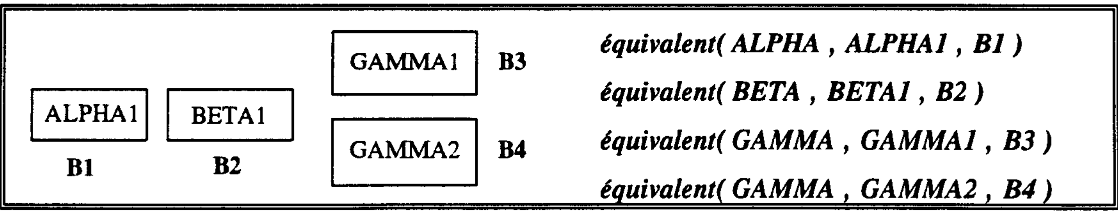


fig 4.52 : Ensemble de faits qui traduisent la multiplicité des blocs.

A partir de ces faits nous détectons plus facilement les erreurs qui demandent une analyse approfondie du contexte. Par exemple, dans la partie déclaration nous pouvons avoir :

1. Une erreur si un des types de bloc n'a pas été pré-défini.
Cette erreur est facilement détectable, il suffit de vérifier si le fichier *.blk existe.

2. Une erreur de redondance, si un bloc (**B_n**) est utilisé pour la déclaration de plusieurs modèles. Par exemple le bloc **B2** sert à définir les modèles **BETA** et **GAMMA**

- Pour détecter les redondances, nous comptons les occurrences du fait ***équivalent*** pour un bloc **B_n** donné et si ce nombre est supérieur à 1, il y a une erreur de répétition.

équivalent(BETA , BETA1 , B2)

équivalent(GAMMA , GAMMA1 , B2)

Les faits ***présent*** et ***équivalent*** servent à structurer les informations sur les ports ***in*** et ***out*** de chaque bloc à travers les faits ***bloc_port_in(Bn,N,J,D)*** et ***bloc_port_out(Bn,N,J,D)*** qui sont constuits à partir des faits ***in(N,J,D)*** et ***out(N,J,D)*** de chaque bloc **Bn**.

Pour chaque type de bloc qui se trouve dans la liste du fait ***présent***, effectuer les opérations suivantes :

présent ([ALPHA BETA GAMMA])

- Charger le fichier ALPHA.blk dans la database.
- Chercher tous les faits ***équivalents*** contenant un type donné.
- Créer le fait ***bloc_port_out*** relativement aux faits ***out*** et le fait ***bloc_port_in*** avec le fait ***in***.

ALPHA.blk	*équivalent(ALPHA , ALPHA1 , B1)*	
out(1 , J1 , α1)		*bloc_port_out (B1 , 1 , J1 , α1)*
BETA.blk	*équivalent (BETA,BETA1,B2)*	
in(1 , J1 , β1)		*bloc_port_in(B2 , 1 , J1 , β1)*
out(1 , J2 , β2)		*bloc_port_out(B2 , 1 , J2 , β2)*
out(2 , J3, β3)		*bloc_port_out(B2 , 2 , J3 , β3)*
GAMMA.blk	*équivalent (GAMMA, GAMMA1, B3)*	*bloc_port_in (B3 , 1 , J1 ,γ1)*
in (1 , J1 ,γ1)	*équivalent(GAMMA, GAMMA2, B4)*	*bloc_port_in (B3 , 2 , J2 ,γ2)*
in (2 , J2 ,γ2)		*bloc_port_in (B4 , 1 , J1 ,γ1)*
		bloc_port_in (B4 , 2 , J2 ,γ2)

fig 4.53 : Création de faits *bloc_port_in* et *out*.

IV.4.2.2. Partie description

La partie description est interprétée à la manière de la partie 'lien' du langage bond-graph. On considère la description blocs et noeuds à la manière d'une structure intermédiaire formée uniquement de lien.

des	***lien(B1,Ne1)***
B1 Ne1 B2 Nf1 B3	***lien(Ne1,B2)***
Nf1 B4	***lien(B2,Nf1)***
	lien(Nf1,B3)
	lien(Nf1,B4)

fig 4.54 : Code intermédaire généré à partir d'une description blocs & noeuds.

Simultanément à cette genération de code, les noeuds présents dans la description sont classés suivant l'ordre de leur rencontre dans la liste L du fait ***ordre_noeud (L)***. Ce fait nous sera utile lors de la phase de sélection. Pour la description précédente, nous avons le fait : ***ordre_noeud ([Ne1 Nf1])***

Une description peut-être syntaxiquement correcte et ne pas traduire exactement le schéma originel. En effet, lorsque la topologie du modèle est composée d'un nombre élevé de blocs et de noeuds, l'utilisateur augmente les erreurs de saisies. Afin de vérifier si la description correspond bien à la structure initiale, il est possible de visualiser celle-ci à l'aide du module graphique

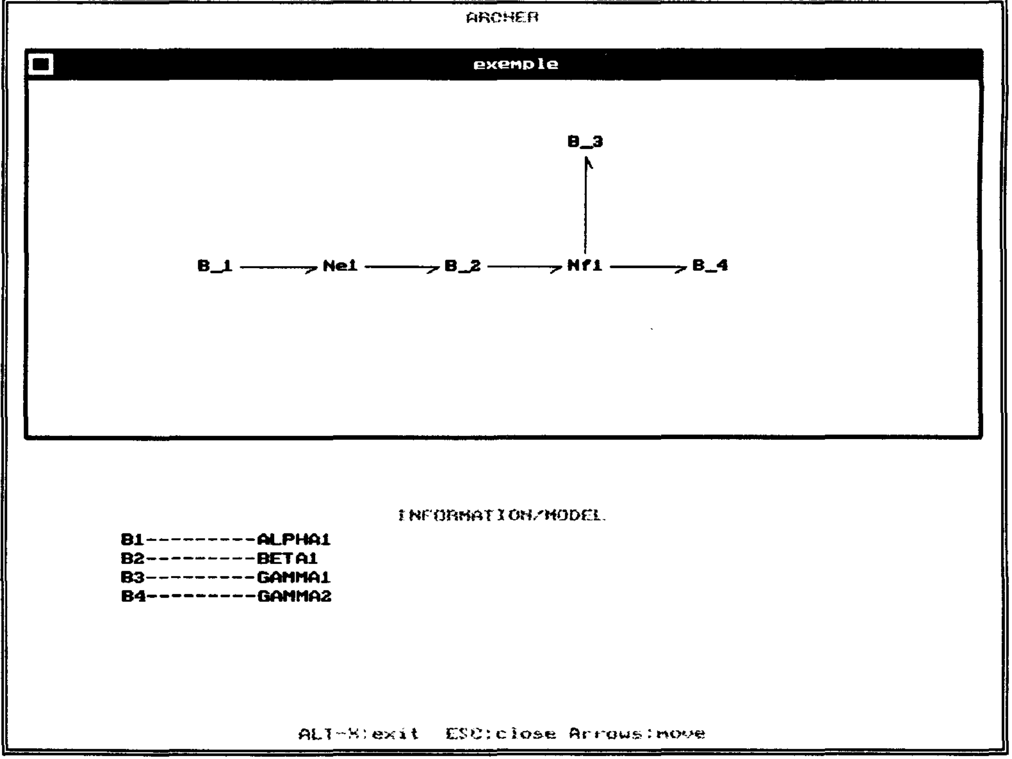


fig 4.55 : Affichage de la struture formée des blocs et des noeuds de la partie description.

IV.4.2.3. La Fusion

La fusion consiste à remplacer le lien entre le bloc $\mathbf{B_n}$ et le noeud **Ni** par un lien qui relie la jonction $\mathbf{J_k}$ du port sélectionné (in_k ou out_k) (concaténée avec le terme alphanumérique '$\mathbf{B_n}$') à ce même noeud.

Condition 1 : Soit un bloc $\mathbf{B_n}$ lié à un noeud de couplage **Ni**, la sélection sera validée si $\mathbf{B_n}$ possède :

- au moins un port en sortie,
- au moins un port en entrée.

Condition 2 : Si $\mathbf{B_n}$ possède un seul port en sortie ou un seul port en entrée, la sélection avec le noeud **Ni** se fait alors automatiquement car il n'y a aucune ambiguïté sur le choix des ports.

Pour chaque noeud contenu dans la liste **L** du fait ***ordre_noeud(L)*** nous effectuons la procédure suivante :

Si le ***lien(Bn , Ni)*** existe

Si le fait ***bloc_port_out(Bn , k , Jk , dk)*** est unique, la sélection est automatique

Effectuer la fusion de la jonction **Jk** du bloc **Bn** avec le noeud **Ni**.

Concaténer 'Jk' avec 'Bn' pour obtenir 'JBn'

Effacer le ***lien(Bn , Ni)*** et ***bloc_port_out(Bn , k , Jk , dk)***

Ajouter le ***lien (JkBn , Ni)***

Sinon proposer à l'utilisateur de choisir un port de sortie sur le bloc **Bn** à l'aide des faits ***bloc_port_out*** relatif au bloc **Bn** :

exemple : ***bloc_port_out(Bn , i , Ji , di)***
bloc_port_out(Bn , k , Jk , dk)

Lorsque ce port est choisi, effacer le fait ***bloc_port_out*** afin d'éviter une resélection de ce port.

exemple : l'utilisateur choisit le port $\mathbf{out_i}$

Faire la fusion

Effacer ***lien(Bn , Ni)*** et ***bloc_port_out(Bn , i , Ji , di)***

Ajouter ***lien(JiBn , Ni)***

Recommencer avec un autre ***lien(Bn , Ni)***

Faire le même traitement avec un ***lien(Ni,Bn)*** et les faits ***bloc_port_in***.

Ainsi, peut-on gérer les erreurs de sélection directement par correction ou par proposition d'autres ports pour un couplage optimal des blocs. Comme nous allons le voir, cette opération de couplage s'effectue en parallèle avec la sélection. En effet, il est préférable de laisser, à la sélection automatique, la charge de proposer les différents ports susceptibles d'être connectés au regard des règles de couplage que nous allons établir.

IV.4.2.4. Règles de couplage

Comme nous l'avons proposé au chapitre II, le couplage des blocs peut amener à une incompatibilité des domaines physiques en présence au niveau d'un noeud **Ni** ou **Nf**.

Nous proposons une mini expertise à travers un dialogue avec l'utilisateur afin de résoudre ce conflit.

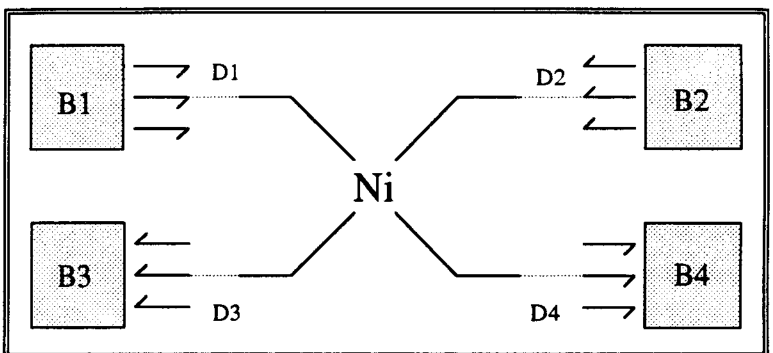


fig 4.56 : Etat du noeud Ni après la sélection des ports.

Pour un noeud **Ni** donné :

<u>Cas1.</u>

si tous les domaines **Di** sont identiques,

alors il n'y a pas de problème de couplage, la fusion se fait entre le noeud **Ni** et les ports des blocs.

<u>Cas 2.</u>

si les domaines **Di** ne sont pas tous équivalents,

et si il existe au moins un domaine **Dc** commun à tous les blocs tels que **Dc** existe :

- parmi les sorties des blocs **B1** et **B2**,
- parmi les entrées de **B3** et **B4,**

alors

Option 1 : L'utilisateur peut changer la sélection pour le noeud Ni de façon à avoir le même domaine (**Dc**) sur tous les ports.

Option 2 : L'utilisateur garde la sélection précédente et ajoute un ou plusieurs blocs intermédiaires (**Bk_i**) pour faire la liaison entre le bloc et le noeud **Ni**. Pour cela on prend un domaine de référence, le premier domaine rencontré lors de la sélection, et on affecte ce domaine au noeud courant **Ni**. Les blocs **Bk** doivent avoir au moins un de leurs ports (en entrée ou en sortie) appartenant à la nature physique du noeud **Ni**.

Remarque : Nous avons donc besoin d'un fait qui garde pour un noeud **Ni** donné, le domaine courant. Ce fait nous l'appelons tout simplement ***domaine_courant(D)***. A chaque nouveau noeud, ce fait est réaffecté du premier domaine rencontré.

Pour avoir l'information sur le domaine **D** courant, il suffit de le lire dans le quatrième paramètre du fait ***bloc_port_in*** ou ***bloc_port_out*** relatif au port sélectionné.

Les autres domaines sélectionnés (**D2**, **D3**, **D4**) sont tous différents, il faut alors placer les blocs **Bk_i** suivants :

- **Bk2** avec au moins 1 port de domaine **D2** en entrée et un port de domaine **D1** en sortie.
- **Bk3** avec **D1** en entrée et **D3** en sortie.
- **Bk4** avec **D1** en entrée et **D4** en sortie.

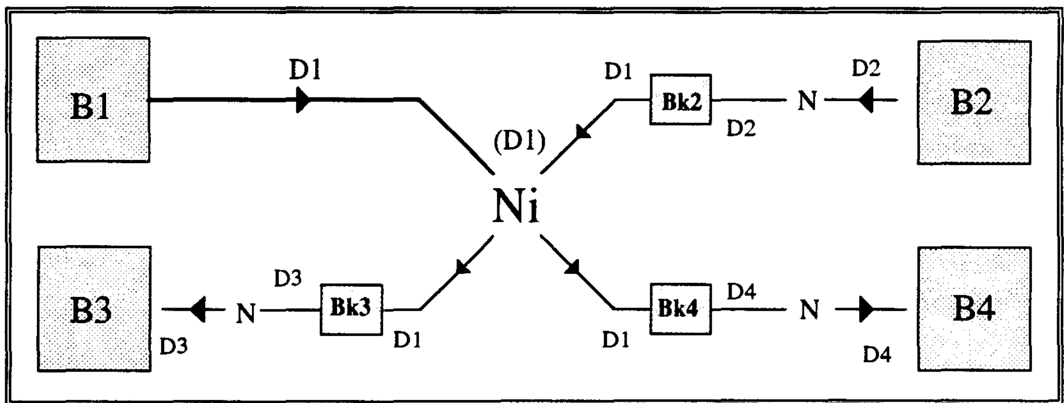


fig 4.57 : Blocs intermédiaires Bk pour le réajustement des domaines physiques.

L'utilisateur peut retourner à l'éditeur de texte pour construire, définir et ajouter ces nouveaux blocs en modifiant la description ainsi que la déclaration.

Option 3 : L'utilisateur peut choisir la proposition d'ARCHER qui consiste à placer entre le bloc et le noeud **Ni** un élément transducteur (**TF** pour transformateur ou **GY** pour gyrateur).

Nous pouvons faire l'analogie avec les blocs **Bk**. En effet, les transducteurs peuvent être considérés comme des blocs bond-graphs constitués soit d'un élément transformateur, soit d'un élément gyrateur, avec un port en entrée et un port en sortie affectés des domaines physiques adéquats.

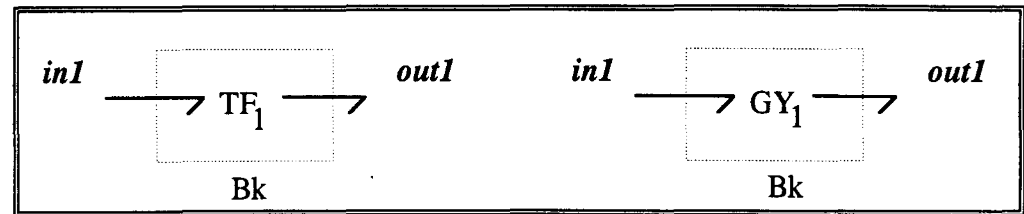


fig 4.58 : Bloc TF et GY

L'utilisateur aurait pu se fabriquer de tels blocs, mais pour des raisons de gain de temps et de convivialité, nous avons donné à l'utilisateur le moyen de résoudre ces conflits de domaines physiques directement lors de la sélection des ports sans retourner à l'éditeur de texte.

L'opération de **fusion** est enrichie par la création d'un fait ***jonction*** **TF** ou **GY** et d'un lien entre le noeud courant et le bloc à traiter selon les cas de la figure suivante :

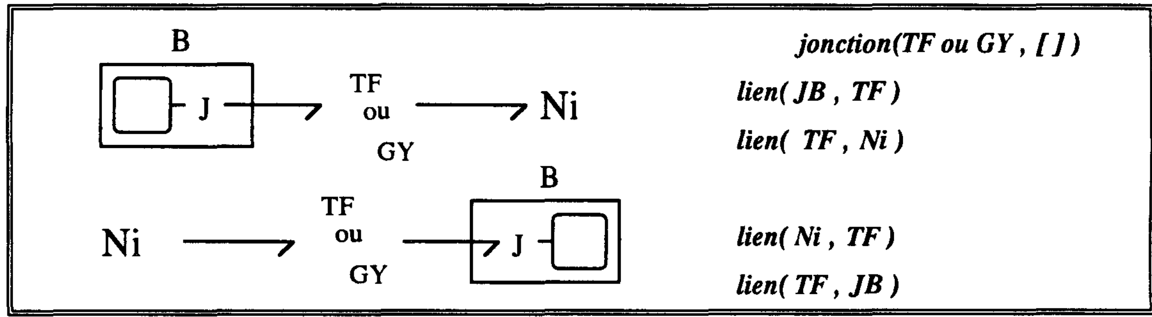


fig 4.59 : Fusion des noeuds selon l'ajout d'un TF ou d'un GY.

Par expérience on peut noter que l'échange de puissance entre certains domaines se fait par l'intermédiaire du même transducteur (voir chapitre II). Nous proposons de constituer une base de données qui fait correspondre aux domaines en présence un transducteur. Pour cela nous définissons un fait ***couplage(champ1, champ2, champ3)*** qui possède la déclaration suivante :

champ1 et **champ2** représente 2 domaines physiques mis en présence,
champ3 caractérise le transducteur **TF** ou **GY** qui sera proposé à l'utilisateur.

Par exemple un moteur est souvent représenté par un gyrateur, celui-ci établit une liaison entre le domaine électrique (e) et mécanique de rotation ou de translation (mr ou mt). De même un vérin se caractérise par un transformateur pour coupler les domaines hydraulique (h) et mécanique.

Nous avons les faits ***couplage(e,mr,GY)***, ***couplage(e,mt,GY)***, ***couplage(h,mt,TF)*** et ***couplage(h,mr,TF)***. Cette base de données est évolutive et peut être complétée par l'utilisateur.

Option 4 : L'utilisateur ne suit pas la proposition d'ARCHER, et il choisit le transducteur en fonction des lois physiques affichées sur l'écran. Celles-ci sont présentées en adaptant les variables effort et flux aux deux domaines conflictuels grâce à deux faits :

loi_tf(domaine1, e1, f1, domaine2, e2, f2)
loi_gy(domaine1, e1, f1, domaine2, e2, f2)

e1/f1 → TF (t) → e2/f2 $\quad\quad$ e1/f1 → GY (g) → e2/f2

$$\begin{cases} e1 = t \cdot e2 \\ f2 = t \cdot f1 \end{cases} \qquad \begin{cases} e1 = g \cdot f2 \\ e2 = g \cdot f1 \end{cases}$$

fig 4.60 : Relations mathématiques du transformateur et du gyrateur.

Remarque : Lorsque nous passerons à la phase de simulations numériques les gains **t** et **g** des transformateurs et des gyrateurs seront initialisés par défaut à la valeur 1. En effet, il s'agit de traduire l'échange de puissance entre deux blocs dans le cas idéal, c'est-à-dire en suivant la loi de conservation de l'énergie (figure 2.128). Si cela est nécessaire, l'utilisateur pourra affecter à ces gains la valeur qu'il voudra.

Nous avons besoin d'un compteur pour les jonctions **TF** et **GY** susceptibles d'être rencontrées au fur-et-à mesure de la sélection : ***compteur(champ1 , champ2)*** :

champ1, représente un élément **TF** ou **GY**
champ2, est l'indice de multiplicité de l'élément du **champ1**.

En **Annexe 4**, nous donnons une session complète de notre exemple.

IV.4.3. Génération d'un code bond-graph unique

A la fin de la sélection, nous obtenons un code intermédiaire qui permet la génération du code bond-graph.

Ce code intermédiaire agit à la manière d'un ciment qui va relier les codes bond-graphs de chaque bloc (*.bgs) préalablement chargé dans la mémoire courante après la renumérotation des jonctions et des éléments afin de constituer le fichier de descendance.

Le code final est obtenu après la phase habituelle de simplification qui a en charge l'élimination des jonctions redondantes.

Une fois la sélection des ports réalisée, nous possédons tous les éléments nécessaires à la construction d'un bond-graph unique, à partir des blocs bond-graphs. Le principe de génération réside dans une opération que nous appelons le mixage des différents codes.

IV.4.3.1. La Renumérotation

Au paragraphe II.2.3.3 du chapitre II nous avons proposé de numéroter tous les éléments d'un bloc afin de garder le caractère unique des indices associés et d'éviter toute ambiguité lors du calcul formel et la simulation numérique. Le traitement est le suivant.

Pour chaque fait ***équivalent(NOM , NAME , Bn)***

Charger en mémoire le fichier 'NOM.bgs'

Toutes les jonctions 'J' sont remplacées par la concaténation de 'J' avec le caractère 'Bn'.

jonction(J, [L]) ⇒ jonction(JBn , [L])

Initialiser le fait compteur de chaque élément **E** ∈ { R, C, I, Se, Sf, } à 1 :

compteur (E,1)

Pour chaque élément $\mathbf{E_i}$ de la liste **L**
Identifier l'élément **E**
Chercher le fait ***compteur(Elément , N)***
Effacer $\mathbf{E_i}$ de la liste **L** et ajouter le nouvelle élément E_{N+1}
Création du fichier *. hlp dans lequel nous écrivons à l'aide d'un traitement

' E_{N+1}' est l'élément 'E_i' du bloc 'B_n' ou 'NAME'

La numérotation sur notre exemple est présentée à la fin de l'**Annexe 4**.

IV.4.3.2. Mixage des différents codes

A partir de la connaissance de la description et des ports sélectionnés, nous avons tous les paramètres nécessaires à la fusion des blocs, autrement dit le mélange des codes renumérotés de chaque bloc avec la "structure intermédiaire".

C'est la phase qui mélange les codes jonctions et liens de chaque bloc ré-indicé, et le code intermédiaire construit à partir de la description bloc et noeud et la sélection des ports.

⇒ Pour chaque fait ***lien*** du code intermédiaire :
les noeuds $\mathbf{Ne_n}$ (resp $\mathbf{Nf_n}$) sont remplacés par des jonctions $\mathbf{0_n}$ (resp $\mathbf{1_n}$) avec création d'un fait ***jonction(0_n , [ ])*** (resp ***jonction(1_n , [ ])***).

⇒ La **simplification** intervient après la fusion, et se déroule en deux parties :

- La **première** fait intervenir les simplifications usuelles du bond-graph introduites dans le tableau de la figure 2.75.

- La **seconde** consiste à éliminer les jonctions **TF** ou **GY** ayant un de leur port en ***in*** ou en ***out*** non sélectionné par l'utilisateur. Lors de la sélection, il est possible d'avoir pour certains blocs le cas où tous les ports ne sont pas connectés.

On suppose que le choix de l'utilisateur est correct et qu'aucune erreur ne subsiste, il y a donc élimination de ces jonctions qui n'ont plus d'utilité dans le bond-graph final.

⇒ La **numérotation** des jonctions { 0, 1, TF, GY }

Les jonctions **0** et **1** sont réindicées suivant leur ordre d'apparition dans la mémoire courante. Les jonctions **TF** et **GY** appartenant aux blocs sont réindicées à partir des numéros contenus dans les faits ***compteur(TF, NT) et compteur(GY, NG)***.

Ces nouvelles informations sont ajoutées dans le fichier de descendance *.hlp correspondant.

En **Annexe 4**, nous donnons le résultat du mixage des codes sur l'exemple proposé.

IV.4.4. Aspects évolutifs du langage

IV.4.4.1. Le pseudo-bloc

Au chapitre II nous avons proposé de mélanger le langage blocs & noeuds avec le langage série et parallèle. Nous n'avons pas voulu réécrire un autre langage, mais utiliser les langages existants pour résoudre ce problème. Les éléments électriques, mécaniques ou hydrauliques dans un réseau blocs et noeuds sont considérés comme des blocs, d'où le nom de pseudo-bloc. Par exemple la description de la figure 4.61 sera traitée de la façon suivante.

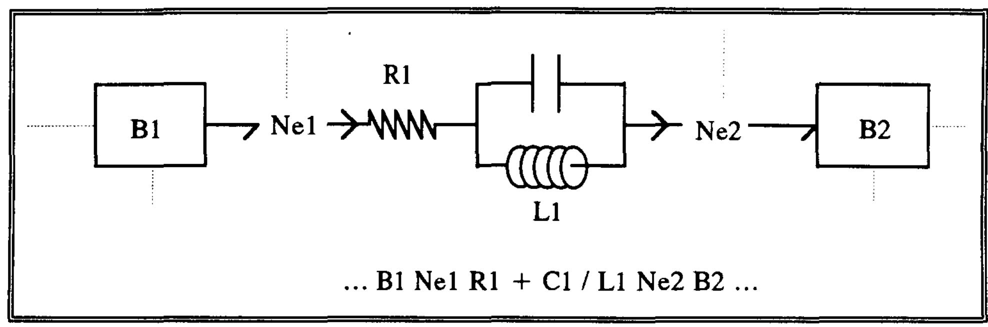


fig 4.61 : Exemple d'une description avec une partie écrite en langage série et parallèle

1. La méthode consiste à isoler lors de l'analyse syntaxique, les parties écrites en langage série et parallèle qui sont toujours placées entre deux noeuds.

'Ne1 R1+C1/L1 Ne2'

'R1+C1/L1'

2. Ensuite d'identifier le domaine physique des éléments par une analyse lexicale et le dictionnaire des éléments physiques.

domaine électrique

3. A partir de cette partie, nous avons vu qu'il était possible de fabriquer des blocs en langage série et parallèle.[*] Pour cela un progamme fabrique le fichier texte ci contre en concaténant, à la partie, un point **e1** pour l'entrée et **s1** pour la sortie. Puis le module, qui a en charge la création d'un bloc en langage série et parallèle, est lancé automatiquement.

'électrique'
'e1+R1+C1/L1+s1'
'port'
'e1=in1'
's1=out1'

---

[*] Voir chapitre II et le paragraphe IV.3.2.

Nous possédons un bloc bond-graph **PB1** avec ses 2 fichiers associés : **'PB1.bgs'** qui contient le bond-graph et **'PB1.blk'** qui contient l'information sur les ports.

On suit les étapes classiques d'une description blocs et noeuds avec :

'B1 Ne1 PB1 Ne2 B2'

à la différence prés que pour les pseudo-blocs, la sélection est automatique car ils possèdent par définition un seul port en entrée et en sortie. Ce traitement est transparent pour l'utilisateur.

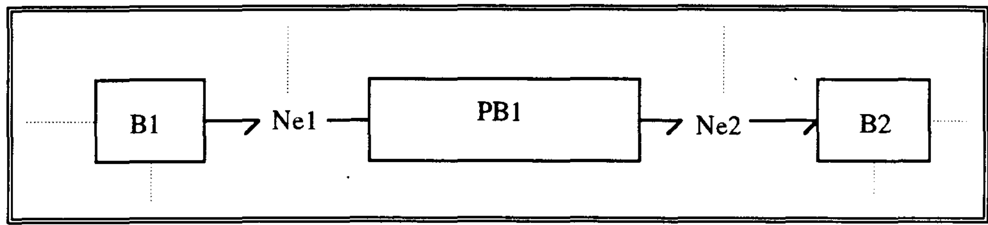


fig 4.62 : Transformation de la partie en pseudo-bloc.

IV.4.4.2. Bloc en langage blocs & noeuds

La principale force de ce langage est la possibilité de créer un bloc à partir d'une description blocs & noeuds. Pour cela on assigne aux noeuds des ports en entrées et en sorties avec leur nature physique.

C'est le principe d'imbrication que nous avons déjà signalé au chapitre II. Nous mettons en évidence l'aspect récursif de la création d'un bloc à savoir l'ajout d'une partie 'port' après les parties 'dec' et 'des' du modèle blocs et noeuds.

La syntaxe de cette partie port est analogue à celle du langage bloc bond-graph avec la différence que le métanom **<jonction>** est remplacé par **<noeud>**.
La description des ports de la figure se fait de la façon suivante :

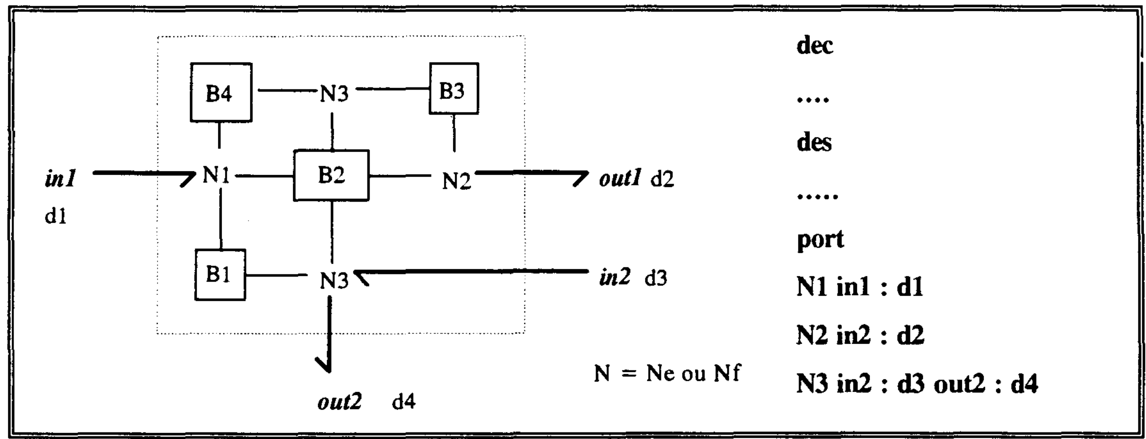


fig 4.63 : Bloc formé d'une description blocs & noeuds.

IV.4.5. Modélisation d'une machine outil

Pour montrer les possibilités et le fonctionnement du langage, nous allons illustrer le langage à travers un exemple assez complet pris dans le livre de KARNOPP et ROSENBERG [75].

La machine outil de la figure 4.64 fait intervenir les domaines électrique, hydraulique et mécanique et la topologie du système permet de dégager trois parties distinctes en relation avec ces mêmes domaines.

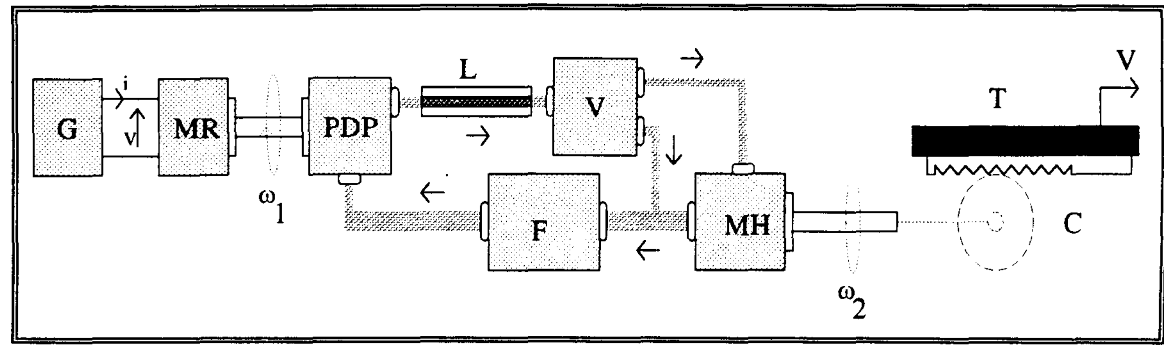


fig 4.64 : Modèle d'une machine outil.

- Dans la **partie 1**, le moteur rotatif (**MR**), alimenté par une tension (**u**) et un courant (i), fournit une vitesse de rotation ω_1 pour le fonctionnement d'une pompe à déplacement **positif (PDP)** du circuit hydraulique de la **partie 2**.

- Le moteur hydraulique (**MH**) a une valve **V** en amont pour contrôler la puissance, dans la **partie 3**, délivrée au pignon de la crémaillère (**C**), qui par une vitesse de rotation ω_2 conduit la table (**T**) de masse M se déplaçant à une vitesse de translation (**V**).

- Le fluide résultant du moteur hydraulique retourne à la pompe à travers un filtre (**F**).

Remarque : Pour symboliser les pertes de pression **P** et de débit **Q** dans les conduits du circuit hydraulique, nous avons introduit l'élément ligne (**L**).

A partir de cette description fonctionnelle, il suffit de suivre les différentes étapes que nous avons décrites au chapitre II (analyse fonctionnelle)

Etape 1 :

On met en évidence les sous-systèmes en soulignant les "variables pivots" qui font le lien entre les parties n'appartenant pas au même domaine de la physique. Par exemple, l'échange de puissance entre le moteur rotatif (**MR**) et la pompe hydraulique boîte (**PDP**) se fait par la même vitesse de rotation ω_1, elle est représentée par un noeud de flux (**Nf**) associé à la variable mise en jeu. La figure 4.66 est une représentation symbolique par blocs et noeuds de l'exemple de la figure 4.65.

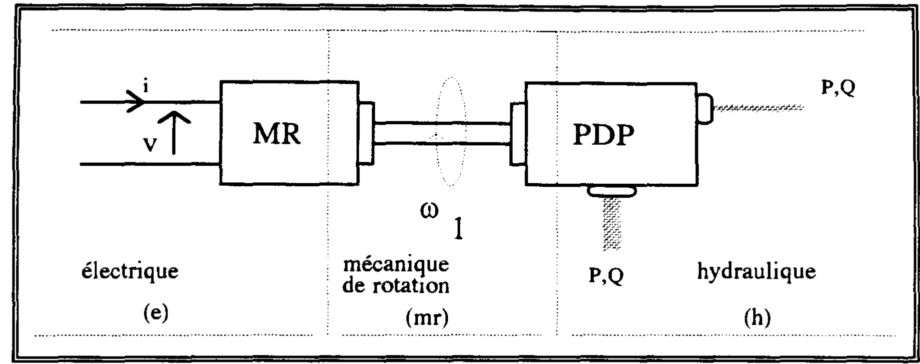


fig 4.65 : Variable vitesse de rotation commune ω1 à chaque bloc.

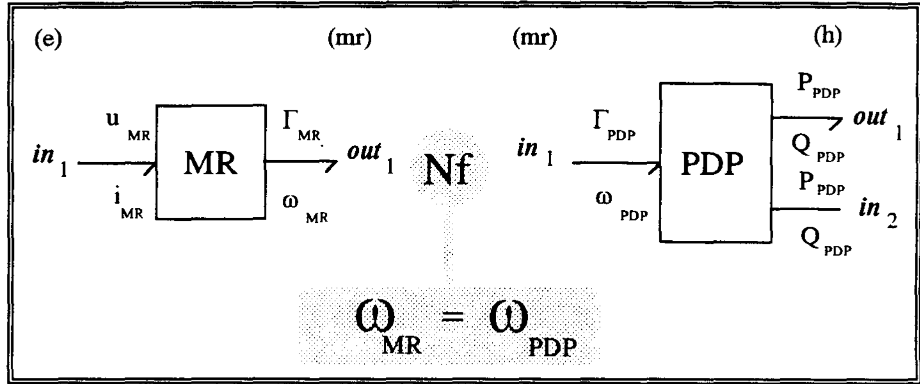


fig 4.66 : Couplage des blocs MR et PDP par un noeud de flux Nf.

A présent, nous pouvons définir, pour chaque sous-partie du système, une représentation bloc à une ou plusieurs entrées et sorties (ou ports) en spécifiant les domaines physiques employés à travers ces ports.

Le type de modélisation pris pour définir chacun de ces blocs est la modélisation par bond-graph. La figure 4.67 schématise les blocs définis plus haut avec la nature physique des ports.

Etape 2 :

Après avoir défini tous les blocs, on décrit le modèle lui-même à l'aide du langage blocs & noeuds.

Pour cela nous dessinons une représentation bloc (figure 4.68) suivant la topologie du modèle en plaçant entre les blocs des noeuds d'efforts (**Ne**) ou de flux (**Nf**) suivant la variable mise à contribution lors de l'échange de puissance.

Ensuite, à l'aide de l'éditeur d'ARCHER, nous entrons une description texte du schéma en déclarant les blocs utilisés dans la partie "dec" et en décrivant, dans la partie 'des', la structure formée par les blocs (Bn) et les noeuds (figure 4.69). Nous pouvons visualiser la structure graphique de la descriptions blocs & noeuds (figure 4.70).

MODELE PHYSIQUE	REPRESENTATION BLOC	BOND-GRAPH
G i v	G → out1 (e)	Se 1 → 1_1 → out1
i v MR Γ ω	in1 → MR → out1 (e) (mr)	in1 → GY_1 → out1
Γ ω PDP → P , Q ← P , Q	in1 → PDP → out1 (h) (mr) ← in2 (h)	out1 in1 → TF_1 → 1_1 in2
P , Q → L → P , Q	in1 → L → out1 (h) (h)	in1 → 0_1 → 1_1 → out1 C1 R1
P Q → V → P , Q → P , Q	in1 → V → out1 (h) (h) → out2 (h)	R2 in1 → 0_1 → 1_2 → out1 1_1 → out2 R1
P , Q ↓ Γ MH P , Q ← ω	in1 → MH → out1 (h) (h) → out2 (mr)	in1 → 1_1 → out1 TF_1 → out2
P , Q ← F ← P , Q	in1 → F → out1 (h) (h)	R1 in1 → 1_1 → out1
Γ → F , V C ω	in1 → C → out1 (mr) (mt)	in1 → TF_1 → out1
→ V Masse + Frottements	in1 → T (mt)	R : R1 in1 → 1_1 I : I1

fig 4.67 : Blocs bond-graph pré-définis pour le modèle de la machine à outil. (*)

(*) Chaque bloc est décrit par un code bond-graph à l'aide du langage 'bloc bond-graph', comme le montre le tableau de l'**Annexe 5**.

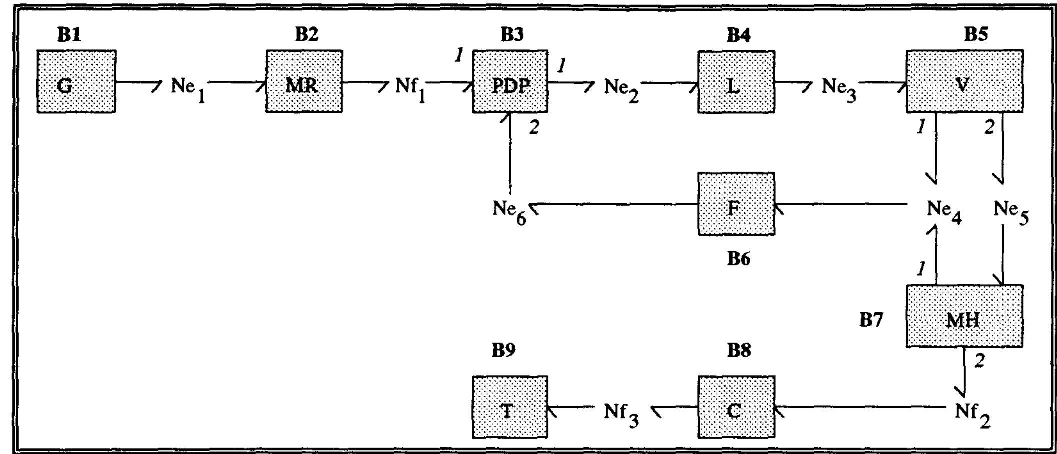


fig 4.68 : Représentation bloc du modèle de la figure.

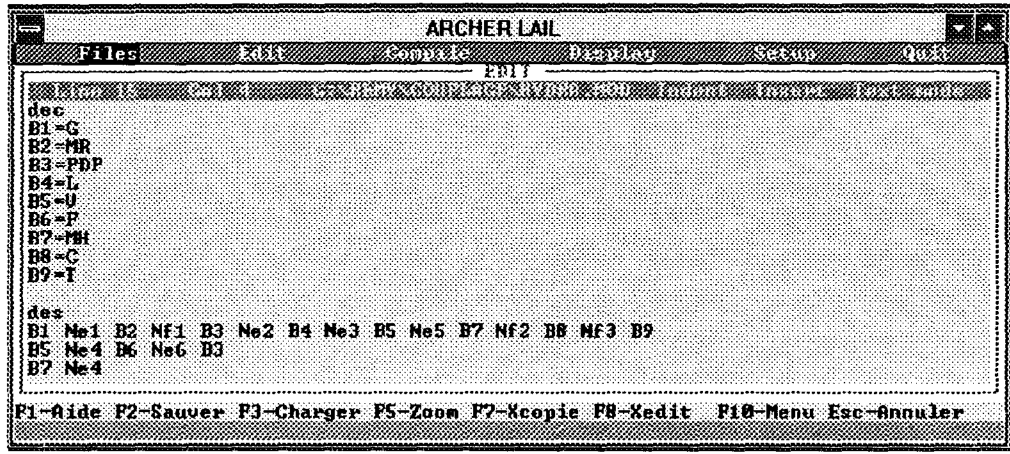


fig 4.69 : Description texte du modèle de la figure 4.68.

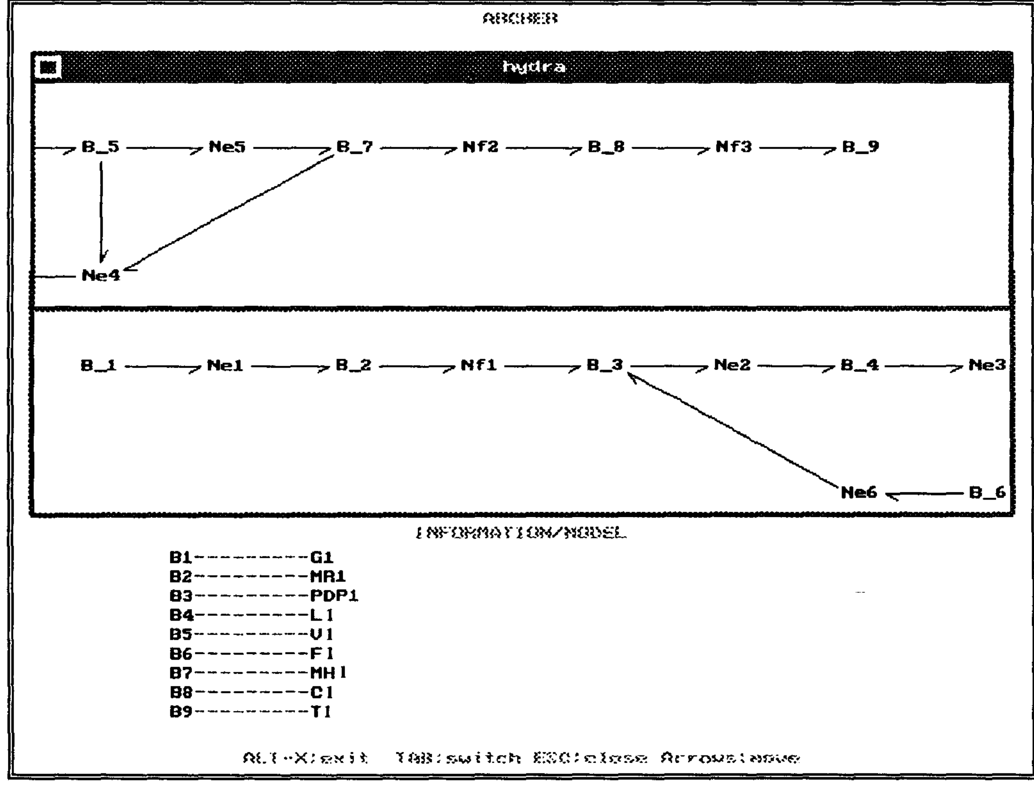
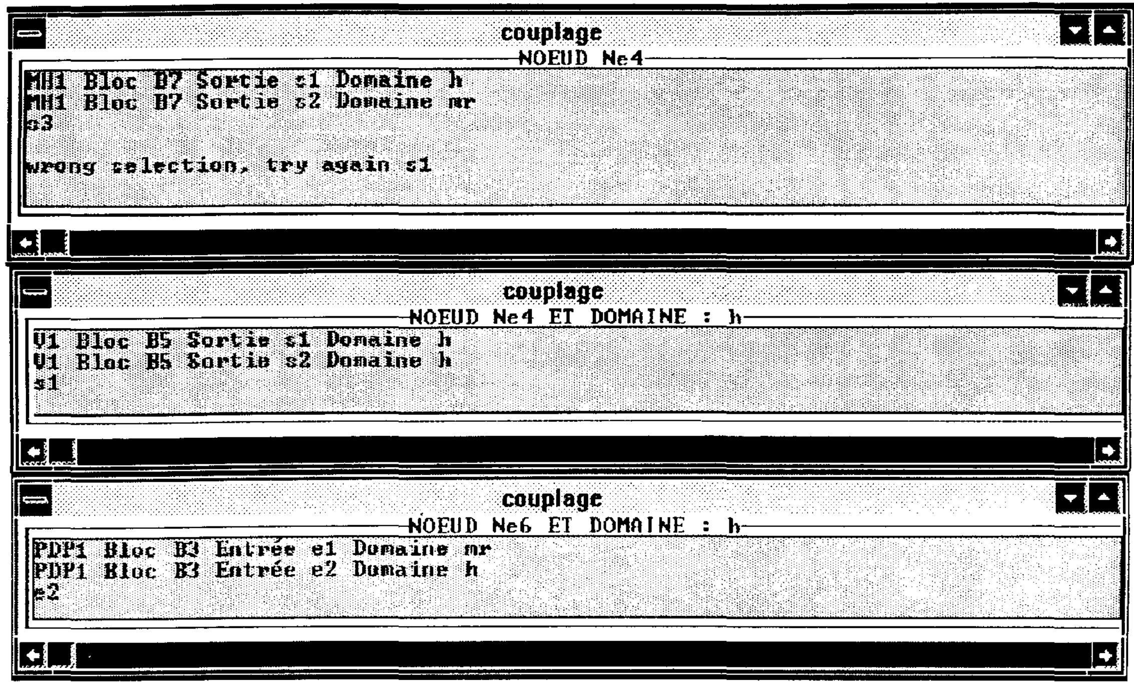


fig 4.70 : Dessin de la description blocs & noeuds



Les autres ports sont sélectionnés automatiquement, de plus aucun conflit physique ne se présente. Le couplage, la numérotation, le mixage et la simplification des codes donnent le bond-graph suivant :

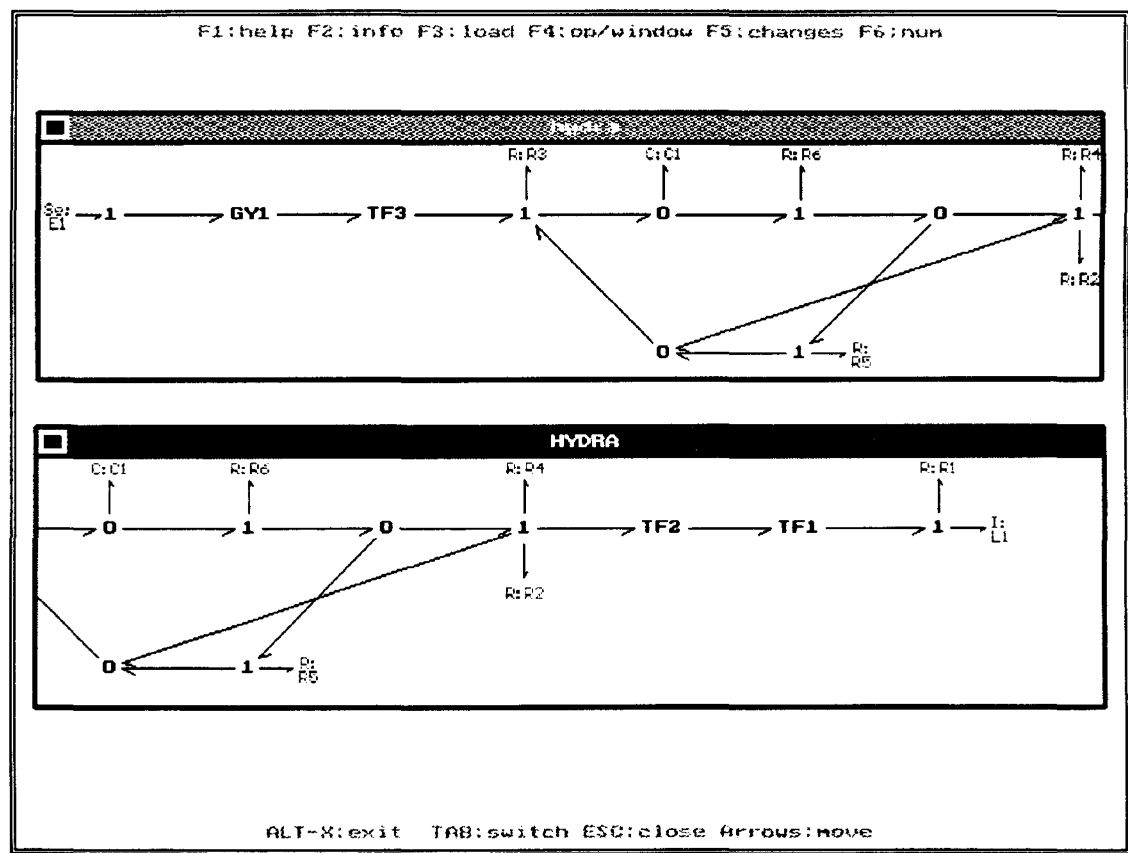


fig 4.71 : Bond-graph acausal simplifié du modèle blocs & noeuds.

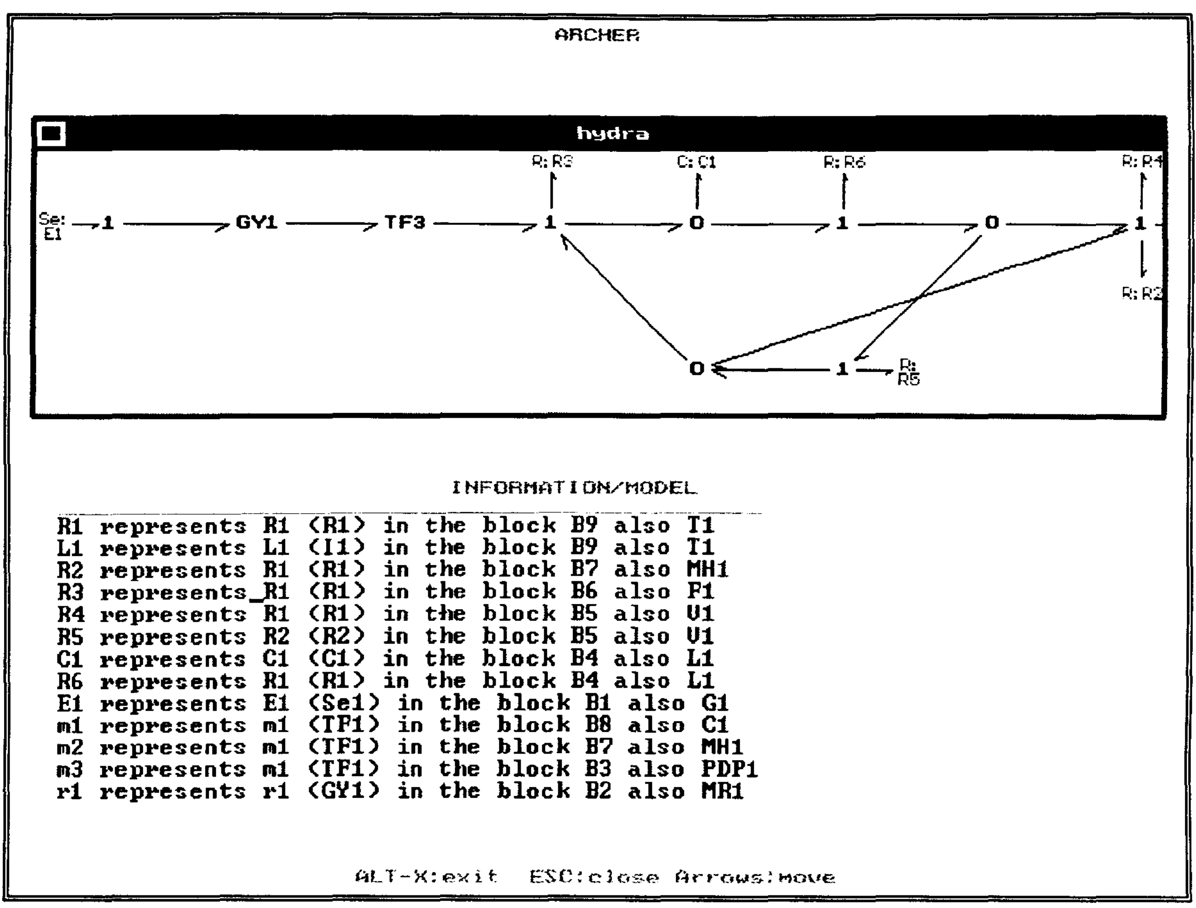


fig 4.72 : Bond-graph final de la machine outil avec informations sur les éléments (*.hlp).

IV.5. Bond-graph et bloc mathématique.

IV.5.1. Position du problème

Dans un système physique, les sous-systèmes peuvent ne pas être tous modélisés par des blocs bond-graph (*). Nous considérons ici le cas de modèles de type équation d'état (nous nous limitons au cas linéaire). Dans ce cas, les informations apparaissent sous la forme d'un signal unique et non de deux signaux complémentairs comme dans le cas des bond-graphs.

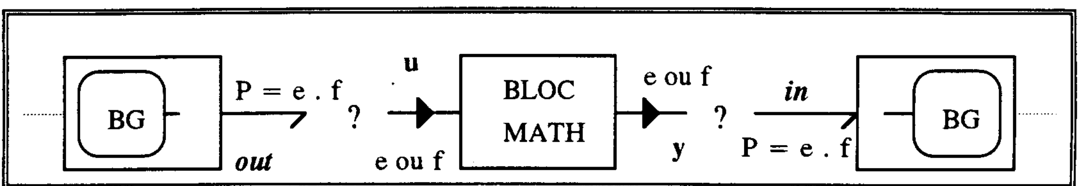


fig 4.73 : Couplage bloc mathématique et bond-graph

Puisque la nature des ports est différente (signaux et puissance), ces blocs mathématiques ne peuvent être couplés directement aux blocs bond-graph par un noeud d'effort ou de flux, comme nous l'avons présenté précédemment.

La méthode à suivre pour résoudre ce problème dépendra du but poursuivi par l'utilisateur.

1. Si le but est la simulation numérique, la solution pour une telle association bond-graphs + équations mathématiques, est de ramener les deux représentations à une forme mathématique directement exploitable par un logiciel de simulation. Cela revient à coupler directement les modèles au niveau numérique.

2. La solution inverse de la première consiste à transformer le bloc mathématique en un bond-graph.

Les travaux proposés par KAMEL [93] permettent de construire un modèle bond-graph à partir d'une matrice de transfert. Le bond-graph ainsi obtenu n'a pas de signification physique, mais possède le même comportement entrées-sorties que la matrice de transfert initiale.

La construction d'un modèle BG associée à une représentation d'état ne peut pas être systématique, à cause de la forme bien particulière des matrices d'états associées à un modèle BG.

(*) Voir chapitre II : "Décomposition d'un système en sous-parties".

En effet, chaque terme des matrices A, B, C, D de l'équation générale

$$\dot{X} = AX + BU$$
$$Y = CX + DU$$

s'interprète à partir des gains des chemins causaux entre les éléments dynamiques (I, C) associés aux variables d'état, les éléments résistifs R, les sources et les détecteurs ; ceci entraîne par exemple une symétrie des termes nuls par rapport à la diagonale principale de A.

3. Lorsque le but de l'étude du modèle est l'analyse structurelle, nous proposons d'utiliser le **digraphe** qui fait correspondre, aux matrices A, B, C et D de l'équation d'état, un graphe G(A,B,C). [REINSCHKE 88].(*)

La notion de "Sources contrôlées" et de "détecteurs" fera le lien entre les signaux d'entrées **U** et de sortie **Y** des modèles mathématiques et les sources de puissance **Se** et **Sf** de la modélisation BG. La figure 4.74 présente le schéma de principe de ce couplage.

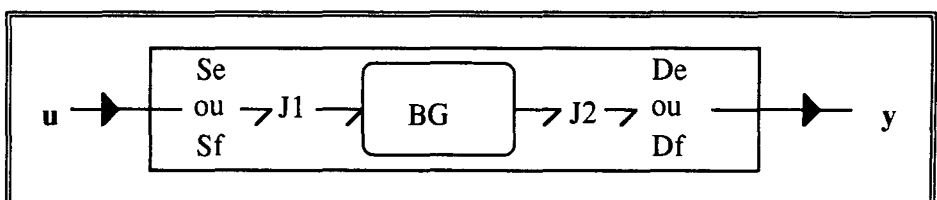


fig 4.74 : Bloc mathématique et équivalence bond-graph.

Nous allons montrer dans ce paragraphe comment ARCHER, à cause de la forme choisie pour les données (quelle que soit leur type) et les méthodologies mises en place, peut s'adapter à cette forme de graphes.

IV.5.2. Couplage Digraphe

Pour analyser structurellement un système formé de blocs mathématiques et de blocs bond-graphs, nous proposons de représenter les équations mathématiques des blocs mathématiques par un digraphe, puis de coupler ces "blocs digraphes" aux blocs bond-graphs. Nous rappelons la définition et la construction d'un digraphe à partir d'une équation d'état, une fonction de transfert, et un bond-graph.

(*) Nous donnons plus loin la définition d'un digraphe.

IV.5.2.1. Digraphe

A. A partir d'une équation d'état

Soit l'équation d'état :

$$\dot{X} = AX + BU$$
$$Y = CX + DU$$

Pour une matrice carrée $A=\{a_{ij} ; i, j \in (1,...,n)\}$ d'ordre n donnée, il y a une correspondance entre la matrice A et le digraphe G(A) qui possède n sommets $z_i \in Z$ et un arc dirigé (z_i, z_j) du sommet initial zi vers le sommet final z_j si l'élément a_{ij} est non nul $(i, j \in (1,...,n))$. Le poids de l'arc est donné par la valeur aij. La représentation des matrices B, C, et D se fait suivant le même principe.

Nous allons prendre l'exemple d'une équation d'état avec une matrice A d'ordre 2, une entrée U1 et une sortie Y1.

$$A = \begin{pmatrix} a11 & a12 \\ a21 & a22 \end{pmatrix} \quad B = \begin{pmatrix} b1 \\ b2 \end{pmatrix} \quad C = \begin{pmatrix} c1 & c2 \end{pmatrix} \quad D = \begin{pmatrix} d1 \end{pmatrix}$$

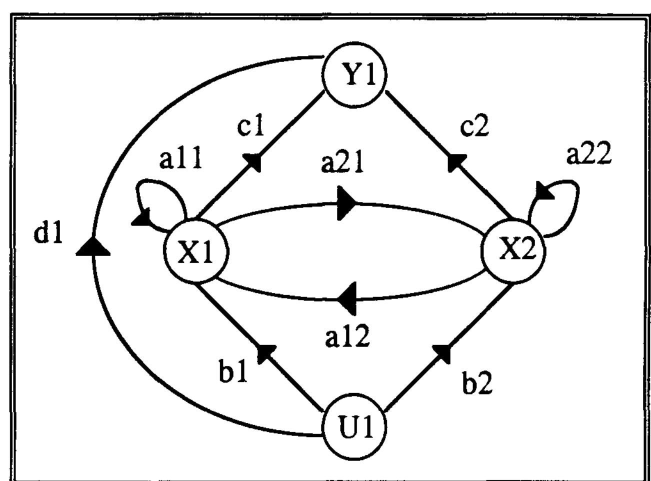


fig 4.75. Digraphe de l'équation d'état.

B. A partir d'une fonction de transfert

Nous pouvons construire indirectement le digraphe d'une fonction de transfert en calculant une équation d'état associée.

Par exemple avec la fonction de transfert suivante, nous pouvons obtenir une représentation d'état sous la forme compagne.

$$\frac{Y1}{U1} = \frac{b_0}{s^2 + a_1 s + a_0} \qquad A = \begin{pmatrix} 0 & 1 \\ -a0 & -a1 \end{pmatrix} \quad B = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$C = \begin{pmatrix} b0 & 0 \end{pmatrix}$$

Le digraphe construit à partir des matrices A, B et C est le suivant :

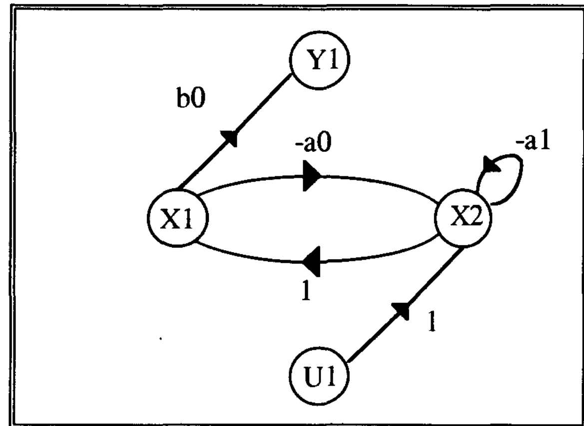


fig 4.76 : Digraphe de la fonction de transfert.

Il faut remarquer qu'il existe de nombreux digraphes différents pour une même fonction de transfert.

C. A partir d'un bond-graph

A tout modèle BG correspondent des modèles mathématiques de type fonctions (ou matrices) de transfert et équations d'état.

Il est donc évident de passer d'un modèle BG à un digraphe, les sommets du digraphes représentent les éléments dynamiques (I et C) associés aux variables d'états, et les noeuds d'entrées et de sorties sont associés aux sources et aux détecteurs respectivement.

IV.5.2.2. Codage et langage d'un digraphe

Pour représenter un digraphe en mémoire nous pouvons employer plusieurs formes de données. Au chapitre III, nous avons présenté le fait ***arc(S1,S2)*** qui caractérise l'arc entre deux sommets **S1** et **S2**, le fait ***transition(Etat1,Transition,Etat2)*** qui décrit un automate à états finis.

A. Le Codage

Le digraphe est une combinaison des 2 représentations et nous proposons de le représenter par un fait ***arc_dg*** avec trois paramètres.

arc_dg(champ1,champ2,champ3)

Le **champ1** et le **champ2** représentent un sommet (X_i, Y_i, U_i) ou (C_i, I_i, De_i, Df_i, Se_i, Sf_i).
Le **champ3** est le poids de l'arc orienté entre les deux sommets **champ1** et **champ2**.

Remarque : La présentation détaillée de l'automate à états finis, faite au chapitre III peut servir comme premier point de départ à un module qui aurait la charge d'analyser le digraphe.

Le code ***arc_dg*** proposé va permettre de manipuler le digraphe afin de calculer des cycles, des chemins..., avec les outils propres au parcours des graphes. Sur le rappel de Prolog, nous avons mentionné quelques exemples d'outils graphiques utilisés avec succès dans ARCHER, notamment pour le parcours des boucles causales.

B. Le Langage

Nous pouvons imaginer un langage déclaratif qui puisse construire automatiquement ce code en commençant par un mot clef 'digraphe' avec une syntaxe similaire au langage blocs et noeuds.

La génération du fait ***arc_dg*** se fait par la description du réseau formé par les sommets et le poids des arcs orientés. La contrainte à respecter est d'avoir un minimum de deux sommets.

```
'digraphe'
'sommet' 'sommmet' 'sommet' 'sommet' ...
```

La méthode employée pour la génération du code ***arc_dg*** est la même que celle déjà employée pour la génération du code ***lien*** dans la partie 'lien' du langage bond-graph.

On prend les lexèmes deux par deux. Le premier et le deuxième sont les sommets **S1** et **S2**, le poids **P** de l'arc entre **S1** et **S2** est donné par l'analyse suivante :

Si **S1** = X_i et **S2** = X_j alors **P** = a_{ji} Si **S1** = U_i et **S2** = X_j alors **P** = b_{ji}

Si **S1** = X_i et **S2** = Y_j alors **P** = c_{ji} Si **S1** = U_i et **S2** = Y_j alors **P** = d_{ji}

Ainsi nous générons le fait ***arc_dg(S1 , S2 , P)***.Le code digraphe est sauvegardé dans un fichier *.dg

Seuls les arcs de poids non nuls sont introduits. Les valeurs numériques des arcs :
a_{ij} (i , j = 1..n) ; b_{ij} (i = 1..n, j = 1..m) et c_{ij} (i = 1..p, j = 1..n) où
n = dim X, m = dim U et p = dim Y,
seront introduits, si nécessaire, dans la phase de simulation numérique.

Exemple : Le digraphe de la fonction de transfert de la figure est décrite de la manière suivante avec les codes correspondants.

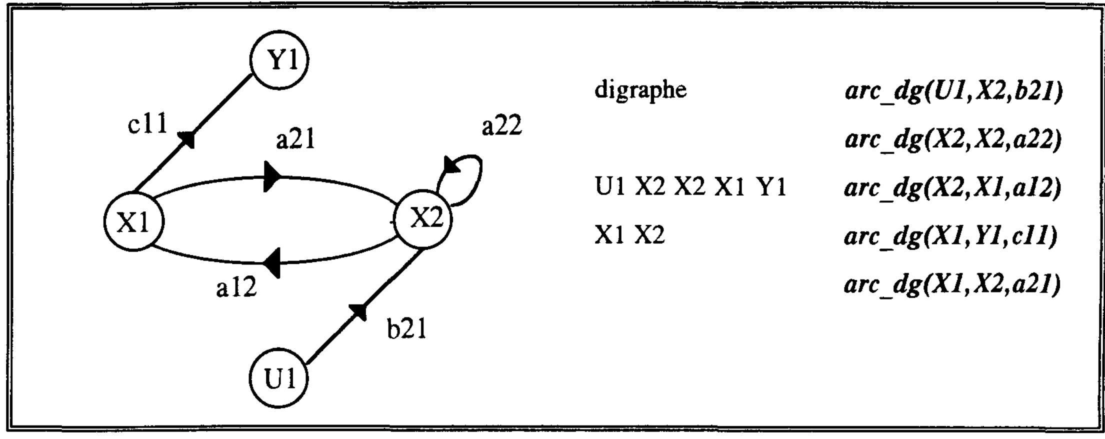


fig 4.77 : Description du digraphe de la figure 4.76.

IV.5.2.3. Bloc digraphe

Un bloc digraphe se caractérise à la manière d'un bloc bond-graph. Nous avons le corps du digraphe (fichier *.dg) et les ports associés à ce bloc avec les faits ***b***, ***in*** et ***out*** qui sont sauvegardés dans un fichier *.blk.

Remarque : Par définition nous les considérons toujours comme des ports, mais cette fois, ils véhiculent non plus une puissance mais un signal (effort ou flux).

Les ports ***in*** sont obligatoirement liés à une entrée (U_i) et les ports ***out*** à une sortie (Y_i).

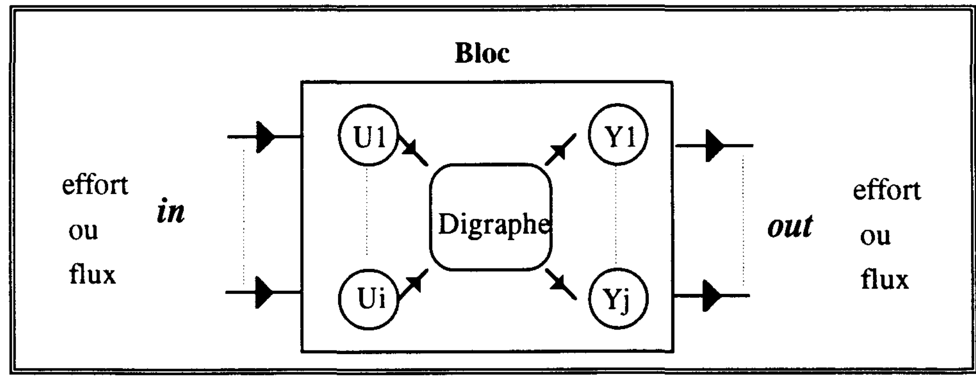


fig 4.78 : Bloc digraphe.

Les faits se composent cette fois-ci de la manière suivante :(*)

b('dg', 'Nom du bloc') : '**dg**' veut dire que le bloc est un digraphe.

in(champ1,champ2,champ3) et ***out(champ1,champ2,champ3)***

champ1 : représente le numéro de l'entrée (***in***) ou de la sortie (***out***).
champ2 : caractérise le paramètre (U_i ou Y_i) que l'utilisateur veut associer au port,
champ3 : indique si le port est associé à un signal de nature 'effort' ou un 'flux'.

Pour affecter les entrées et les sorties qui composent un bloc, nous ajoutons une partie 'port' comme pour les autres langages avec la syntaxe suivante :

```
'port'
'entrée ou sortie' = 'port' : 'nature du signal'
```

Par exemple le bloc '**ALPHA**' de la figure est traduit dans le fichier '**ALPHA.blk**' de la manière suivante :

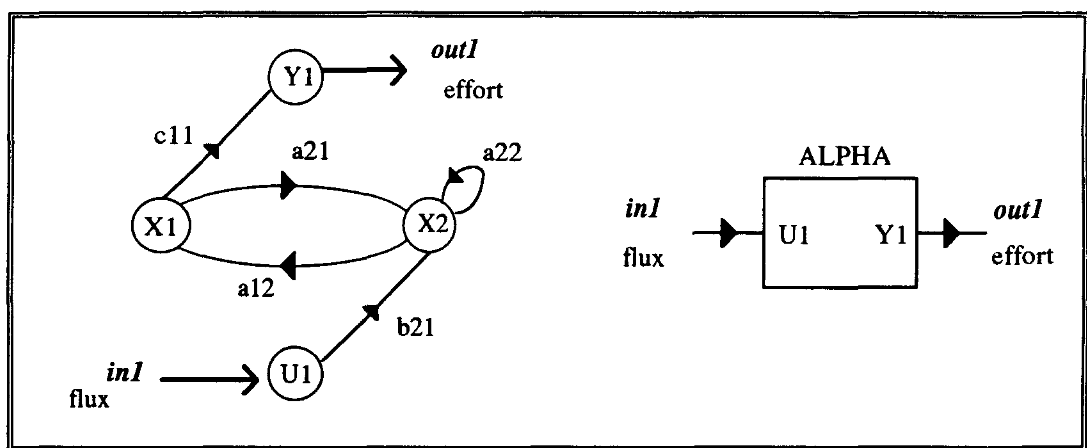


fig 4.79 : Bloc digraph avec 1 entrée flux et une sortie effort.

digraphe	**Fichier *.blk**
...	
port	*b(dg , ALPHA)*
U1 = in1 : flux	*in(1 , U1 , flux)*
Y1 = out1 : effort	*out(1 , Y1 , effort)*

fig 4.80 : Description et codage du bloc digraphe ALPHA.

(*) Voir le paragraphe IV.3.

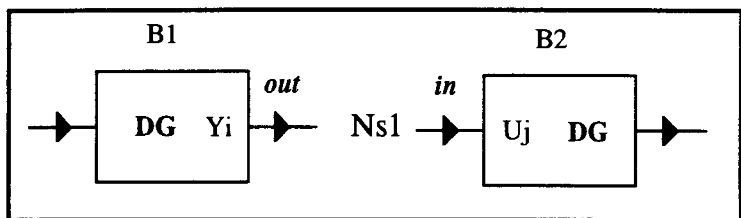
IV.5.2.4. Couplage digraphe et bond-graph

Nous utilisons le langage blocs & noeuds augmenté d'un nouveau noeud pour coupler un bloc bond-graph (BG) avec un bloc digraphe (DG).

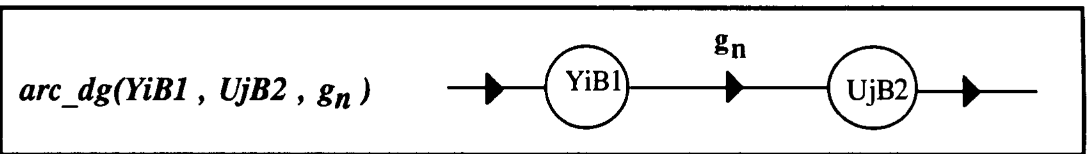
Noeud signal Ns :

Le noeud **Ns** est utilisé pour associer des signaux d'effort ou de flux entre 2 blocs. Lorsque le noeud **Ns** est rencontré dans une description nous créons une liaison, entre les ports ***in*** et ***out*** des blocs en présence, en créant un fait ***arc_dg*** adapté à chaque combinaison.

A. Couplage digraphe + digraphe

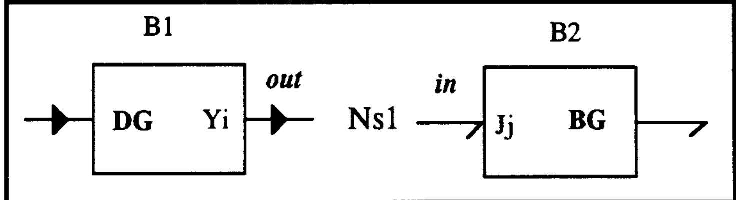


Pour coupler 2 blocs digraphes **B1** et **B2**, nous créons un ***arc_dg*** de poids g_n entre le sommet Y_i de **B1** et le sommet U_j de **B2** du bloc indépendamment de la nature (effort ou flux) des ports. g_n est non nul et peut être égal à 1.



remarque : l'indice **n** de **g** correspond au nombre de nouveaux faits ***arc_dg*** déjà créés.

B. Couplage digraphe + bond-graph



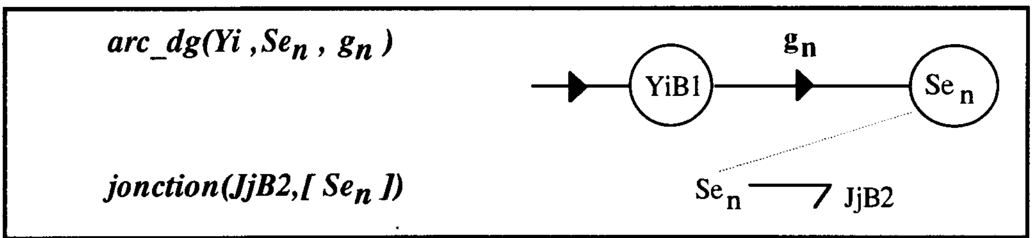
Principe :

Pour coupler un bloc digraphe et un bloc bond-graph, nous créons une source contrôlée **S** (**Se** ou **Sf**) de gain **g** entre la sortie ***out*** du bloc digraphe et l'entrée ***in*** du bloc bond-graph **B2**.

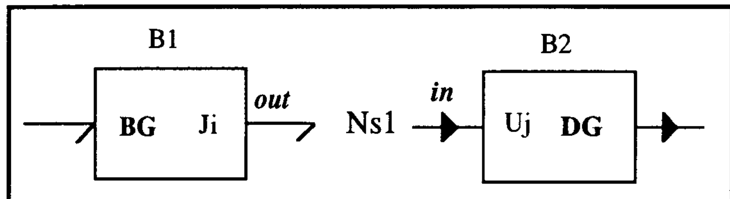
BU LILLE

Au niveau du code cela se traduit par la création d'un fait ***arc_dg*** de poid $\mathbf{g_n}$ entre le sommet $\mathbf{Y_i}$ de **B1** et un sommet source **S** (**Se** ou **Sf**) attaché à la jonction $\mathbf{J_j}$ de **B2**.

- Le type de la source sera précisé au moment de l'affectation de la causalité. Cependant si la nature du signal de sortie $\mathbf{Y_i}$ est un effort (resp 'flux'), nous conseillons de mettre une source d'effort **Se** (resp de flux **Sf**).



C. Couplage bond-graph + digraphe

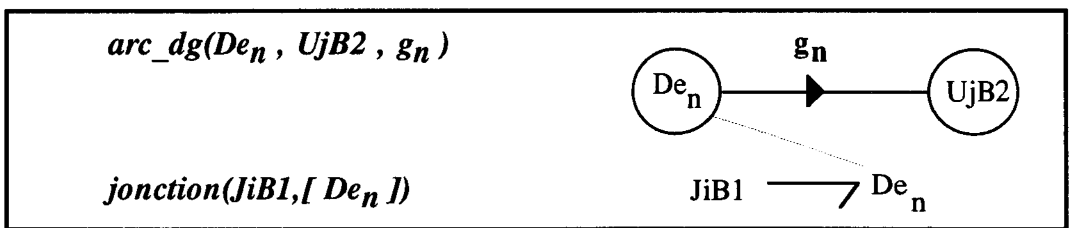


Principe :

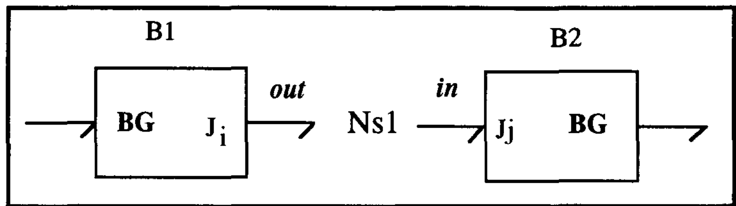
Pour coupler un bloc bond-graph et un bloc-digraphe, nous devons prendre le signal effort ou flux au niveau de la sortie ***out*** du bloc **B1**. Cela revient à mettre un détecteur (d'effort **De** ou de flux **Df**) sur la jonction $\mathbf{J_i}$ du port ***out*** sélectionné.

Celui-ci devient le sommet de sortie pour le bloc **B1**, autrement dit on crée un ***arc_dg*** de poids $\mathbf{g_n}$ entre $\mathbf{De_n}$ ou $\mathbf{Df_n}$ et l'entrée $\mathbf{U_j}$ du port ***in*** du bloc **B2.**

Si $\mathbf{U_j}$ est un effort, nous conseillons un détecteur d'effort **De**, si $\mathbf{U_j}$ est un flux, le même traitement est effectué avec un détecteur d'effort **Df.**



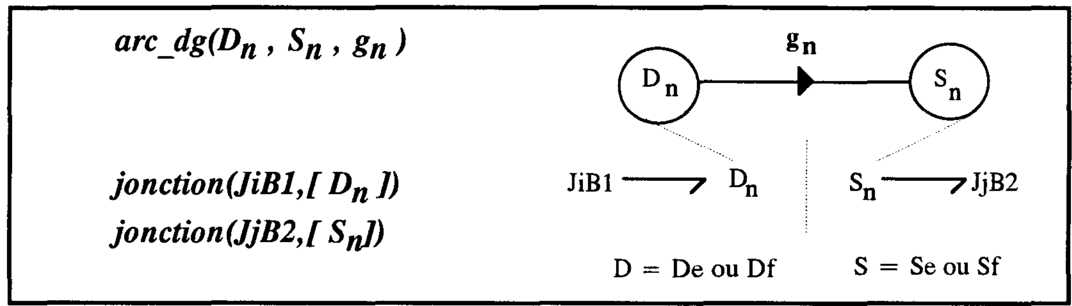
D. Couplage bond-graph avec un noeud signal



Principe :

Le couplage de deux blocs bond-graphs par l'intermédiaire d'un noeud signal consiste, non pas à lier une puissance, mais un signal (effort ou flux) du port ***out*** du bloc **B1** avec le signal du port ***in*** du bloc **B2**.

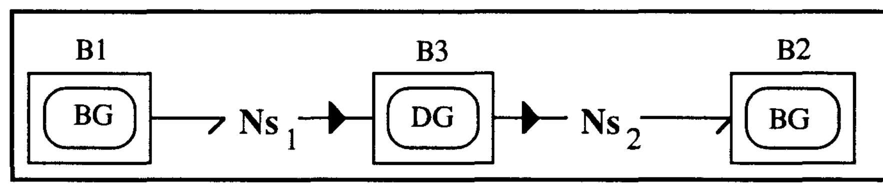
Cela se traduit par l'adjonction d'un détecteur **D (De** ou **Df)** sur la jonction $\mathbf{J_i}$ de **B1** et d'une source **S (Se** ou **Sf)** sur la jonction $\mathbf{J_j}$ de **B2** et la création d'un fait ***arc_dg*** de poids $\mathbf{g_n}$ entre les sommets **D** et S.



Lorsque nous rencontrons ce cas de figure dans une description, nous signalons à l'utilisateur la possibilité de choisir par un dialogue les détecteurs **D** et les sources **S** entre les quatres combinaisons suivantes :

S = Se	Source d'effort contrôlée	S = Sf	Source de flux contrôlée
& D = De ⇒	en effort	& D = Df ⇒	en flux
& D = Df ⇒	en flux	& D = De ⇒	en effort

Remarque : Une variante de ce couplage est de proposer à l'utilisateur de mettre un bloc digraphe (DG) **B3** intermédiaire entre les blocs bond-graph **B1** et **B2**. On retrouve ainsi le traitement BG + DG + BG décrit plus haut.



ARCHER, au stade actuel ne manipule que des codes bond-graphs. Nous pouvons envisager à l'avenir, un module qui puisse réunir les informations déduites d'une analyse graphique du digraphe, selon les résultats proposés par REINSCHKE [88] et d'une analyse structurelle à partir du bond-graph à l'aide des informations sur les boucles causales combinées aux travaux de SUEUR et RAHMANI [92-93].

IV.6.Conclusion

Dans ce chapitre nous avons précisé la grammaire des langages de type déclaratif définis au chapitre II. Ceux-ci sont basés sur une syntaxe rigoureuse à partir de règles de cohérence et de bon sens. Ces langages permettent de réaliser le "couplage" entre des blocs (physique, bond-graph ou digraphe) déduites du modèle (structurel ou fonctionnel) fourni par l'utilisateur.

La démarche globale utilise les principes orientée de la programmation objets pour le développement d'une solution. C'est une approche descendante et systématique par affinements successifs visant à mettre en évidence des blocs dans le modèle étudié.

Lorsqu'un type de bloc est représenté un nombre de fois dans le système, il n'est pas nécessaire de définir et de stocker tous les blocs qui lui sont associés. Il suffit, dans la partie déclaration, de signaler pour un type de bloc donné le nombre d'occurrence de cette sous-partie.

Il faut veiller à avoir une homogénéité dans la numérotation des informations afin d'éviter les redondances des indices associés aux composants bond-graphs. Le fichier de descendance (*.hlp) associé à une renumérotation des éléments permet de résoudre ce problème. En effet, l'unicité des éléments joue un rôle important lors de la phase de calculs des équations et la phase de simulation dans laquelle chaque élément est associé à une valeur numérique.

Le bloc est une entité qui caractérise une sous-partie représentée par un modèle physique, bond-graph ou digraphe, avec un ou plusieurs ports d'entrées (***in***) et un ou plusieurs ports de sorties (***out***) de nature.

Les informations qui caractérisent un bloc sont sauvegardées dans une base de données, ainsi que les règles permettant de le connecter à un autre bloc, tout en respectant les grandeurs physiques ou les signaux (effort ou flux) mises en jeu lors de cette connexion.

Nous avons dissocié le modèle des ports qui lui sont associés en les sauvegardant respectivement dans un fichier *.bg ou *.dg selon le type de modèle (BG ou DG) et un fichier *.blk pour les ports.

Cette façon de procéder nous a permis de trouver des règles d'association en ne travaillant que sur la description topologique des blocs, le nombre et la nature des ports, indépendamment du modéle bloc proprement dit.

- Si les blocs à connecter possèdent plusieurs entrées et sorties un choix doit s'opérer parmi les ports présents.

- Si les ports sélectionnés ne sont pas de même nature : il ne peut pas y avoir de connexions directes.

Nous avons proposé d'automatiser ces deux étapes de sélection à travers un dialogue avec l'utilisateur.

Pour coupler les blocs, nous avons introduit le noeud quit représente une entité de connexion généralisée comparable à un système d'aiguillage. Lorsque les blocs sont de type bond-graph, nous utilisons des noeuds **Ne** et **Nf** où la puissance s'égalise et qui permettent de vérifier la validité du point de vue physique des différentes connexions.

Le couplage des blocs digraphes avec les blocs bond-graphs se fait à travers un noeud **Ns** qui caractérise le passage des signaux effort ou flux entre l'entrée **U** ou la sortie **Y** d'un digraphe et une source (**Se** ou **Sf**) ou un détecteur (**De** ou **Df**) dans un bond-graph.

Le langage développé utilise la programmation orientée objet (POO). Cette approche semble naturelle dans cette application. En effet, un des axiomes de la POO consiste à modéliser les composants d'un problème en tant que composants en interaction. Cela permet d'augmenter le niveau "d'intelligence" des programmes.

Nous avons créé un langage à travers un objet intelligent possédant à la fois les méthodes nécessaires à son exploitation et les règles améliorant l'analyse des cas et réduisant au minimum l'appel à l'utilisateur sauf en cas de conflit, auquel cas un dialogue s'opère avec l'utilisateur.

CONCLUSION GENERALE

CONCLUSION GENERALE

Le travail présenté dans ce mémoire contribue à faire du logiciel ARCHER un outil performant pour la modélisation.

La démarche bond-graph, par la construction du bond-graph à mots, repose sur la décomposition d'un système physique en sous-systèmes échangeant de la puissance. La phase de construction du modèle global par association de sous-systèmes, souvent de natures différentes, peut s'avérer difficile.

Pour cela, nous avons défini plusieurs langages de type déclaratif, et développé un compilateur en Turbo-Prolog qui permet la génération d'un code bond-graph à partir des différents niveaux de description d'un système physique, tels que :

- un langage bond-graph à travers les jonctions et les liens,
- un langage utilisateur, avec les notions de série et parallèle,
- un langage bond-graph à mots avec les concepts de blocs et noeuds.

Des tests de cohérence des données, tant du point de vue syntaxique que sémantique, ainsi que des analyses de validité pré-causale ou physique du modèle sont effectués systématiquement, avec retour éventuel vers l'utilisateur en cas de problème.

Nous avons utilisé une démarche de programmation orientée objet, pour la définition de classes d'objets, ce qui augmente le niveau "d'intelligence" des programmes et l'aspect réutilisabilité des blocs construits à l'aide des langages proposés.

En effet, la possibilité d'utiliser une bibliothèque de blocs pré-définis contribue à améliorer le niveau de convivialité d'ARCHER pour la construction d'un modèle ou la création d'autres blocs.

Un des atouts d'ARCHER, par rapport aux autres logiciels utilisant l'outil bond-graph, est l'aspect analyse structurelle à partir de manipulations causales et de la détermination des chemins causaux et boucles causales.

Lorsqu'un des sous modèles sont sous forme d'équations mathématiques, il apparaît possible d'appliquer les outils de l'approche structurelle par digraphes sur cette sous-partie du système, et de les combiner avec ceux obtenus en bond-graph, ce qui laisse espérer une généralisation importante des résultats méthodologiques obtenus au laboratoire par l'équipe bond-graph.

La conception, la mise en place et la conduite d'un projet comme ARCHER demande la collaboration de plusieurs personnes. En ce qui nous concerne, nous nous sommes placés en amont des différents modules du logiciel pour lui fournir des modèles en langage texte.

L'une des perspectives de développement est d'adapter les langages proposés à un environnement graphique tel que Windows. Cela demande de repenser la façon de dessiner à l'écran le modèle (bond-graph, physique, blocs et noeuds) mais ne remet pas en question les différentes analyses et expertises proposées dans ce mémoire ainsi que la génération du code bond-graph nécessaire au fonctionnement des modules d'ARCHER.

ANNEXES

ANNEXE 1 : LANGAGE ET GRAMMAIRE

De façon formelle, un langage se définit à partir de la grammaire qui le produit [GENTHON 89]. Nous allons commencer à donner quelques définitions générales qui nous aiderons à formaliser la notion de langage [NOYELLE 88].

1. Définitions

Vocabulaire (V)

Est un ensemble fini dont les éléments sont appelés "**symboles**".

Alphabet (A)

Est l'ensemble des symboles terminaux du vocabulaires V.

Vocabulaire non terminal (Vn)

Est l'ensemble des symboles terminaux du vocabulaire V. (*)

Une phrase

Est une suite quelconque, de longueur finie, de symboles terminaux tirés d'un alphabet donné.(**)

Un lexème

Sera soit une suite de symboles terminaux adjacents dans une grammaire, soit une construction de la grammaire dont la syntaxe est simple et qui se comporte comme un élément indissociable vis-à-vis de l'analyse syntaxique.(*)

Exemple : Le monoïde libre engendré par l'alphabet $A = \{a,b\}$est :
$A^* = \{ \varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, \ldots\}$ ou "ε" représente la phrase vide.

La Notation de Backus-Naur (NBN)

Elle a été définie à la fin des années 50 pour pouvoir spécifier rigoureusement la syntaxe du langage. Cette notation utilise les caractères :

' < ' et ' > ' servant à indiquer les métanoms, c'est-à-dire les noms que l'on donne aux différentes constructions permises par la grammaire.

'::=' signifiant "se compose de".

'|' traduisant le "ou" dans le sens d'une énumération.

(*) On a $V = Vn \cup A$ avec Vn et A disjoints.

(**) L'ensemble de toutes les phrases que l'on peut construire sur un alphabet A est un monoïde libre noté A^*

Exemple :

Soit à définir la syntaxe de la construction "nombre entier"

<nombre entier> ::= **<chiffre>** |
<chiffre> <chiffre> |
<chiffre> <chiffre> <chiffre> | ...

<chiffre> ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |

Remarque :

Une difficulté apparaît dans la définition du métanom <nombre entier>. En effet, toutes les combinaisons de chiffres n'ont pas été représentées. On tourne cette difficulté en introduisant la récursivité :

<nombre entier> ::= **<chiffre>** |
<nombre entier> <chiffre>

Cette possibilité de définition récursive donne beaucoup de puissance à la NBN.

Une production

Est une règle de réécriture utilisant la NBN. Elle se présente sous cette forme :
$X ::= Y$ avec X et $Y \subset V^*$ et $X \neq \varepsilon$

Une grammaire

Elle est représentée par le quadruplé : G (A, Vn, P, R)
A : est l'alphabet (Vocabulaire terminal),
Vn : le vocabulaire non terminal,
P : est l'ensemble des productions,
R : est la racine (élément distingué de Vn).

Le langage L(G)

Il est défini par la grammaire G et représente l'ensemble des phrases (c'est-à-dire des suites où il n'y a plus que des terminaux) dérivables de la racine R :

$$L(G) = \{ X \in A^* / R \Rightarrow\!> X \}.$$ (*)

(*) Une suite dérivable de R, où il reste encore des non- terminaux, s'appelle une forme sententielle.

2. Hiérarchie des grammaires

Il existe une hiérarchie des grammaires basée sur des restrictions imposées aux productions. C'est CHOMSKY qui, le premier, s'est intéressé à formaliser la notion de grammaire et en a défini quatre classes.

Les grammaires de type 0

Ceux sont celles pour lesquelles il n'existe aucune restriction pour les productions. Les langages correspondants sont reconnus par une machine de TURING générale. Ce type de système non limité est trop peu structuré pour pouvoir servir de grammaire.

Les grammaires de type 1

Toutes les production doivent être de la forme :

U X V :: = U Y V avec **U, V, Y $\in$ V* et X $\in$ Vn**

Les grammaires de type 2

M :: = X ou **M $\in$ Vn et X $\in$ V***

Les grammaires de type 3

Ceux sont des grammaires dites régulières dont les productions existent sous 2 formes :

M :: = a N	**grammaire régulière**	**M :: = N a**	**grammaire régulière**
M :: = a	**à droite**	**M :: = a**	**à gauche**

avec M, N $\in$ Vn et a $\in$ A

Exemple :

Les nombres décimaux peuvent être définis par la grammaire de type 2 représentée par le quadruplé :

G = ({0, 1, 2, ... ,9 },{<nombre décimal>, <nombre entier>,<chiffre>}, P ,<nombre décimal>})

P étant l'ensemble des productions suivantes :

<nombre décimal> :: = <nombre entier> . <nombre entier>

<nombre entier> :: = <chiffre> |
<nombre entier> <chiffre>

<chiffre> :: = 0 | 1 | 2 | 3 | ... | 9

ANNEXE 2 : AUTOMATE EN PROLOG

Un **automate à nombre fini d'états** est une machine abstraite caractérisée par :

1. Un nombre fini d'états dans lequel l'automate peut se trouver ; certains de ces états sont spéciaux car ils caractérisent des états initiaux ou finals.

2. Un nombre fini de transitions, c'est-à-dire de passage d'un état à un autre sur la réception d'un signal d'entrée. On peut toutefois prévoir des transitions dites nulles qui forcent le passages d'un état à un autre.

Donc globalement, nous avons besoin pour représenter un automate de 4 prédicats :

initial(champ1) et ***final(champ1)*** dans lequel **champ1** représente respectivement un état initial ou final de l'automate.

transition(champ1, champ2, champ3) qui, en fonction de l'état courant et du métanom, détermine l'état suivant avec **champ1** le numéro de l'état de départ, **champ2** le métanom attendu, **champ3** le numéro de l'état d'arrivée.

nulle(champ1, champ2) semblable au prédicat précédent avec une transition nulle, on passe directement de l'état de départ (**champ1**) à l'état d'arrivée (**champ2**).

Exemple :

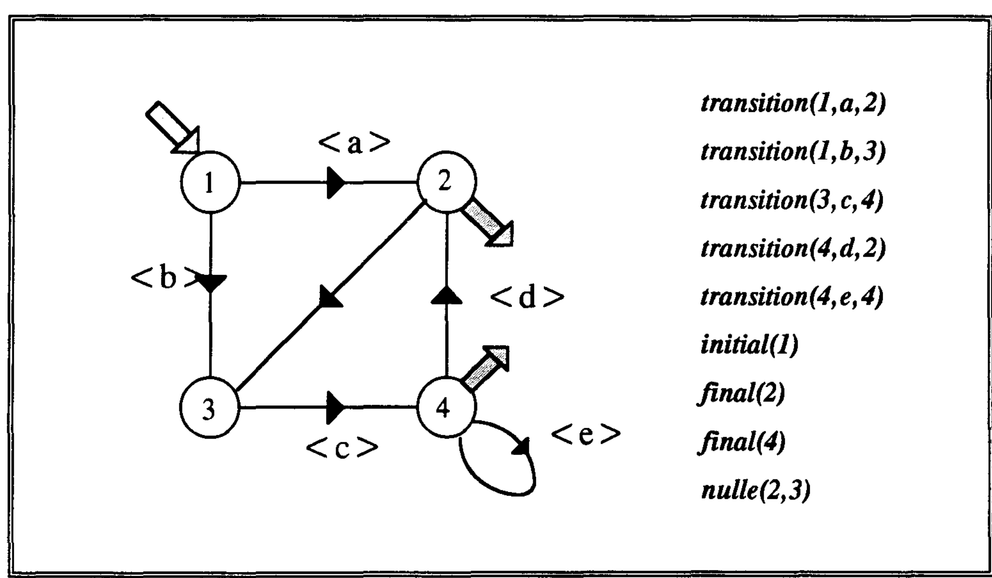
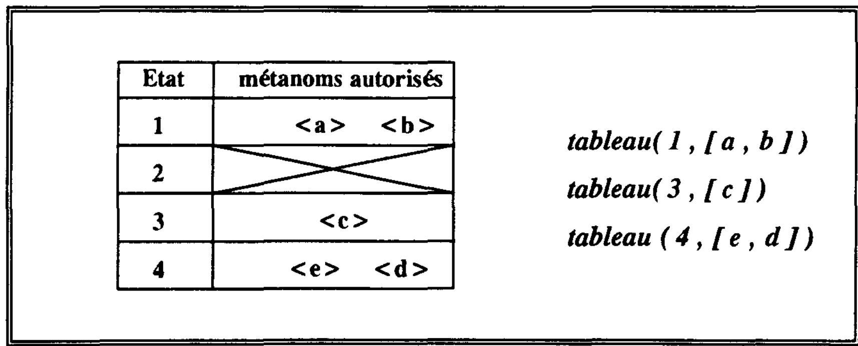


fig A.2.1: Exemple d'un automate.

Nous avons besoin du tableau des métanoms autorisés (permission d'adjacence). Pour cela nous utilisons le prédicat ***tableau(champ1,champ2)*** dans lequel **champ1** représente un état de l'automate et **champ2** la liste des transitions autorisées.



Etat	métanoms autorisés
1	<a> <b>
2	
3	<c>
4	<e> <d>

tableau(1 , [a , b])
tableau(3 , [c])
tableau (4 , [e , d])

fig A.2.2 : tableau des adjacences relatif à l'automate de la figure A.2.1.

Lors de la lecture de la ligne, nous devons identifier la nature de chaque lexème. Cette reconnaissance peut être aborder de deux manières :

- Dans **la première**, l'état de départ **E1** est connu, le lexème **L** courant est lu et identifié à partir d'une table de symboles afin de trouver le type de métanom **M** auquel il appartient. Si un lexème n'est pas identifié, le message "terme inconnu" est annoncé et la progression de l'automate s'arrête sans autres diagnostics.

Ensuite prolog recherche grâce à son moteur d'inférence le fait ***transition(E1,M,X)*** qui contient en variables d'entrées (au sens de Prolog) l'état **E1** et le métanom **M** et en variable de sortie, l'état **X** qui sera instanciée à **E2**.

Si aucune transition n'est trouvée, le lexème **L** n'est pas à sa place dans la ligne, il y a erreur et l'automate tombe dans un état-puits qui active le message adéquat déduit du tableau des métanoms autorisés pour l'état courant E1.

Dans l'automate de la figure A.2.1 la ligne suivante : **' L1 M3 N1 N2 O2 '** est analysée de la manière suivante, sachant que **'Li'** est un lexème de type **<b>**, **'Mi'** de type **<c>** **'Ni'** de type **<e>** et **'Oi'** de type **<d>** (la variable **i** représente un indice numérique).

Ligne	Lexème	Type de métanom	Transition appelée	Transition trouvée
'L1 M3 N1 N2 O2'	'L1'	<b>	*transition(1,b,X)*	*transition(1,b,3)*
'M3 N1 N2 O2'	'M3'	<c>	*transition(3,c,X)*	*transition(3,c,4)*
'N1 N2 O2"	'N1'	<e>	*transition(4,e,X)*	*transition(4,e,4)*
'N2 O2'	'N2'	<e>	*transition(4,e,X)*	*transition(4,e,4)*
'O2'	'O2'	<d>	*transition(4,d,X)*	*transition(4,d,2)*

La ligne est vide et l'état 2 est final, la phrase fait partie de la grammaire associée à l'automate.

- Dans **la deuxième**, on se sert de la syntaxe du langage pour prédire et vérifier la nature du lexème qui sera lu dans la ligne. Comme dans le cas précédent nous avons connaissance de l'état de départ **E1**, et du lexème courant L (pas encore de son type). Prolog cherche tous les faits de la forme ***transition(E1,Y,X)***, **X** et **Y** étant des variables inconnues qui seront instanciées respectivement à **E2** et **M**.

Pour chaque transition trouvée, on identifie le lexème **L** avec le métanom **M**. Si le test réussit, l'état suivant est **E2**, sinon on recommence avec une autre ***transition(E1,Y,X)***.

Nous pouvons montrer son fonctionnement en reprenant l'exemple précédent :

Ligne	Lexème	Transition appelée	Transition trouvée	Type du Lexème	Etat suivant
'L1 M3 N1 N2 O2'	'L1'	*transition(1,Y,X)*	*transition(1,a,2)* *transition(1,b,3)*	type <a> ⇒non type <b> ⇒oui	Etat 3
'M3 N1 N2 O2'	'M3'	*transition(3,Y,X)*	*transition(3,c,4)*	type <c> ⇒oui	Etat 4
'N1 N2 O2"	'N1'	*transition(4,Y,X)*	*transition(4,d,2)* *transition(4,e,4)*	type <d> ⇒non type <e> ⇒oui	Etat 4
'N2 O2'	'N2'	*transition(4,Y,X)*	*transition(4,d,2)* *transition(4,e,4)*	type <d> ⇒non type <e> ⇒oui	Etat 4
'O2'	'O2'	*transition(4,Y,X)*	*transition(4,d,2)* *transition(4,e,4)*	type <d> ⇒oui	Etat 2

Les méthodes, en apparence semblables, diffèrent essentiellement dans l'identification du lexème.

- Dans la première méthode, l'identification se fait par une recherche pure, qui se traduit par une analyse poussée du lexème afin de connaître sans ambiguïté son type.

En prolog cette fonction peut se traduire par un prédicat : ***identifier(L,M)*** (e,s) qui renvoit le métanom **M** associé au lexème **L**. Si le nombre des métanoms est grand et si la syntaxe de ceux-ci est complexe, le procédé peut-être coûteux en temps machine.

- Dans la deuxième, la recherche est en quelque sorte dirigée. En effet, on connait le métanom **M** et on vérifie si le lexème **L** est de type **M**.

Le prédicat ***identifier*** fonctionne cette fois avec 2 variables en entrées et retourne un message vrai ou faux. Elle est plus rapide que la précédente.

Pour l'analyse syntaxique d'une ligne, nous utilisons le prédicat ***lexico_syntaxique*** que nous modifions de manière à introduire un nouveau prédicat ***automate*** qui permet de parcourir l'automate ainsi qu'un prédicat ***état-puits*** qui s'active losqu'il y a échec du prédicat ***automate*** sur un état courant :

automate(champ1,champ2)
Le **champ1** représente l'état de départ.

états-puits(champ1,champ2)
Le **champ1** caractérise l'état qui a échoué.

Pour les deux prédicats le **champ2** caractérise la chaîne de caractères à analyser.

Remarque : Il est facile d'enrichir empiriquement la structure du programme proposé, par l'ajout de prédicats supplémentaires correspondant à des erreurs que l'on a pas prévues. Petit-à-petit, on augmente l'efficacité du compilateur. Cela est d'autant plus facile que l'automate est déterministe.

Résolution du non-déterminisme

Nous allons garder le même traitement, en ajoutant un prédicat, à la description de l'automate, pour caractériser les transitions d'un état non-déteministe.

Nous avons vu que dans la plupart des cas, un automate 2-décidable suffisait à lever le conflit sur l'état en question.

Il suffit pour forcer la transition de lire non plus un lexème mais deux lexèmes consécutifs. Selon les types de métanoms attendus, on en déduit l'état suivant.

Nous nous définissons un nouveau prédicat ***transition*** qui intervient lorsque l'état de départ est un état non déterministe :

transition_bis(champ1, champ2, champ3, champ4)

Le **champ1** désigne l'état de départ (non-déterministe)
Le **champ2** le type de métanom du 1° Lexème.
Le **champ3** l'état d'arrivée.
Le **champ4** le métanom du 2° lexème.

Par exemple, l'état **4** de l'automate suivant est non-détermiste car la transition **<a>** mène soit à l'état **5**, soit à l'état **7**. Il faut lire une transition plus loin.

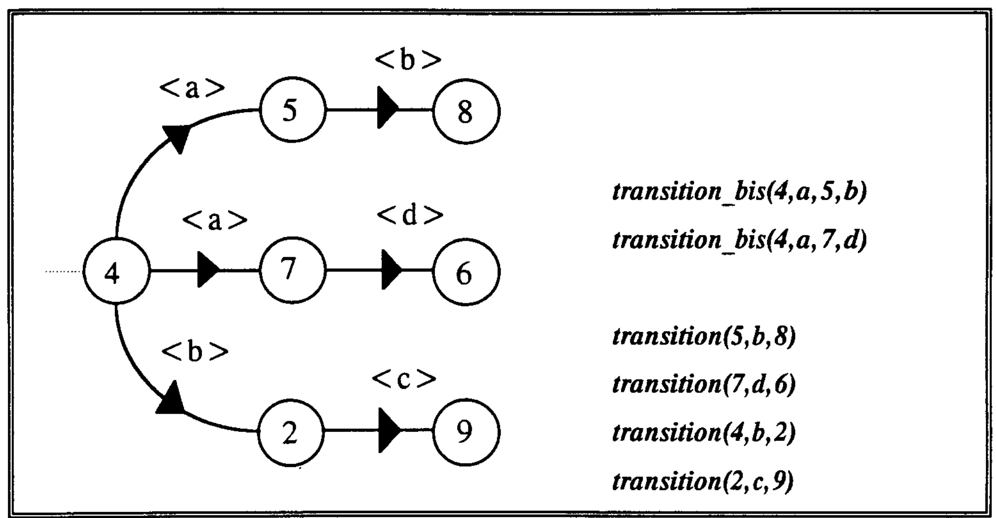


fig A.2.3. : Exemple d'un automate non-déterministe.

Pour traiter les transitions bis, il suffit d'ajouter une clauses automate dans le programme précédent.

Remarque : Pour certains, Prolog est considéré comme un langage non-déterministe. La fonction cut (!) permet d'introduire un semblant de déterminisme dans les clauses. En fait, les clauses sont parcourues dans l'ordre séquentiel strict du programmeur, les buts et sous-buts sont analysés dans l'ordre séquentiel strict des clauses. La programmation ne se limitera pas à disposer quelques cuts judicieux, mais surtout à méditer sur l'ordre des clauses.

ANALYSE LEXICO-SYNTAXIQUE

predicates

lexico_syntaxique(string)	
extraire_lexème(string, string, string)	
identifier(string,string)	Déclaration des prédicats
automate(integer, string)	
transition(integer, string, integer)	
initail(integer)	
final(integer)	
nulle(integer,integer)	

clauses

lexico_syntaxique(Ligne) :-	Lire une ligne
initial(E1),	Appel d'un état initial
automate(E1 , Ligne),!.	Activer le prédicat ***automate***
automate(E , ' ') :- final(E).	Arrêt si la ligne est vide et l'état **E** est final
automate(E1 , Ligne) :-	Passer automatiquement de l'état **E1** à l'état
nulle(E1 , E2),	**E2** si la transition est nulle
automate(E2 , Ligne),!.	appeler le prédicat automate avec le nouvel état **E2**.
automate(E1 , Ligne) :-	
extraire_lexème(Ligne , Lexème , Reste_de_la _ligne),	
traiter_lexème(Lexème),	Lire le premier lexème de la chaîne ligne
transition(E1 , Métanom , E2),	Le lexème appartient au dictionnaire
	Appel du fait ***transition*** (e,s,s)
identifier_lexème(Lexème , Métanom),	Le lexème est-il du type métanom ?
automate(E2 , Reste_de_la_ligne),!.	Appel récursif du prédicat ***automate*** avec le nouvel état **E2** et le reste de la ligne
automate(E1 , Ligne) :-	Echec du prédicat ***automate***
extraire_lexème(Ligne , Lexème , Reste_de_la _ligne),	Lire le lexème incorrect
état_puits(E1 , Lexème),!.	Activer le prédicat ***état_puits***
états_puits(E1 , Lexème) :-	Appel du fait tableau relatif à l'état E1.
tableau(E1 , Liste_métanoms),	Prédicat qui génère le message suivant :
écrire_message_erreur(Lexème , Liste_métanoms),!.	'le mot **Lexéme** est incorrect, il n'appartient pas à l'un des types de métanom suivant : **Liste_métanoms** '

Code	Commentaire
automate(E1 , Ligne) :-	
extraire_lexème(Ligne , Lexème , Reste_de_la _ligne),	Voir programme précédent
traiter_lexème(Lexème),	
transition(E1 , Métanom , E2),	Echec du prédicat ***automate*** car aucun fait
identifier_lexème(Lexème , Métanom),	***transition*** n'a été trouvé.
automate(E2 , Reste_de_la_ligne),!.	Appel du prédicat ***automate*** suivant
	Lire le premier et le deuxième lexème de
automate(E1 , Ligne) :-	la chaîne de caractères **Ligne**
extraire_lexème(Ligne , Lexème1 , R_ligne),	**Lexème1** et **Lexème2** appartiennent au
extraire_lexème(R_Ligne , Lexème2 ,New_ R_ligne),	dictionnaire
traiter_lexème(Lexème1),	Appel du fait ***transition_bis*** tel que
traiter_lexème(Lexème2),	**Lexème1** de type **Métanom1**
transition_bis(E1 , Métanom1 , E2 , Métanom2),	et **Lexème2** de type **Métanom2**
identifier_lexème(Lexème1 , Métanom1),	Appel récursif du prédicat ***automate***
identifier_lexème(Lexème2 , Métanom2),	
automate(E2 , New_R_ligne),!.	

ANNEXE 3 : ALGORITHMES DU LANGAGE SERIE PARALLELE

La ligne commence par un point (une masse) : une jonction **0** est créée pour ce point avec dans la liste le nom du point. La jonction **0** est en attente de branchement au niveau zéro. Ensuite, pour chaque terme nous appliquons les **Actions Sémantiques** suivantes :

'+'

A.1.1. On crée une jonction **0** avec une liste vide.

A.1.2. On y attache toutes les jonctions en attente de branchement au niveau courant dans le sens

jonctions en attente → jonction **0** courante

A.1.3. La jonction **0** devient la dernière jonction d'attache.

'('

A.2.1. La dernière jonction d'attache est empilée comme adresse de branchement.

A.2.2. Le niveau est incrémenté de 1.

')'

A.3.1. On dépile la dernière adresse de branchement.

A.3.2. Elle devient la dernière jonction d'attache.

A.3.3. On décrémente le niveau de 1.

A.3.4. On décrémente aussi le niveau de la dernière jonction en attente de branchement.

'élément' (autre qu'un transformateur)

A.4.1. On crée une jonction **1**.

A.4.2. On ajoute l'élément à la liste de la jonction **1**.

A.4.3. On crée un lien entre la dernière jonction d'attache et la jonction **1** dans le sens

dernière jonction → jonction **1** courante

A.4.4. On met la jonction **1** en attente avec le niveau courant.

'point'

A.5.1. Si la ***jonction(J,[point])*** existe déjà, on ne fait rien, sinon on crée une jonction **0** avec dans la liste le nom du point.

A.5.2. On crée un lien entre la dernière jonction d'attache et la jonction **0** dans le sens

dernière jonction → jonction **0** point courante

A.5.3. On met la jonction **0** en attente avec le niveau courant.

'/'

A.6. On ne fait rien.

'TFp ou TFs'

Le transformateur se compose de deux parties : le primaire (**TFp**) et le secondaire (**TFs**). Dans une description, ces deux éléments peuvent être rencontrés sans ordre défini, pour lesquels est associé un seul transformateur **TF**.

A.7.1. Si la jonction *j(TF,[])* existe déjà, on ne fait rien, sinon on crée une jonction **TF** avec une liste vide,

A.7.2. On crée une jonction **1** avec une liste vide

A.7.3. On crée un lien entre la dernière jonction d'attache et la jonction **1** dans le sens

dernière jonction → jonction **1** courante

A.7.4. La jonction **1** est mise en attente de branchement au niveau courant,

A.7.5.

'TFp' on attache la jonction **1** à la jonction **TF** 1 ⇀ TF

'TFs' on attache la jonction **TF** à la jonction **1** TF ⇀ 1

Nous allons décomposer les étapes du traitement en soulignant pour chaque lexème, son correspondant bond-graph (dictionnaire des éléments), les règles utilisées, les prédicats générés, et une illustration de l'ossature du bond-graph formée par les liens au fur-et-à mesure du traitement. Ces informations se disposent ainsi :

lexème lu	identification syntaxique	
Actions Sémantiques	faits générés dans la data-base	correspondance des faits au niveau du bond-graph

Nous effectuons ce traitement sur la première partie du circuit électrique appelé TRANSFO de la figure A3.1.

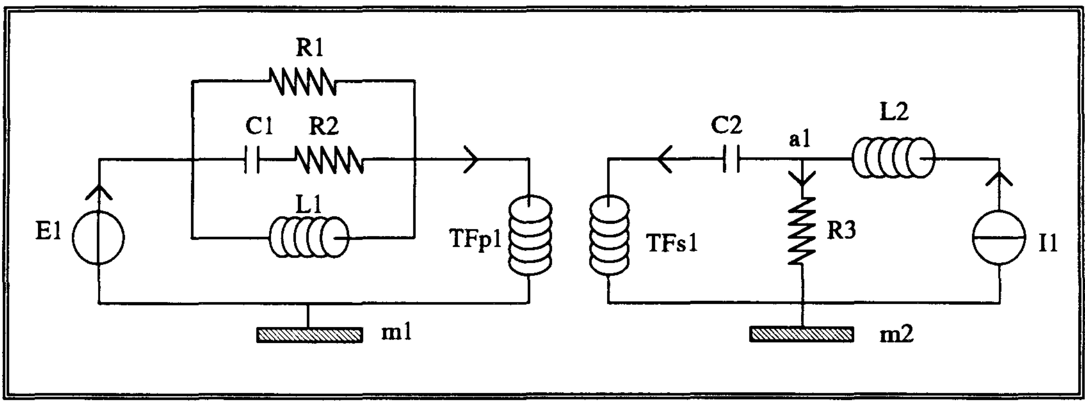


fig A.3.1 : Circuit électrique avec un transformateur.

La description se fait à l'aide de l'éditeur d'ARCHER illustré par la copie d'écran de la figure A.3.2.

Le succès de l'analyse syntaxique conduit à générer le fichier de conversion entre les éléments physique et bond-graphs 'TRANSFO.cnv'.

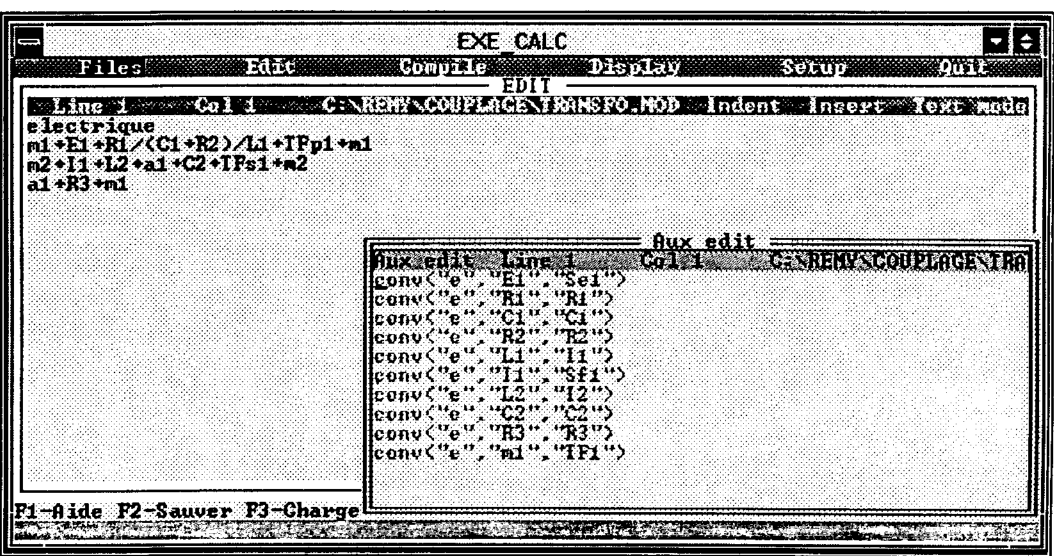


fig A.3.2 : Description du circuit électrique et fichier de conversion.

- Lire la première ligne

- Niveau des parenthèses = 0

lexème lu 'm1'	**identification** point	
Actions sémantiques **A.5.1.** **A.5.3.**	 *j(01,[m1])* *jonction_attente(01,0)*	01

'+'	signe+	
A.1.1. **A.1.2.** **A.1.3.**	*j(02,[ ])* *l(01,02)* *dernière_jonction(02)**	01 → 02 *

'E1'	élément : Se1	
A.4.1.& A.4.2. **A.4.3.** **A.4.4.**	*j(11,[Se1])* *l(02,11)* *jonction_attente(11,0)*	01 → 02 → 11 *

'+'	signe+	
A.1.1.	*j(03,[ ])*	01 → 02 → 11 → 03*
A.1.2.	*l(11,03)*	
A.1.3.	*dernière_jonction(03)**	

'R1'	élément : R1	
A.4.1.& A.4.2.	*j(12,[R1])*	03* → 12
A.4.3.	*l(03,12)*	
A.4.4.	*jonction_attente(12,0)*	

'/'	signe/	
A.6.	---------	---------

'('	parenthèse_ouv	
A.2.1.	*branchement(03)*	---------
A.2.2.	*niveau(1)*	

'C1'	élément : C1	
A.4.1. & A.4.2.	*j(13,[C1])*	03* → 12 ; 03* → 13
A.4.3.	*l(03,13)*	
A.4.4.	*jonction_attente(13,1)*	

'+'	signe+	
A.1.1.	*j(04,[ ])*	03 → 12 ; 03 → 13 → 04*
A.1.2.	*l(13,04)*	
A.1.3.	*dernière_jonction(04)**	

'R2'	élément : R2	
A.4.1. & 4.2	*j(14,R2)*	03 → 12 ; 03 → 13 → 04* → 14
4.3	*l(04,14)*	
4.4	*jonction_attente(14,1)*	

')'	parenthèse_ferm	
A.3.1.	*↑branchement(03)*	03* → 12 ; 03* → 13 → 04 → 14
A.3.2.	*dernière_jonction(03)**	
A.3.3.	*niveau(0)*	
A.3.4.	*jonction_attente(14,0)*	

'/'	signe/	
A.6.	---------	---------

'L1'	élément : I1	
A.4.1 & A.4.2	***j(15,[I1])***	12
A.4.3.	***l(03,15)***	03 * → 13 → 04 → 14
A.4.4.	***jonction_attente(15,0)***	15

'+'	signe+	
A.1.1.	***j(05,[ ])***	
A.1.2.	**dépiler les jonctions en attente**	12
	au niveau 0 ⇒	03 → 13 → 04 → 14 → 05 *
	l(15,05)	15
	l(14,05)	
	l(12,05)	
A.1.3.	***dernière_jonction(05)***	

'TFp1'	élément transformateur	
A.7.1.	***j(TF1,[ ])***	TF1
A.7.2.& A.7.3.	***j(16,[ ]) & l(05,16)***	↑
A.7.4.	***jonction_attente(16,0)***	05 * → 16
A.7.5.	***l(16,TF1)***	

+	signe+	
A.1.1.	***j(06,[ ])***	TF1
A.1.2.	***l(16,06)***	↑
A.1.3.	***dernière_jonction(06)***	05 → 16 → 06 *

m1	point	
A.5.1.	***j(01,[m1])***	01 → ... → 06 *
A.5.2.	***l(06,01)***	
A.5.3.	***jonction_attente(01,0)***	

Fin de la ligne, le traitement s'arrête pour reprendre à la ligne suivante.

Le bond-graph global obtenu est le suivant :

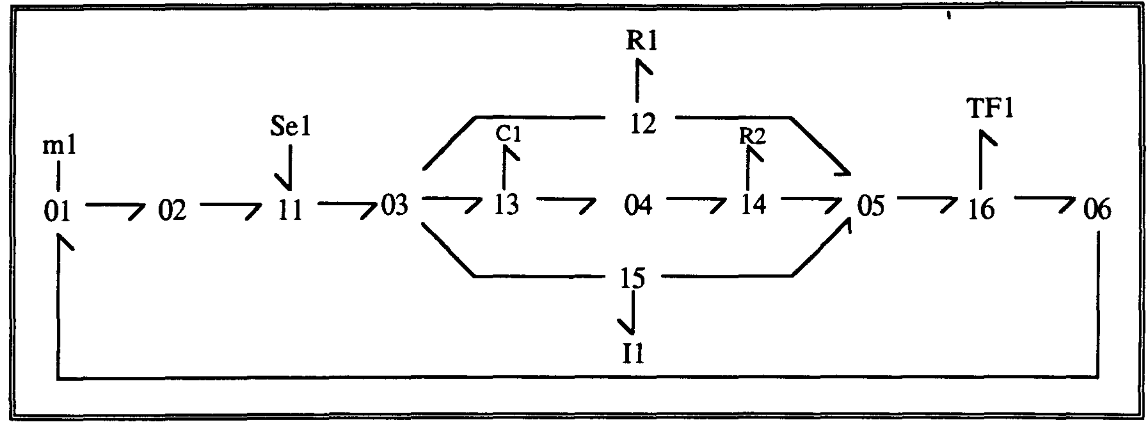


fig A.3.3 : Bond-graph non simplifié.

De même pour la ligne

'm2 + I1 + L2 + a1 + C2 + TFs1 + m2'

nous avons la génération du bond-graph non simplifié suivant :

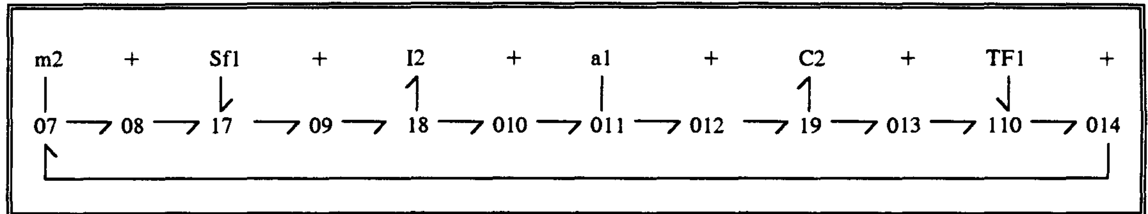


fig A.3.4 Bond-graph non simplifié de la deuxième ligne.

Enfin pour la dernière ligne

'a1 + R3 + m2'

nous avons :

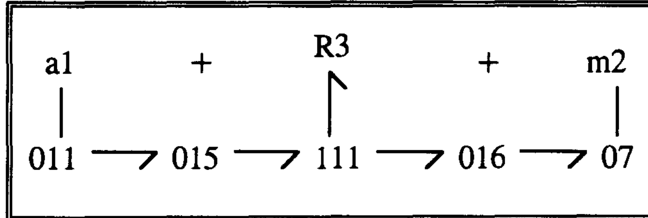


fig A.3.5 : Bond-graph non simplifié de la dernière ligne.

Il reste à simplifier le bond-graph et de stocker les prédicats ***jonction*** et ***lien*** dans un fichier *.bgs comme le montre la figure A3.6.

```
EXE_CALC
Files    Edit    Compile    Display    Setup    Quit
EDIT
Line 1   Col 1   TRANSFO.BGS   Indent   Insert   Text mode
j("02",["Se1"],0)
j("13",["R1"],0)
j("16",["I1"],0)
j("04",[],0)
j("TF1",[],0)
j("015",["R3"],0)
j("14",["C1","R2"],0)
j("110",["Sf1","I2"],0)
j("113",["C2"],0)
l("02","13")
l("02","14")
l("02","16")
l("13","04")
l("16","04")
l("015","113")
l("04","TF1")
l("14","04")
l("110","015")
l("TF1","113")
F1-Aide F2-Sauver F3-Charger F5-Zoom F7-Xcopie F8-Xedit  F10-Menu Esc-Annuler
```

fig A.3.6 : Code du bond-graph simplifié.

A partir de ce code, il est possible de visualiser le dessin formé par les jonctions et les liens du fichier *.bgs, grâce au module graphique d'ARCHER.

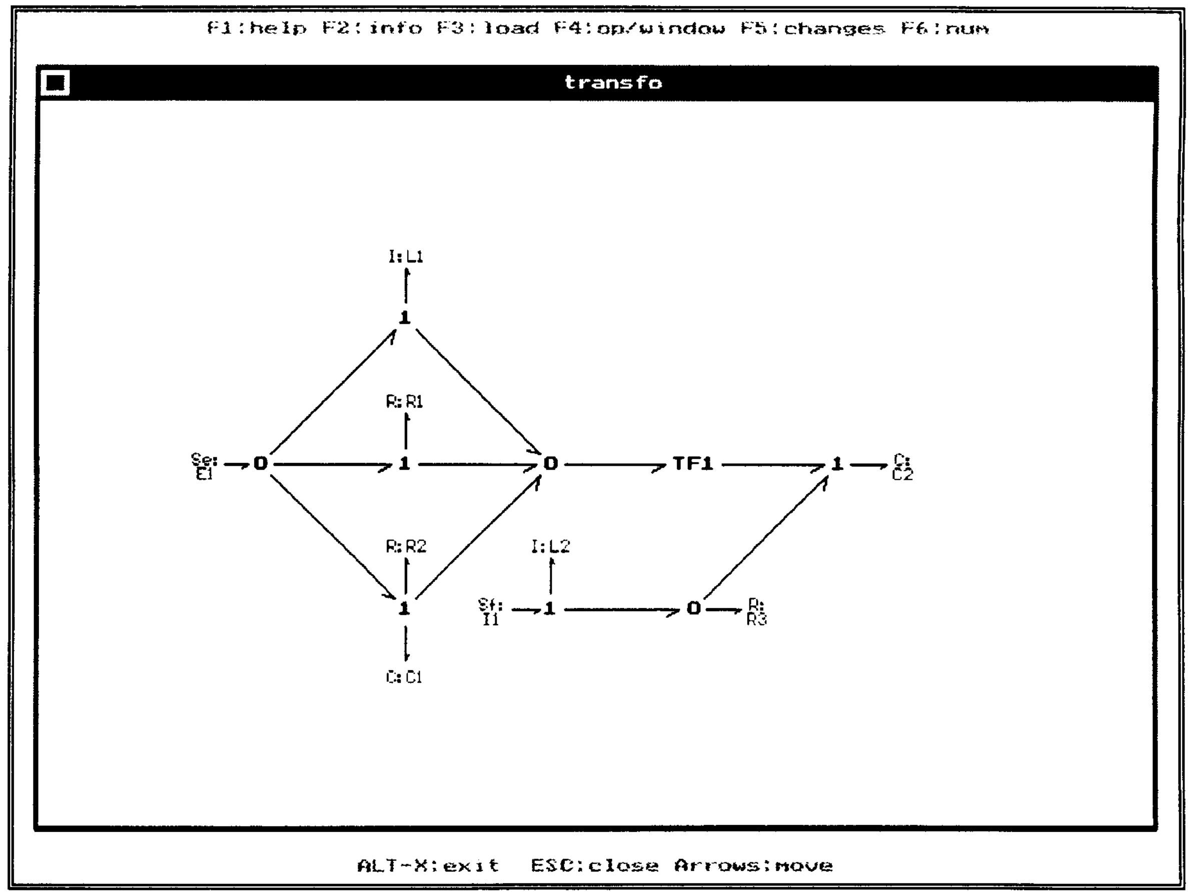
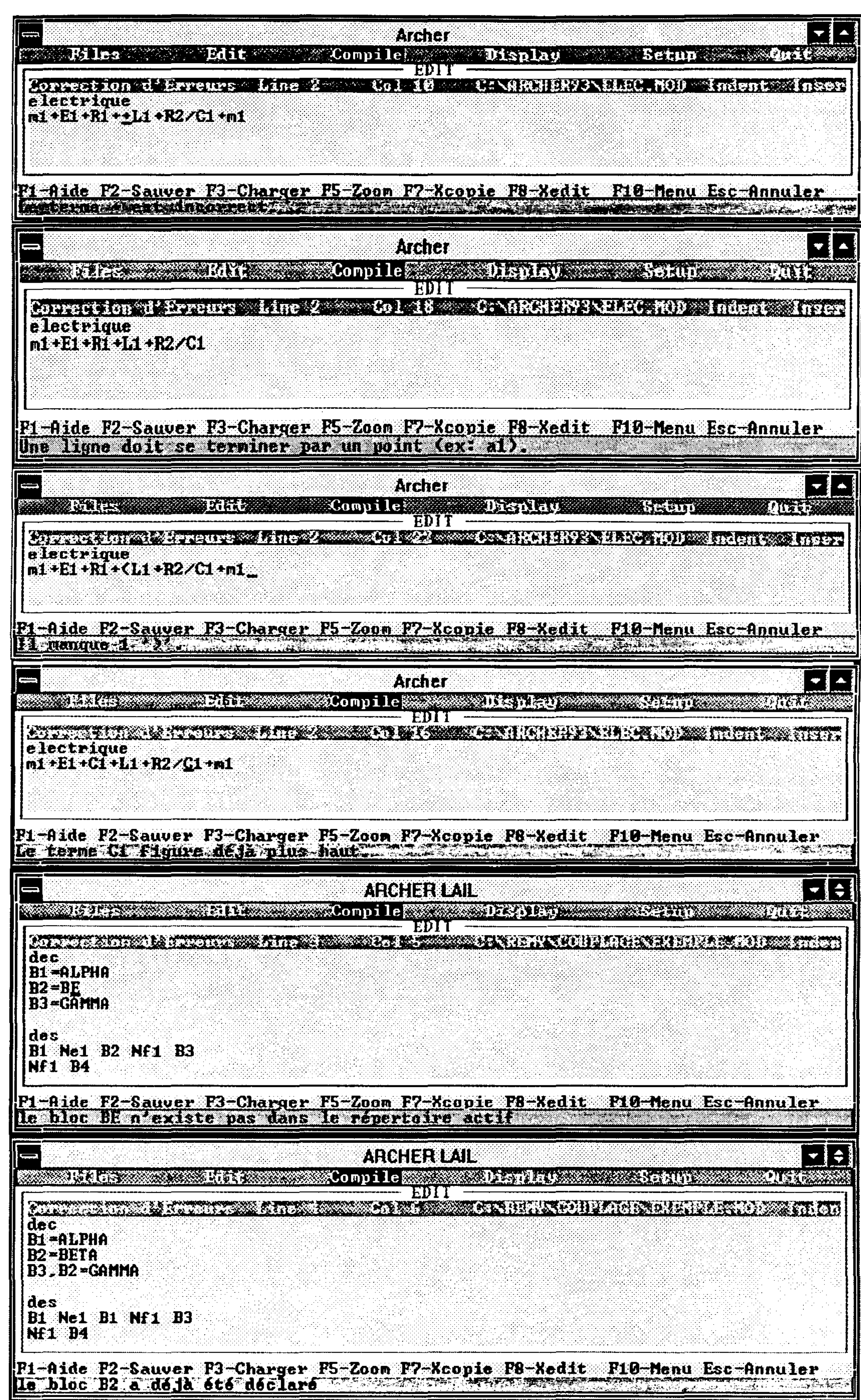


fig A.3.7 : Dessin du code bond-graph.

QUELQUES ERREURS SYNTAXIQUES DETECTEES PAR LE COMPILATEUR



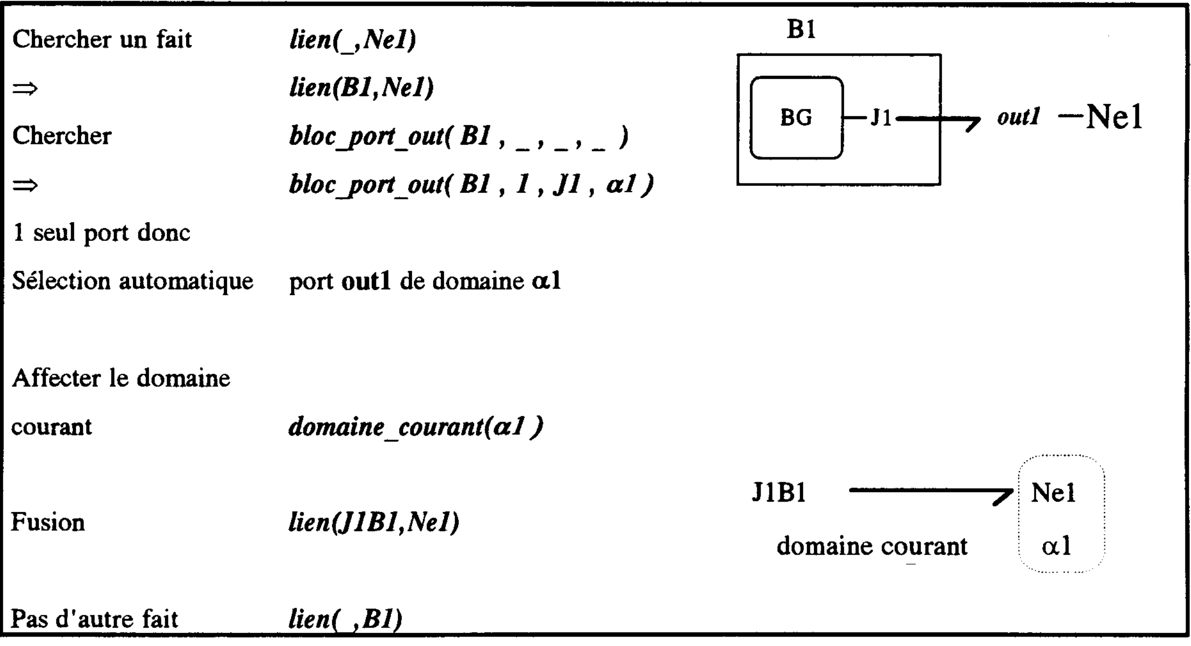
ANNEXE 4 : SESSION COMPLETE D'UN COUPLAGE BOND-GRAPH

Nous avons en mémoire tous les faits nécessaires pour le déroulement du couplage des blocs à travers la sélection (manuelle ou automatique des ports).

Nous allons décomposer les opérations du couplage à travers notre exemple.

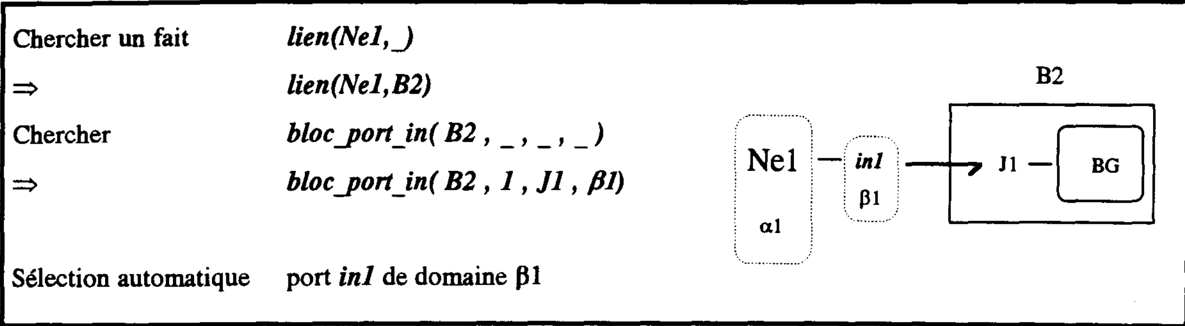
lien(B1,Ne1)	*bloc_port_out (B1 , 1 , J1 , α1)*
lien(Ne1,B2)	*bloc_port_in (B2 , 1 , J1 , β1)*
lien(B2,Nf1)	*bloc_port_out (B2 , 1 , J2 , β2)*
lien(Nf1,B3)	*bloc_port_out (B2 , 2 , J3 , β3)*
lien(Nf1,B4)	*bloc_port_in (B3 , 1 , J1 ,γ1)*
	bloc_port_in (B3 , 2 , J2 ,γ2)
ordre_noeud([Ne1 Nf1])	*bloc_port_in (B4 , 1 , J1 ,γ1)*
	bloc_port_in (B4 , 2 , J2 ,γ2)

B---Ne1



Chercher un fait	*lien(_,Ne1)*
⇒	*lien(B1,Ne1)*
Chercher	*bloc_port_out(B1 , _ , _ , _)*
⇒	*bloc_port_out(B1 , 1 , J1 , α1)*
1 seul port donc	
Sélection automatique	port **out1** de domaine α1
Affecter le domaine courant	*domaine_courant(α1)*
Fusion	*lien(J1B1,Ne1)*
Pas d'autre fait	*lien(,B1)*

Ne1---B



<u>**cas 1.** Si les domaines **α1** et **β1** sont égaux</u>

Il n'y a aucun problème, et l'opération de fusion s'effectue normalement.

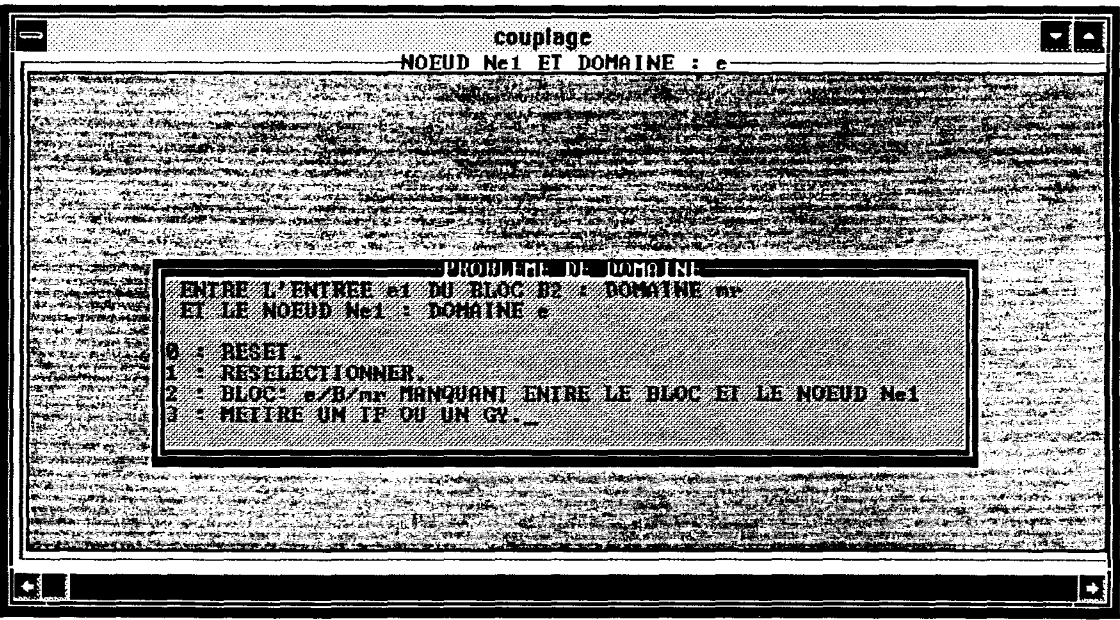
si $\alpha 1 = \beta 1$	*lien(Ne1,J1B2)*	Ne1 ⟶ J1B2

<u>**cas 2.** Si les domaines **α1** et **β1** sont différents</u>

Ici nous sommes en présence d'un conflit entre le domaine physique **α1** du noeud **Ne1** et le domaine **β1** du bloc **B2.**

Comme le **B2** ne possède qu'un seul port en entrée ***in1***, il n'est pas possible pour l'utilisateur d'en resélectionner un autre.

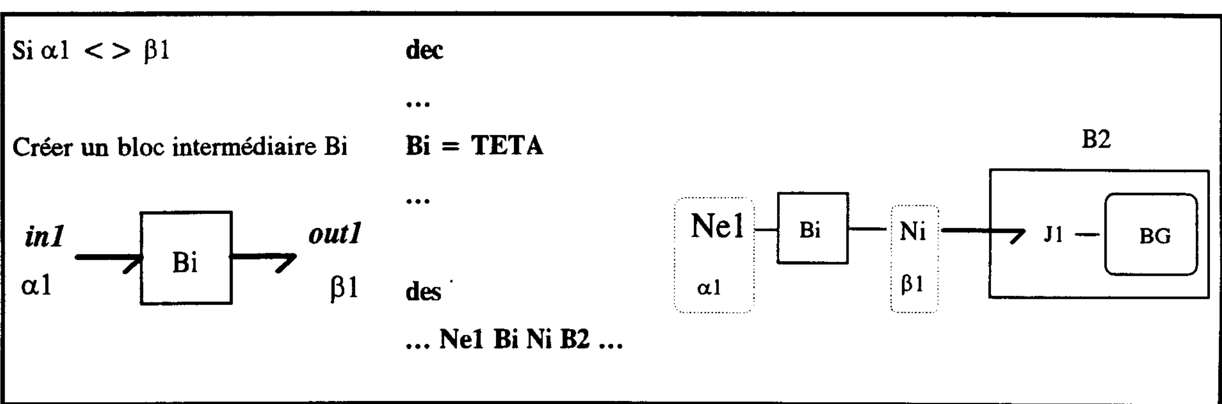
On lui propose 4 options :



Option 0. Elle provoque une interruption et l'utilisateur retourne à l'éditeur.

Option 1. Pas de resélection car le bloc B2 possède qu'une entrée.

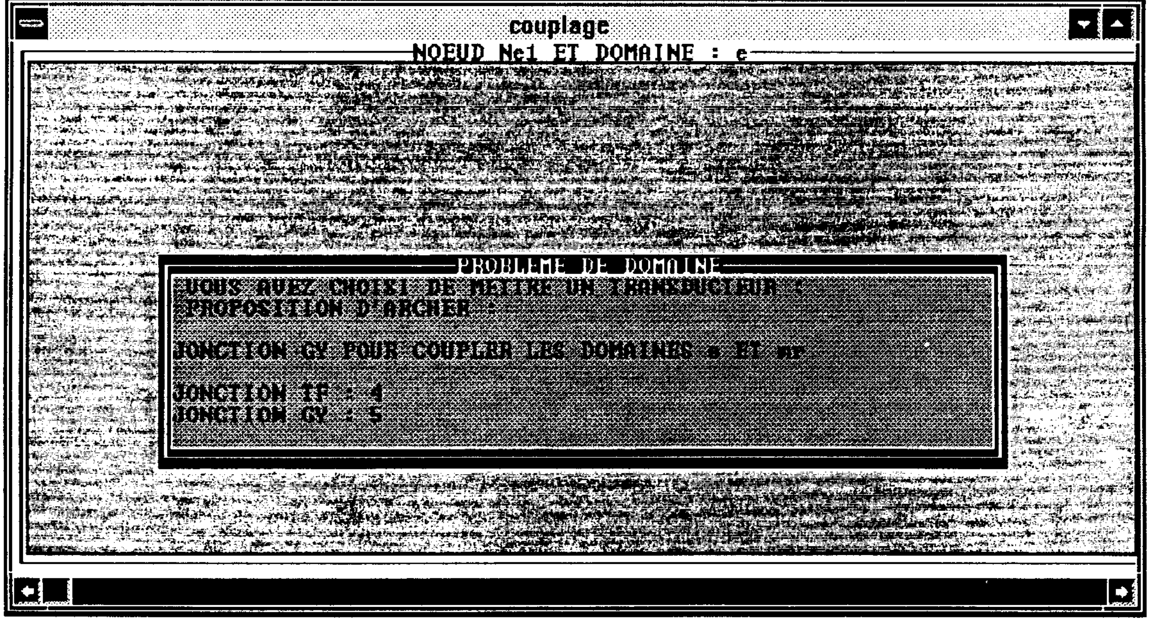
Option 2. De retourner à l'éditeur texte pour ajouter un bloc **Bi** dont les ports d'entrées et de sorties sont respectivement **α1** et en sortie **β1**

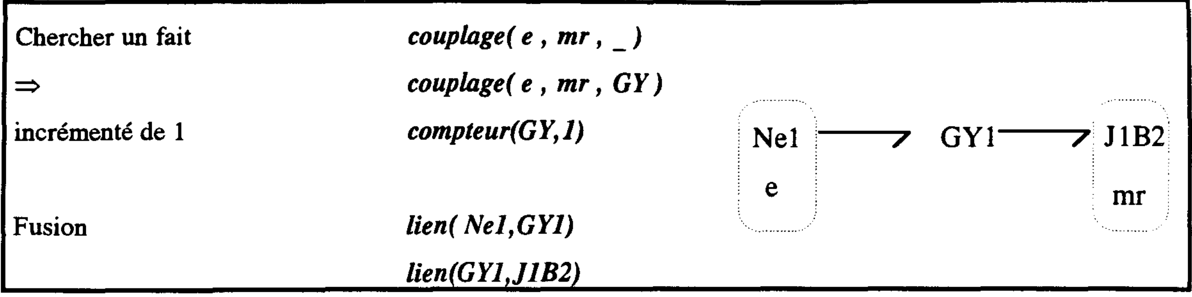


Option 3. D'ajouter un transducteur entre le noeud **Ne1** et le bloc **B2** en suivant la suggestion d'ARCHER.

A l'aide de l'analyse des domaines, ARCHER propose une suggestion pour le transducteur (transformateur ou gyrateur) en le cherchant, dans la base de données, le fait ***couplage*** qui correspond aux domaines en présence.

On incrémente l'indice du transducteur de 1 à travers le fait compteur.
Par exemple si **α1** = électrique (e) et **β1** = mécanique de translation (mr).





Chercher un fait	*couplage(e , mr , _)*
⇒	*couplage(e , mr , GY)*
incrémenté de 1	*compteur(GY,1)*
Fusion	*lien(Ne1,GY1)*
	lien(GY1,J1B2)

Option 4 : Il décide du transducteur **TF** ou **GY**.

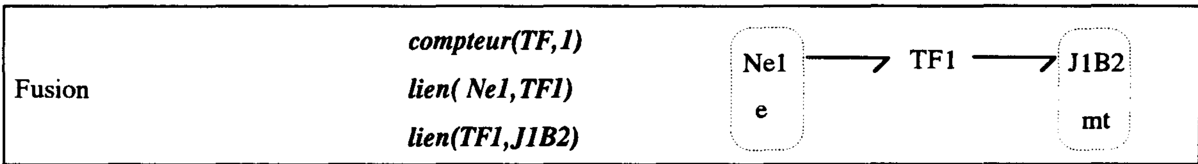
La loi associée sera écrite en fonction des variables physiques mises en présence. Par exemple pour les mêmes domaines que précédemment les variables sont déduites des faits suivants :

loi_tf(e,mr,u,i,Γ,ω) et ***loi_gy(e,mr,u,i,Γ,ω)*** à partir desquelles nous fabriquons les relations génériques :

$$\xrightarrow[i1]{u1} TF \overset{\cdot\cdot}{_{t}} \xrightarrow[\omega 2]{\Gamma 2} \qquad \xrightarrow[i1]{u1} GY \overset{\cdot\cdot}{_{g}} \xrightarrow[\omega 2]{\Gamma 2}$$

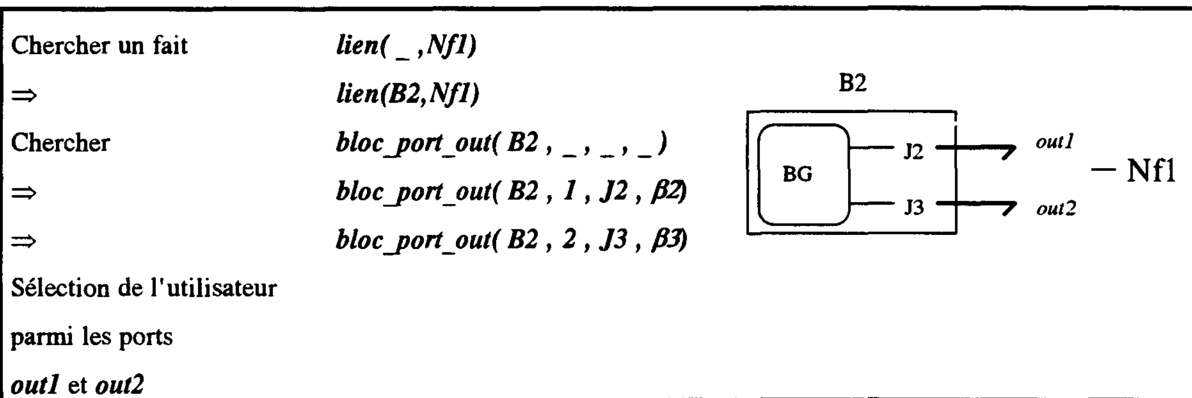
$$\begin{cases} u1 = t\,\Gamma 2 \\ \omega 2 = t\,.\,i1 \end{cases} \qquad \begin{cases} u1 = g\,.\omega 2 \\ \Gamma 2 = g\,i1 \end{cases}$$

Par exemple, l'utilisateur choisit un transformateur pour coupler le noeud **Ne1** et le bloc **B2**.

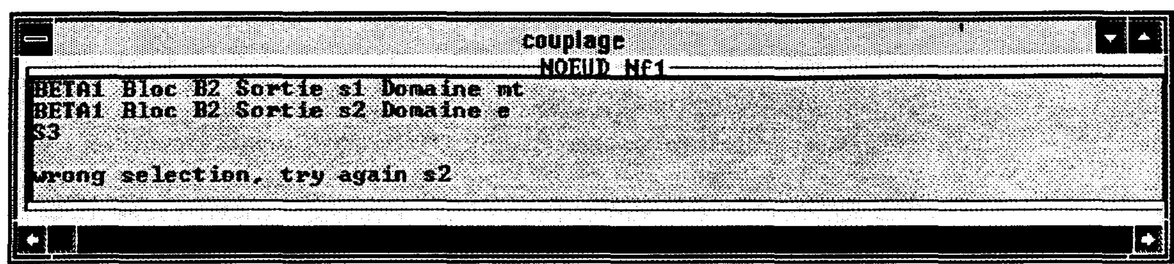


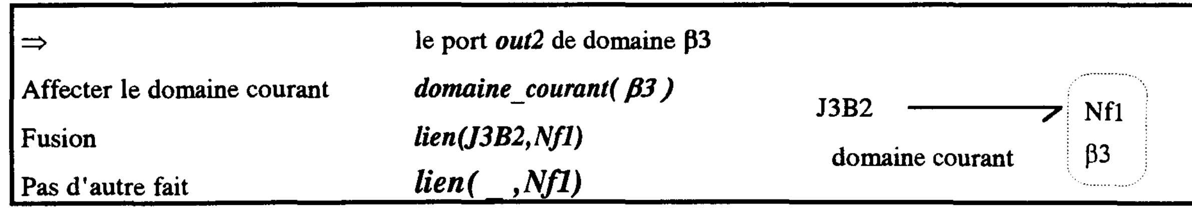
	compteur(TF,1)
Fusion	*lien(Ne1,TF1)*
	lien(TF1,J1B2)

B---Nf1

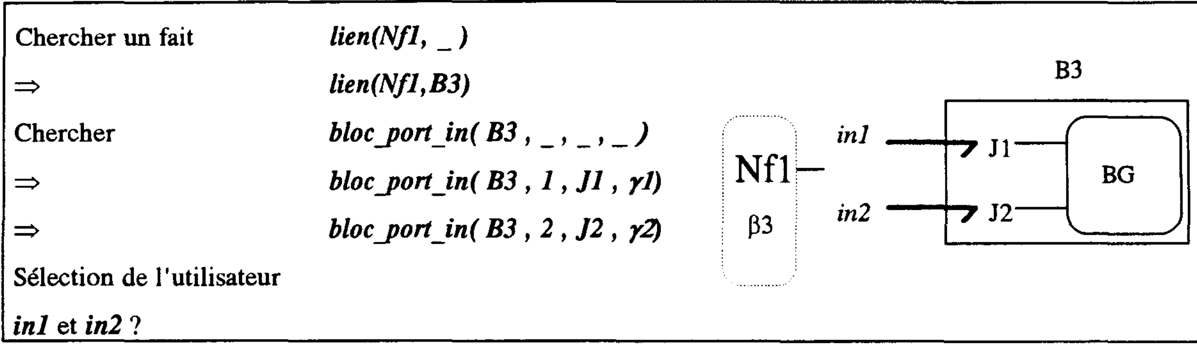


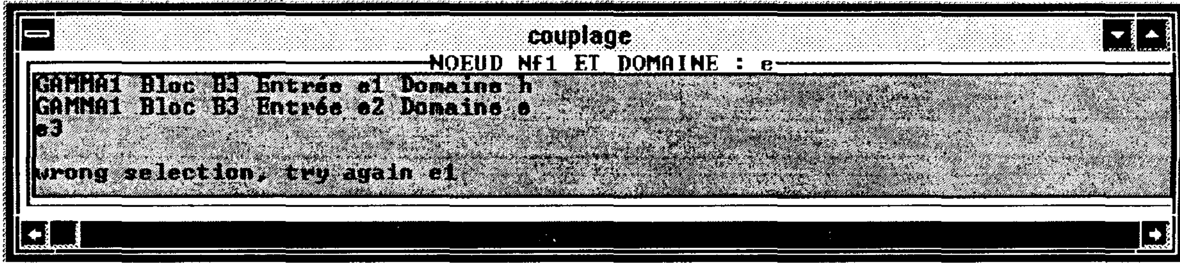
Chercher un fait	*lien(_ ,Nf1)*
⇒	*lien(B2,Nf1)*
Chercher	*bloc_port_out(B2 , _ , _ , _)*
⇒	*bloc_port_out(B2 , 1 , J2 , β2)*
⇒	*bloc_port_out(B2 , 2 , J3 , β3)*
Sélection de l'utilisateur parmi les ports *out1* et *out2*	





Nf1---B





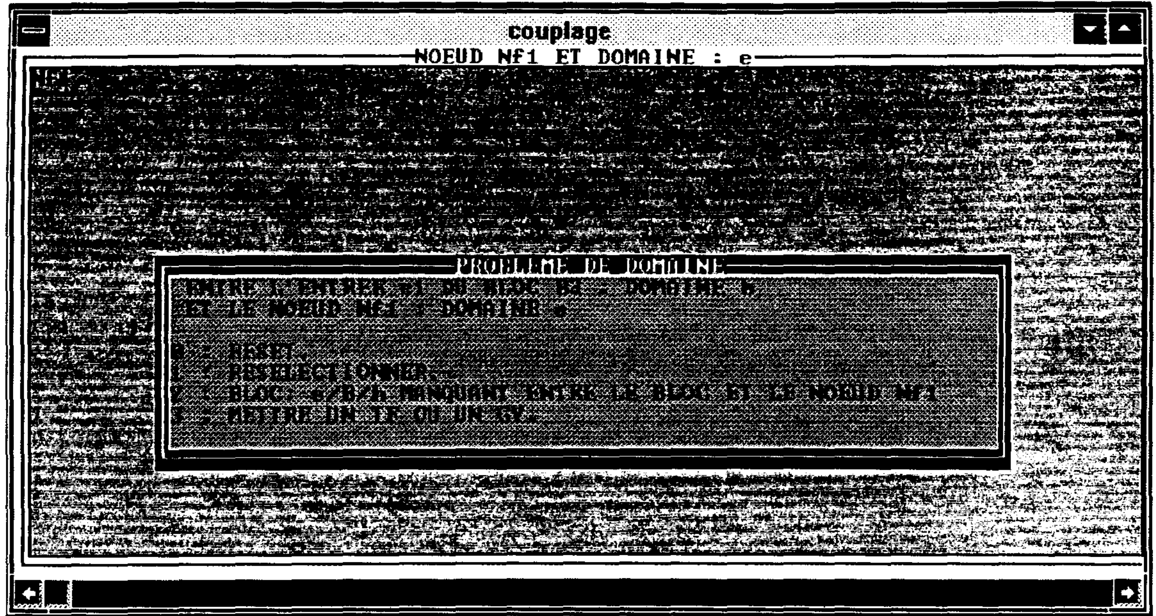
⇒ port ***in1*** de nature γ1

Comme pour le noeud précédent, nous pouvons avoir un conflit sur les domaines physiques β3 et γ1.

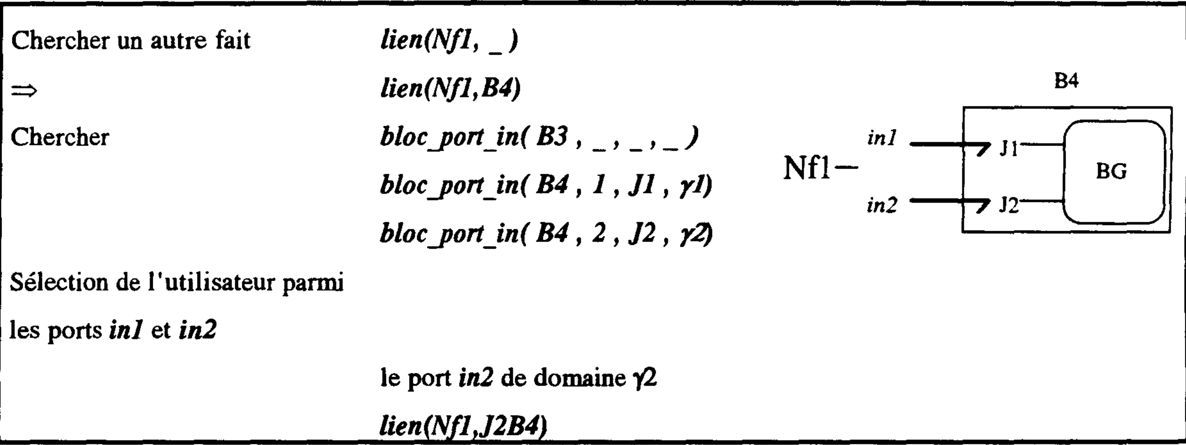
<u>**cas 1.** Si les domaines β3 et γ1 sont égaux</u>

<u>**cas 2.** Si les domaines α1 et β1 sont différents</u>

Option 1. L'utilisateur peut resélectionner un autre port en entrée, en l'occurrence le port **in2** de domaine γ2. Selon la nature de **β3** et γ2 nous appliquons soit le **<u>cas1</u>** soit le **<u>cas2</u>**.



Nf1---B



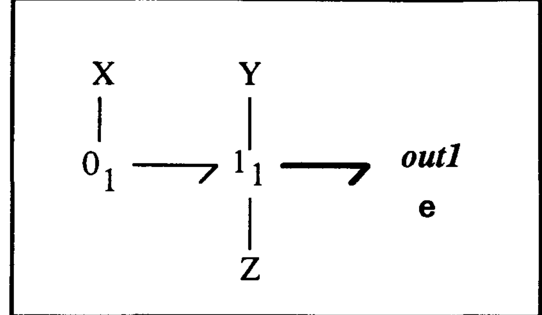
Chercher un autre fait	***lien(Nf1, _)***
⇒	***lien(Nf1,B4)***
Chercher	***bloc_port_in(B3 , _ , _ , _)***
	bloc_port_in(B4 , 1 , J1 , γ1)
	bloc_port_in(B4 , 2 , J2 , γ2)
Sélection de l'utilisateur parmi les ports ***in1*** et ***in2***	
	le port ***in2*** de domaine γ2
	lien(Nf1,J2B4)

Il n'y a plus d'autres faits ***lien(Nf1, _)*** et plus d'autres noeuds, donc la sélection est terminée.

Pour la suite de l'exemple nous supposons que les domaines physiques des blocs sont figés. Il nous faut à présent utiliser le modèle bond-graph de chaque type de bloc du modèle. Par exemple pour les blocs **ALPHA**, **BETA** et **GAMMA** nous avons les bond-graphs suivants.

α1 = électrique (e)
β1 = mécanique de rotation (mr)
β2 = mécanique de rotation (mt)
β3 = électrique (e)
γ1 = hydraulique (h)
γ2 = électronique (el)

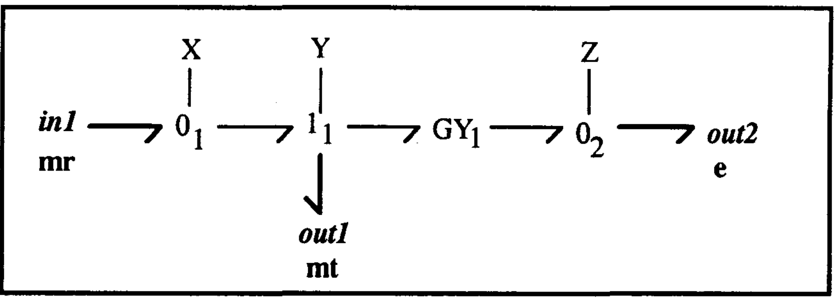
BLOC ALPHA



DESCRIPTION ET CODAGE DES DONNEES

jonction	*jonction(01 , [X])*	*out(1 , 11 , e)*
01 X	*jonction(11 , [Y Z])*	
11 Y Z	*lien(01 , 11)*	
lien		
01 11		
port		
11 out1 : e		

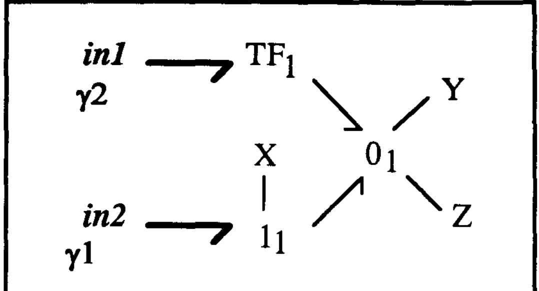
BLOC BETA



CODAGE DES DONNEES

jonction	*jonction(01 , [X])*	*in(1 , 01 , mr)*
01 X	*jonction(11 , [Y])*	*out(1 , 11 , mt)*
11 Y	*jonction(GY1 , [ ])*	*out(2 , 02 , e)*
02 Z	*jonction(02 , [Z])*	
lien	*lien(01 , 11)*	
01 11 GY1 02	*lien(11 , GY1)*	
port	*lien(GY1 , 02)*	
01 in1 : mr		
11 out1 : mt		
02 out2 : e		

BLOC GAMMA



DESCRIPTION ET CODAGE DES DONNEES

jonction	*jonction(11 , [X])*	*in (1 , TF1 , h)*
11 X	*jonction(01 , [Y , Z])*	*in (2 , 11 , e)*
01 X Y	*jonction(TF1 , [ ])*	
lien	*lien(TF1 , 01)*	
TF1 01	*lien(11 , 01)*	
11 01		
port		
TF1 in1 : h		
11 in2 : e		

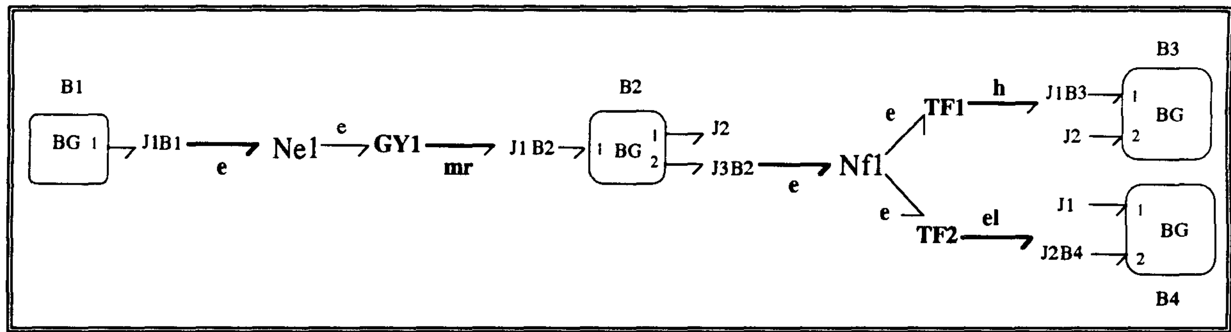


fig A.4.1: Modèle blocs & noeuds après sélection des ports et l'application des règles de couplage.

lien(J1B1,Ne1)	*jonction(GY1, [ ])*
lien(Ne1,GY1)	*jonction(TF1 , [ ])*
lien(GY1,J1B2)	*jonction(TF2 , [ ])*
lien(J3B2,Nf1)	
lien(Nf1,TF1)	
lien(Nf1,TF2)	
lien(TF1,J1B3)	
lien(TF2,J2B4)	

fig A.4.2 : Code intermédiaire généré après la sélection.

équivalent(ALPHA , ALPHA1 , B1)

équivalent(BETA , BETA1 , B2)

équivalent(GAMMA , GAMMA1 , B3)

équivalent(GAMMA , GAMMA2 , B4)

BLOC B1 ou ALPHA1

jonction(01B1 , [X1])	X1 est l'élément X1 du bloc B1 ou ALPHA1
jonction(11B1 , [Y 1])	Y1 ----------------Y1 ------------------------------
jonction(12B1 , [Z1])	Z1 ----------------Z1 ------------------------------
lien(01B1 , 11B1)	
lien(01B1 , 12B1)	

BLOC B2 ou BETA1

jonction(01B2 , [X2])	X2 est l'élément X1 du bloc B2 ou BETA1
jonction(11B2 , [Y 2])	Y2 ----------------Y1 ------------------------------
jonction(GY1B2 , [ ])	Z2 ----------------Z1 ------------------------------
jonction(02B2 , [Z2])	
lien(01B2 , 11B2)	
lien(11B2 , GY1B2n)	
lien(GY1B2 , 02B2)	

BLOC B3 ou GAMMA1

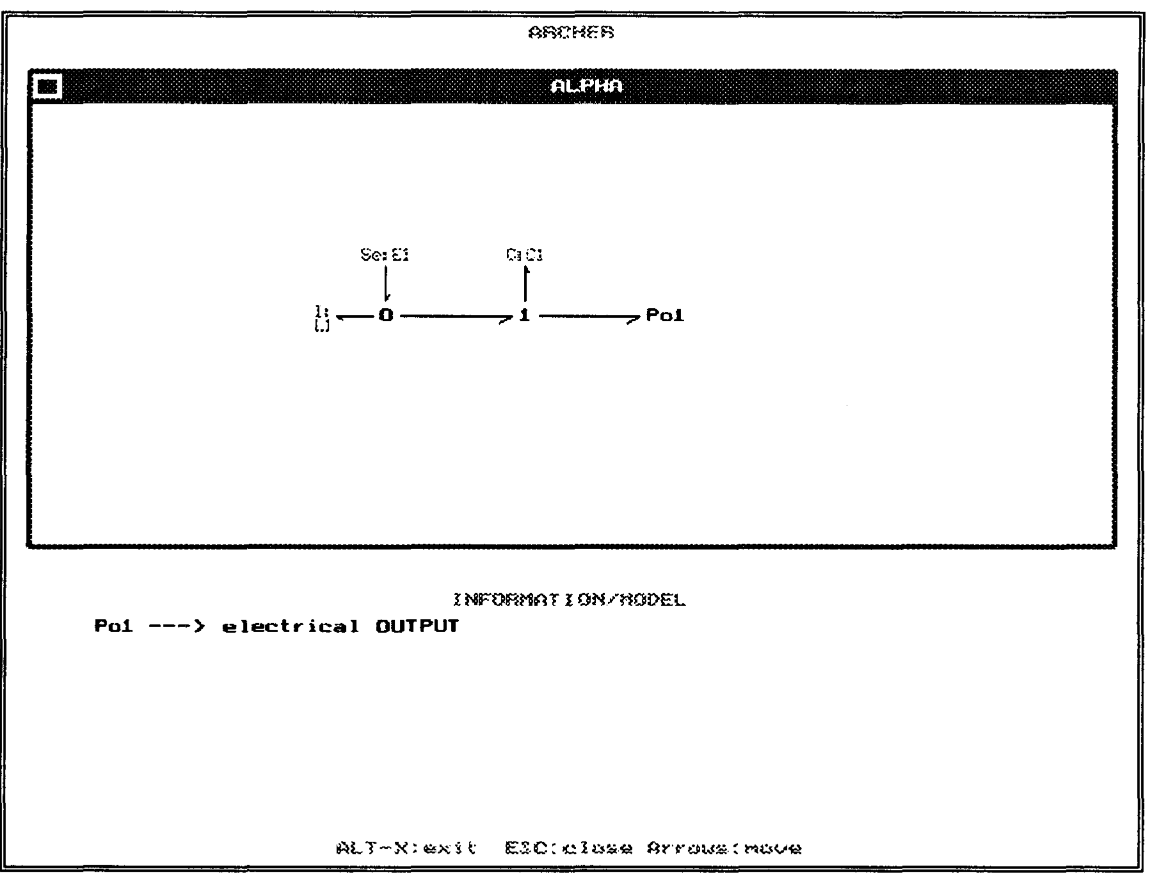
jonction(11B3 , [X3])	X3 est l'élément X1 du bloc B3 ou GAMMA1
jonction(01B3 , [Y3 Z3])	Y3 ----------------Y1 ------------------------------
jonction(TF1B3 , [ ])	Z3 ----------------Z1 ------------------------------
lien(TF1B3 , 01B3)	
lien(11B3 , 01B3)	

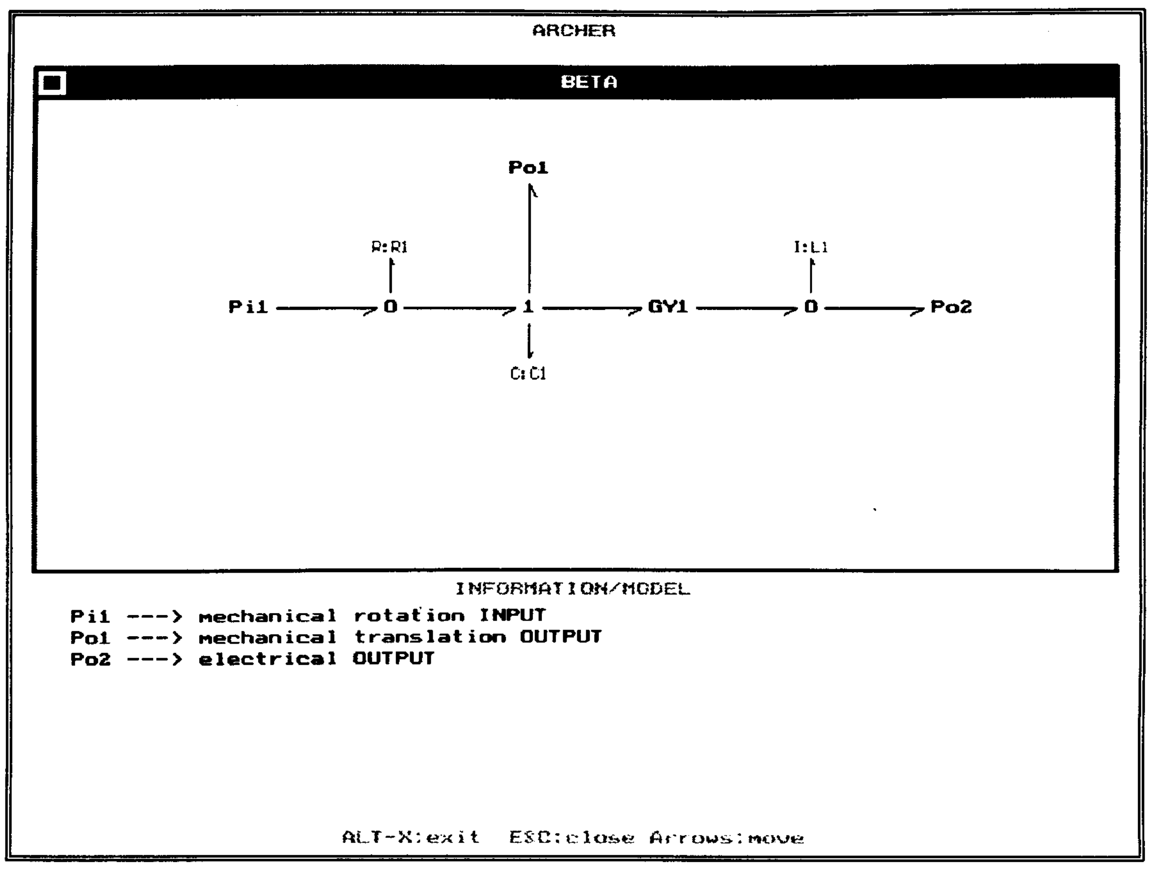
BLOC B4 ou GAMMA2

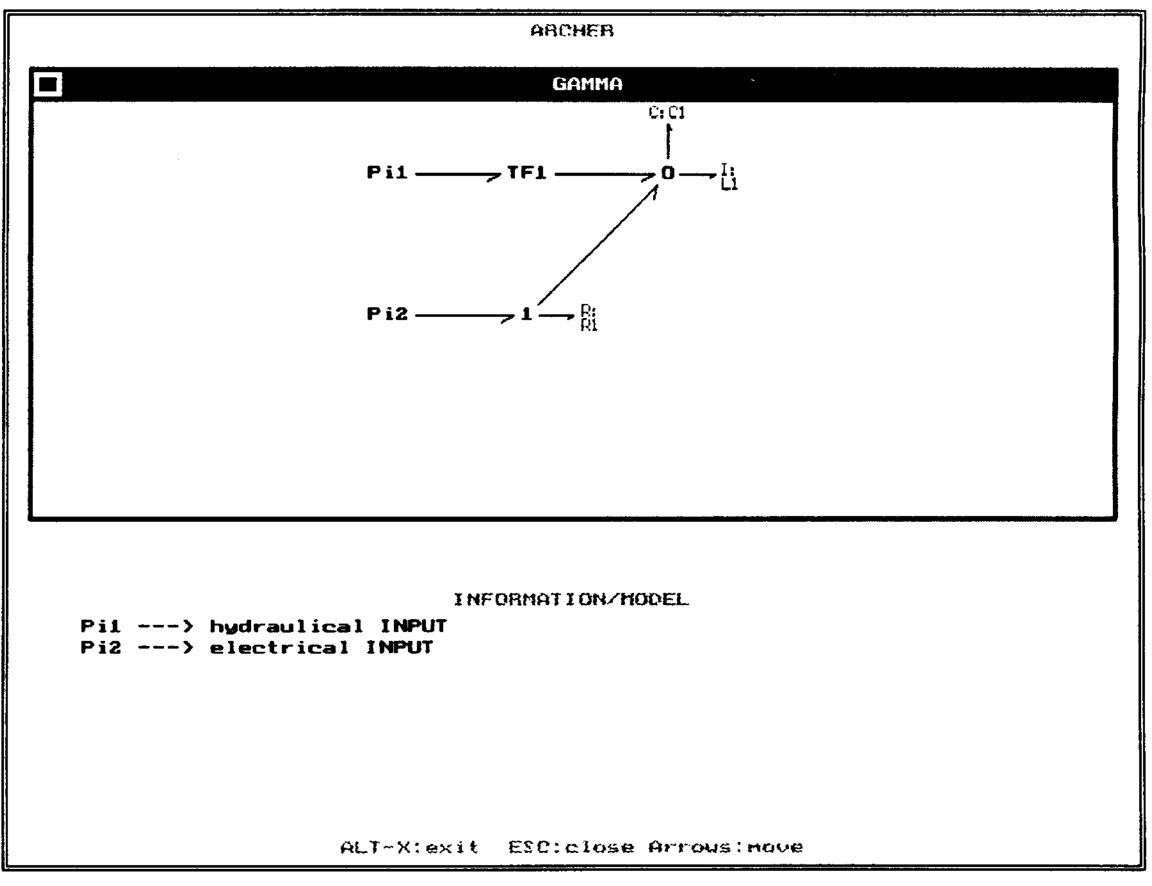
jonction(11B4 , [X4])	X4 est l'élément X1 du bloc B4 ou GAMMA2
jonction(01B4 , [Y4 Z4])	Y4 ----------------Y1 ------------------------------
jonction(TF1B4 , [ ])	Z4 ----------------Z1 ------------------------------
lien(TF1B4 , 01B4)	
lien(11B4 , 01B4)	

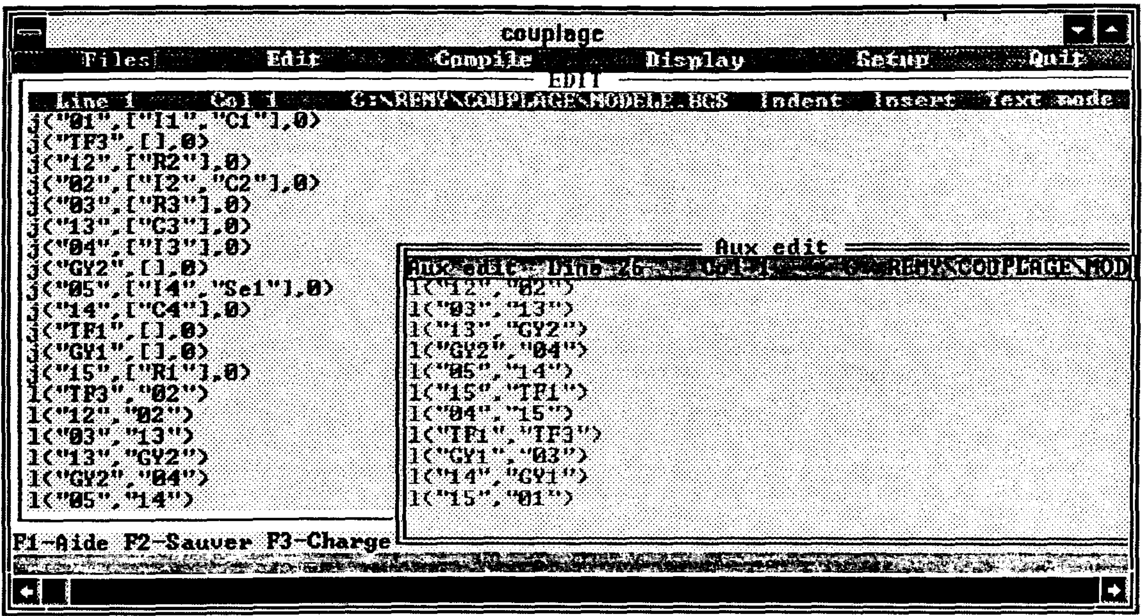
Code intermédiaire	Code bond-gaph des blocs renumérotés	
jonction(01 , [ ])	*jonction(01B1 , [X1])*	*jonction(11B3 , [X3])*
jonction(11 , [ ])	*jonction(11B1 , [Y 1])*	*jonction(01B3 , [Y3 Z3])*
jonction(GY1, [ ])	*jonction(12B1 , [Z1])*	*jonction(TF1B3 , [ ])*
jonction(TF1 , [ ])	*lien(01B1 , 11B1)*	*lien(TF1B3 , 01B3)*
jonction(TF2 , [ ])	*lien(01B1 , 12B1)*	*lien(11B3 , 01B3)*
lien(11B1,01)	*jonction(01B2 , [X2])*	*jonction(11B4 , [X4])*
lien(01,GY1)	*jonction(11B2 , [Y 2])*	*jonction(01B4 , [Y4 Z4])*
lien(GY1,01B2)	*jonction(GY1B2 , [ ])*	*jonction(TF1B4 , [ ])*
lien(02B2,11)	*jonction(02B2 , [Z2])*	*lien(TF1B4 , 01B4)*
lien(11,TF1)	*lien(01B2 , 11B2)*	*lien(11B4 , 01B4)*
lien(11,TF2)	*lien(11B2 , GY1B2n)*	
lien(TF1,TF1B3)	*lien(GY1B2 , 02B2)*	
lien(TF2,11B4)		

En pratique, nous allons prendre des blocs ALPHA, BETA et GAMMA en remplacant les éléments X, Y et Z par des éléments bond-graphs comme le montre les trois figures suivantes :




ARCHER
BETA
Po1
R:R1
I:L1
Pi1
0
1
GY1
0
Po2
C:C1
INFORMATION/MODEL
Pi1 ---> mechanical rotation INPUT
Po1 ---> mechanical translation OUTPUT
Po2 ---> electrical OUTPUT
ALT-X:exit ESC:close Arrows:move


ARCHER
GAMMA
C:C1
Pi1
TF1
0
I:
L1
Pi2
1
R:
R1
INFORMATION/MODEL
Pi1 ---> hydraulical INPUT
Pi2 ---> electrical INPUT
ALT-X:exit ESC:close Arrows:move



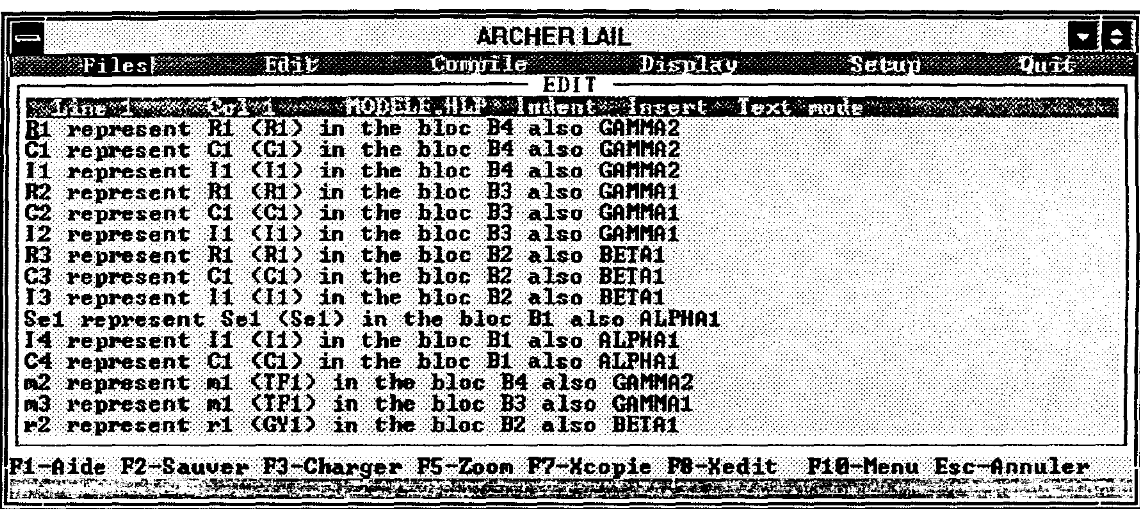


fig A.4.3 : Code bond-graph simplifié (*.bgs) et fichier de descendance (*.hlp).

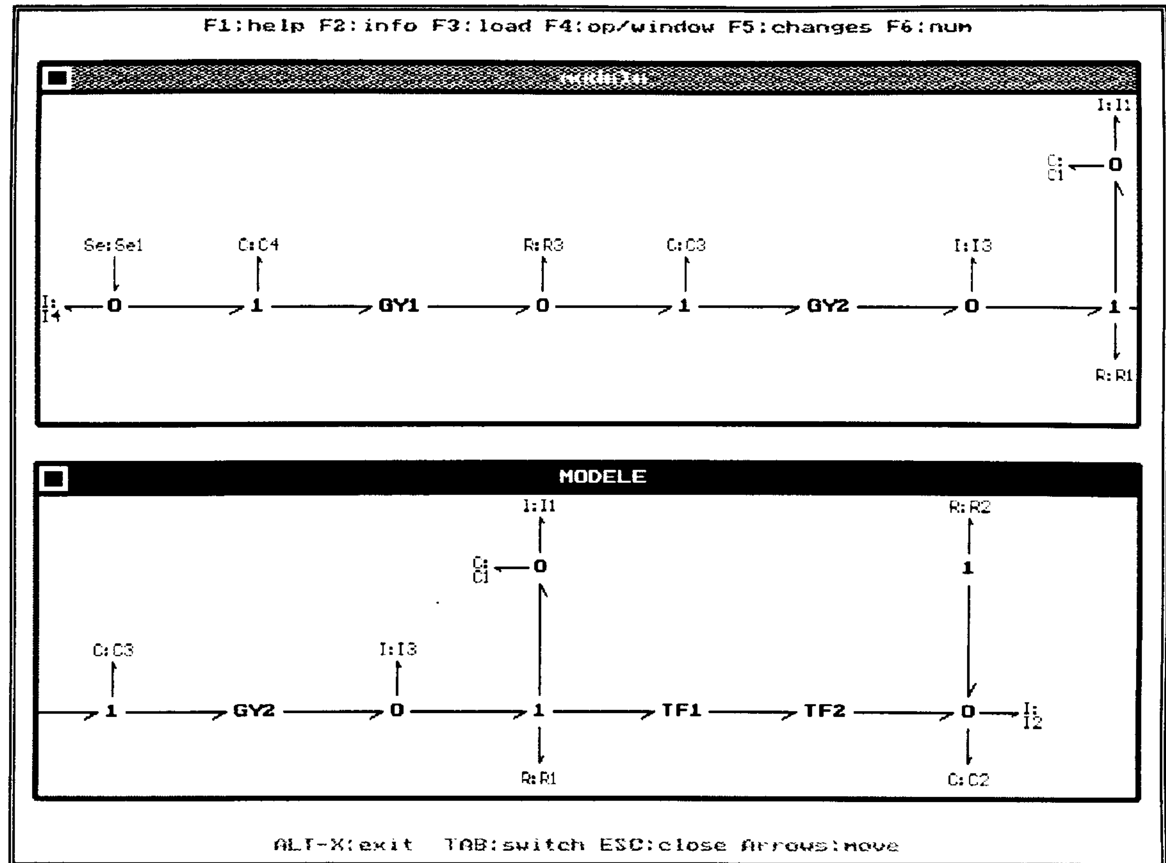


fig A.4.4 : Dessin du bond-graph simplifié.

ANNEXE 5 : BIBILIOTHEQUE DE BLOC PRE-DEFINIS POUR L'APPLICATION DU CHAPITRE IV

BLOC	DESCRIPTION	CODE BOND-GRAPH *. bg	*. blk
G	**jonction** **11 Sel** **port** **11 out1 : e**	*jonction (11 , [Sel])*	*out (1 , 11 , e)*
MR	**port** **GY1 in1 : e out1 : mr**	*jonction (GY1 , [ ])*	*in (1 , GY1 , e)*
PDP	**lien** **TF1 11** **port** **TF1 in1 : mr** **11 in2 : h out1 : h**	*jonction (TF1 , [ ])* *jonction (11 , [ ])* *lien (TF1 , 11)*	*in (1 , TF1 , mr)* *in (2 , 11 , h)* *out (1 , 11 , h)*
L	**jonction** **01 C1** **11 R1** **lien** **01 11** **port** **01 in1 : h** **11 out1 : h**	*jonction (01 , [C1])* *jonction (11 , [R1])* *lien (01 , 11)*	*in (1 , 01 , h)* *out (1 , 11 , h)*

BLOC	DESCRIPTION	CODE BOND-GRAPH *. bg	*. blk
V	**jonction** **11 R1** **12 R2** **lien** **01 12** **01 11** **port** **01 in1 : h** **11 out2 : h** **12 out1 : h**	*jonction (11 , [R1])* *jonction (12 , [R2])* *lien (01 , 12)* *lien (01 , 11)*	*in (1 , 01 , h)* *out (1 , 12 , h)* *out (2 , 11 , h)*
MH	**lien** **11 TF1** **port** **11 in1 : h out1 : h** **TF1 out2 : mr**	*jonction (11 , [ ])* *jonction (TF1 , [ ])* *lien (11 , TF1)*	*in (1 , 11 , h)* *out (1 , 11 , h)* *out (2 , TF1 , mr)*
F	**jonction** **11 R1** **port** **11 in1 : h out1 : h**	*jonction (11 , [R1])*	*in (1 , 11 , h)* *out (1 , 11 , h)*
C	**port** **TF1 in1 : mr out1 : mt**	*jonction (TF1 , [ ])*	*in (1 , TF1 , mr)* *out (1 , TF1 , mt)*
T	**jonction** **11 R1 I1** **port** **11 in1 : mt**	*jonction (11 , [R1 , I1])*	*in (1 , 11 , mt)*

REFERENCES BIBLIOGRAPHIQUES

REFERENCES BIBLIOGRAPHIQUES

AZMANI A. (1991)
Analyse symbolique du bond-graph par une approche Intelligence Artificielle
Contribution à la réalisation d'un processeur d'aide à la modélisation
Thèse de Doctorat, Université des Sciences et Techniques de Lille Flandre Artois (France).

AZMANI A., BOUAYAD R., DAUPHIN-TANGUY G. (1991)
Artificial Intelligence approach for the causal analysis of bond-graph models
Mathematical and intelligent models in system simulation
R. Hanus, P. Kool, S. Tzafestas (editors)
J.C. Baltzer AG, Scientific Publishing Co. IMACS, 1991
pp. 319-324.

AZMANI A., DAUPHIN-TANGUY G. (1992)
Archer : a Program for computer Aided Modelling and Analysis
Papier Invité, Bond Graphs for Engineers, Proc. of the 13th IMACS World Congress, Dublin(Irlande), Juillet 1991, and IMACS Conf. on "Modelling and Control of Technological Systems", Lille, Mai 1991, G. Dauphin-Tanguy & P.C. Breedveld Ed., Elsevier Sc. Pub., 1992.

BIDARD C., FAVRET F., GOLDZTEJN & LARIVIERE E. (1993)
Bond-graph and variable causality
IEEE-International Conference on "Systems, Man and Cybernetics, Systems Engineering in the Service of Humans", Le Touquet (France)
Vol 1, pp. 270-275, October 1993.

BIRKETT S.H. (1990)
Combinatorial analysis of dynamical systems
Thèse de PHD, Université de Waterloo (Canada).

BORLAND (1988)
Turbo Prolog 2.0. Reference guide
Turbo Prolog 2.0. User's guide
Borland.

BORNE P., DAUPHIN-TANGUY G., RICHARD J.P., ROTELLA F., ZAMBETTAKIS I. (1992)
Modélisation et identification des processus, tome 1, tome 2
Editions Technip.

BOUAYAD R., AZMANI A., DAUPHIN-TANGUY G. (1991)
Algorithm for the computer aided drawing of bond-graph models
Mathematical and intelligent models in system simulation
R. Hanus, P. Kool, S. Tzafestas (editors)
J.C. Baltzer AG, Scientific Publishing Co. IMACS
pp. 201-205.

BRESCH D. (1986)
Etude et réalisation d'un outil logiciel de modélisation et de simulation de systèmes physiques utilisant la représentation des graphes à liens
Thèse de Doctorat, Université de Haute Alsace (France).

BREEDVELD P.C. (1984)
Physical systems theory in terms of bond-graphs
PhD thesis, University of Twente, Enschede, Netherlands.

BROENINK J.F. (1990)
Computer-aided physical-systems modelling and simulation : a bond-graph approach
Thèse de PHD, Université de Enschede (Pays-Bas).

CONDILLAC M. (1986)
Prolog : fondements et applications
Edition DUNOD informatique.

COTTINI M. (1989)
PC et compatibles. Le bios mis à nu
Micropassion collection.

COUTEREEL L., BOUAYAD R., DAUPHIN-TANGUY G. (1989)
Artificial Intelligence and Bond-Graph methodology for the modelling of dynamical systems
Modelling and simulation of systems
P.C. Breedveld et al. (editors)
J.C. Baltzer AG, Scientific Publishing Co. IMACS
pp. 21-23.

CUELLAR G. (1987)
Graphismes sur IBM PC/ XT/ Compatibles
Eyrolles.

DAUPHIN-TANGUY G., SUEUR C., COUTEREEL L. (1987)
Presentation of a processor for the computer aided modelling of dynamicals systems through a Bond-Graph approach
IMACS-Intenational Symposium on "Artificial Intelligence, Expert Systems and Language in Modelling and Simulation" Barcelone (Espagne)
pp. 197-202.

DAUPHIN-TANGUY G., ROMBAUT C. (1993)
Why a unique causality in the elèmentary commutation cell bond-graph model of a power electronics converter
IEEE-International Conference on "Systems, Man and Cybernetics, Systems Engineering in the Service of Humans", Le Touquet (France)
Vol 1, pp. 257-263, October 1993.

DECLERCK P. (1991)
Analyse structurale et fonctionnelle des grands systèmes, application à une centrale PWR 900 MW
Thèse de Doctorat, 21 décembre 1991, Université des Sciences et Techniques de Lille Flandres Artois.

DELAHAYE J.P. (1986)
Outils logiques pour l'Intelligence Artificielle
Collection de la Direction des Etudes et Recherches d'Eléctricité de France
Eyrolles.

DELAHAYE J.P. (1987)
SYSTEMES EXPERTS : organisation et programmation des bases de connaissance en calcul propositionnel
Eyrolles.

DELGADO DE NIETO M. (1991)
Description et simulation de systèmes linéaires représentés par le bond-graph
Thèse de Doctorat, Université de Rennes (France).

FARRENY H. (1985)
LES SYSTEMES EXPERTS principes et exemples
Cepadues Editions.

GENTHON P. (1989)
Dictionnaire de l'intelligence artificielle
Hermes.

GERTLER J.J. (1988)
Survey of model-based failure detection and isolation in complex plants.
IEEE Trans. on A.C. december 1988.

GIANNESINI F., KANOUI H., PASERO R., VAN CANEGHEM M. (1985)
Prolog
Inter Editions

GONDRAN M., MINOUX M. (1979)
Graphes et algorithmes
Collection de la Direction des Etudes et Recherches d'Eléctricité de France
Eyrolles.

HASSENFORDER M., LIENHARDT D., GISSINGER G. (1991)
PROUESSE : A software tool for simulation and analysis of physical systems, specially designed for automatic applications
IMACS 91, 13th World Congress on Computation and Applied Mathematics, Dublin
vol. 3, pp. 1296-1299.

JAYEZ J.-H. (1982)
Compréhension automatique du langage naturel, le cas du groupe nominal en français
Méthode + Programmes, Masson 1982.

KAMEL A. (1994)
Etude des transferts de puissance dans les systèmes physiques par l'approche Bond-Graph et le formalisme Scattering
Thèse de Docteur d'Université de Lille I, Février 1994.

KARNOPP D.C., ROSENBERG R.C. (1975)
Systems dynamics : an unified approach
Wiley and Sons, New-York.

KRON G. (1963)
The Piecewise Solution of Large-scale Systems
Macdonald & Co., Ltd., London.

KUBIAK P. (1992)
Réalisation d'un Module de Couplage ARCHER / Logiciels de simulation
DEA de Productique, Université des sciences et techniques de Lille Flandres Artois.

LIND M. (1982)
Multilevel flow of process plant for diagnosis and control.
Riso national laboratory, DK 4000 Roskilde, Denmark, august 82.

LISSANDRE M. (1986)
La méthode SADT
Génie Logiciel, n°4, pp. 58-62, 1986.

NOYELLE Y. (1988)
Traitement des langages évolués, compilation, interprétation, support d'éxécution.
Masson Editions.

MONTBRUN-DI FILLIPO J., DELGADO M., BRIE C., PAYNTER H.M. (1991)
A Survey of BondGraph : Theory, Applications and Programs
Journal of Franklin Institute.

ORT J.R., MARTENS M.R. (1974)
A topological procedure for converting a bond-graph to a linear graph
J. Dynamical Systems, Measurement and Control
pp. 307-314.

PAYNTER M.M. (1961)
Analysis and Design of Engineering Systems
M.I.T. Press, Cambridge (Mass).

PERELSON A., OSTER G.F. (1976)
Bond-graphs and linear graphs
J. Franklin Institute
vol. 302, n°2, pp. 159-185.

RAHMANI A. (1993)
Etude Structurelle des Systèmes Linéaires par l'Approche Bond-Graph
Thèse de Doctorat d'Université de Lille I, Octobre 1993.

RASMUSSEN J. (1985)
The role of hierarchical knowledge representation in decision making and system management.
IEEE Transactions on systems, man, and cybernetics,
vol. SMC-15, n°2, march/april 85.

REINSCHKE K.J. (1988)
Multivariable control. A graph-theoretic approach
Lecture Notes in Control and Information Sciences vol. 108,
Spring Verlag.

REMY P. (1990)
Elaboration d'un module de couplage entre les modèles bond-graphs pour le processeur ARCHER
DEA de Productique, Université des sciences et techniques de Lille Flandres Artois.

REMY P., AZMANI A., DAUPHIN-TANGUY G. (1992)
Archer's Compiler for the Generation of Bond-graph Models from Different Descriptions of Physical Systems
ESS 92 Simulation and AI in Computer Aided Tecniques,
november 5-8 Dresden,, pp. 49-53.

RINGOT D. (1990)
Analyseur syntaxique et sémantique d'un langage utilisateur pour la construction des bond-graph
DEA de productique,Université des Sciences et Techniques de Lille Flandres Artois.

ROSENBERG R.C., ANDRY A.N (1979)
Solvability of Bond-Graph Junction Structures with Loops
IEEE Trans. on Circuits and Systems, Vol Cas-26, n°2, pp. 130-137.

ROSENBERG R.C., KARNOPP D.C. (1983)
Introduction to physical system dynamics
Mc Graw-Hill. Book compagny.

ROSS D.T. (1977)
Applications and extensions of SADT,
IEEE Trans. Software Engineering, Vol. SE-3, n°1, jan.1977, pp. 6-15.

SUEUR C. (1990)
Contribution à la modélisation et à l'analyse des systèmes dynamiques par une approche bond-graph : application aux systèmes polyarticulés, plan à segments flexible
Thèse, Université des Sciences et Techniques de Lille (FRANCE).

SUEUR C., DAUPHIN-TANGUY G. (1989)
Stuctural Contrability/Observability of Linear Systems Represented by bond-graph
Journal of the Franklin Institute, vol 326, n°6, pp. 869-883.

SUEUR C., DAUPHIN-TANGUY G. (1991)
Bond-Graph Approach for Structural Analysis of MIMO Linear Systems
Journal of Franklin Institute, vol. 328, n°1, pp. 55-70.

SUEUR C., DAUPHIN-TANGUY G. (1991)
Bond-Graph Approach to Multi-time Scale Systems Analysis
Journal of Franklin Institute, vol. 328, n°5/6, pp. 1005-1026.

STAROSWIECKI M. (1989)
Automatic Analytical Redundancy relationships generation in complex interconnected system based on a structural approach
4th INCARF, New-Delhi (Inde), Décembre 1989.

TABORIN V. (1989)
Coopération entre opérateur et système d'aide à la décision pour la conduite de procédés continus : application à l'interface opérateur système expert du projet ALLIANCE.
Thèse de doctorat, Université de Valenciennes, mars 1989.

THOMA J. (1975)
Introduction to Bond-Graphs and their Applications
Pergamon Press.

WLASOWSKI M., LORENZ F. (1991)
How to Determine the Solvability of Bond-Graph Linear Junction Structures
Journal of the Franklin Institute vol. 328, n° 5/6, pp. 855-870,

ZWINGELSTEIN B., UPADHYAYA R. (1979)
Identification of multivariate models for noise analysis of nuclear plant
5th IFAC Symposium on identification and system parameter Estimation.
Darmstadt 1979,
Pergamon Press.