

EDF R&D

MANAGEMENT DES RISQUES INDUSTRIELS

ETUDES PROBABILISTES DE SURETE ET DE DISPONIBILITE DES SYSTEMES

7 boulevard Gaspard Monge 91120 PALAISEAU - +33 (1) 78 19 32 00

20/04/2017

Manuel de référence du langage de modélisation probabiliste Figaro

Marc BOUISSOU

HUMBERT Sybille

HOUDEBINE Jean-Christophe

EDF R&D - MRI

EDF R&D

Société Aristè

6125-1612-2017-00624-FR



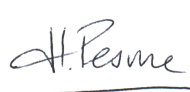
1.0

Type d'information : Note technique

Le langage Figaro est un langage spécialisé (Domain Specific Language) dédié à la modélisation des systèmes pour les études de sûreté de fonctionnement. Ce document explique les principes de modélisation en Figaro, l'existence de deux niveaux appelés ordre 1 et ordre 0, la sémantique du langage et la façon de l'utiliser pour construire des bases de connaissances permettant la construction rapide de modèles fiabilistes des systèmes discrets. Il est destiné aux personnes cherchant à se familiariser avec le langage Figaro et ses possibilités (chapitres 1 et 2) et aux personnes devant écrire ou modifier des bases de connaissances. Il illustre par de nombreux exemples la syntaxe du langage, définie formellement dans le document « Syntaxe du langage de modélisation probabiliste Figaro » et la sémantique associée. Il donne également une définition formelle de la sémantique du langage Figaro d'ordre 0 qui est le point de départ de tous les traitements. Une fois que le lecteur aura assimilé le contenu du présent document il pourra utiliser seulement le document plus court « Syntaxe du langage de modélisation probabiliste Figaro ».

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	6125-1612-2017-00624-FR Version 1.0
---------	--	--

Circuit de validation

Auteur	Marc BOUISSOU	27/03/2017	
Vérificateur	Thomas CHAUDONNERET	03/04/2017	
Approbateur	Hélène PESME	20/04/2017	

Code affaire	P11DD
---------------------	-------

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	6125-1612-2017-00624-FR Version 1.0
---------	--	--

Liste de diffusion

Groupe destinataire
1612-EPSDS

Pré-diff	Diff	Destinataire	Structure	E-mail
	X	BILLET Laurent	EDF R&D - DIR R&D	laurent.billet@edf.fr
	X	BOUISSOU Marc	EDF R&D - MRI	marc.bouissou@edf.fr
	X	BOUSKELA Daniel	EDF R&D - STEP	daniel.bouskela@edf.fr
	X	BUFFONI Lena	Linköping University	lena.buffoni@liu.se
	X	BUGAT Stephane	EDF R&D - SINETICS	stephane.bugat@edf.fr
	X	CARER Philippe	EDF R&D - MIRE	philippe.carer@edf.fr
X	X	CHAUDONNERET Thomas	EDF R&D - MIRE	thomas.chaudonneret@edf.fr
	X	DESPUJOLS Antoine	EDF R&D - MRI	antoine.despujols@edf.fr
	X	DONAT Roland	Société Edgemind	roland.donat@edgemind.net
	X	FAVENNEC Jean-melaine	EDF R&D - STEP	jean-melaine.favennec@edf.fr
	X	FRIEDLHUBER Thomas	Société Edgemind	thomas.friedlhuber@edgemind.net
	X	GUUINIC Philippe	EDF R&D - THEMIS	philippe.guunic@edf.fr
	X	HEBRAIL Georges	EDF R&D - ICAME	georges.hebrail@edf.fr
	X	HOUDEBINE Jean-Christophe	Société Aristè	ariste@wanadoo.fr
	X	HUMBERT Sybille	EDF R&D	
	X	IOOSS Bertrand	EDF R&D - MRI	bertrand.iooss@edf.fr
	X	JARDIN Audrey	EDF R&D - STEP	audrey.jardin@edf.fr
	X	LAAROUCI Youssef	EDF R&D - SINETICS	youssef.laarouchi@edf.fr
	X	LAIR William	EDF R&D - MIRE	william.lair@edf.fr
	X	LALEUF Jean-claude	EDF R&D - SINETICS	jean-claude.laleuf@edf.fr
	X	LUCAS Jean-yves	EDF R&D - OSIRIS	jean-yves.lucas@edf.fr
	X	MCDONALD John	EDF R&D - MIRE	john.mcdonald@edf.fr
	X	PELLET Laure	EDF R&D - SINETICS	laure.pellet@edf.fr
	X	PESME Helene	EDF R&D - MRI	helene.pesme@edf.fr
	X	SANCHEZ-TORRES Jose	EDF R&D - MIRE	jose.sanchez-torres@edf.fr
	X	THUY N	EDF R&D - STEP	n.thuy@edf.fr



MANUEL DE REFERENCE DU LANGAGE DE MODELISATION PROBABILISTE FIGARO

Marc Bouissou, Sybille Humbert, Jean-Christophe Houdebine

Version E - Mars 2017

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Synthèse

Ce document est destiné aux personnes cherchant à se familiariser avec le langage Figaro et ses possibilités (chapitres 1 et 2) et aux personnes devant écrire ou modifier des bases de connaissances. Il illustre par de nombreux exemples la syntaxe du langage, définie formellement dans le document associé « Syntaxe du langage de modélisation Figaro » et la sémantique associée. Il donne également une définition formelle de la sémantique du langage Figaro d'ordre 0 qui est le point de départ de tous les traitements.

Une fois que le lecteur aura assimilé le contenu du présent document il pourra utiliser seulement le document plus court « Syntaxe du langage de modélisation Figaro ».

- Le premier chapitre décrit les objectifs du langage et explique pourquoi ce langage a été développé.
- Le deuxième chapitre décrit les principes de modélisation des systèmes en Figaro. Il décrit notamment le principe d'exploitation des règles d'un modèle Figaro en chaînage avant et en chaînage arrière.
- Le troisième chapitre donne une définition formelle de la sémantique du langage, grâce à quelques notions mathématiques simples.
- Le quatrième chapitre décrit les éléments lexicaux du langage.
- Le cinquième chapitre décrit principalement la syntaxe des expressions utilisables en Figaro.
- Le sixième chapitre décrit de façon détaillée la syntaxe de définition des classes d'objets Figaro.
- Le septième chapitre décrit les possibilités du langage d'abord plus complexe.
- Le huitième chapitre détaille les possibilités de documentation des modèles
- Le neuvième chapitre s'intéresse à la détection automatique des incohérences dans les modèles et donne des conseils pour écrire des bases de connaissances dont la cohérence est assurée par construction.
- Enfin, en annexe se trouvent la liste des évolutions du langage depuis 1995 et un index des principales notions manipulées dans ce manuel.

Ce document constitue la cinquième version du manuel de référence du langage Figaro. Les modifications apportées depuis la troisième version comprennent entre autres :

- d'une part, les changements dus aux évolutions du langage nécessitées par l'application KB3 avec l'introduction de nouvelles informations associées aux pannes, et la notion nouvelle d'objet-système qui a permis de remplacer ce qui était écrit dans un pseudo-type appelé GLOBAL. En outre, des fonctions exploitables uniquement par simulation de Monte Carlo ont pu être ajoutées grâce à la création d'un nouvel outil de quantification reposant sur cette méthode, appelé YAMS.
- d'autre part l'ajout d'un chapitre donnant une définition formelle de la sémantique du langage et d'un autre décrivant divers mécanismes permettant d'éviter ou bien de détecter des incohérences dans les modèles. Ces chapitres reprennent la teneur de deux articles publiés au congrès ESREL en 2002.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Sommaire

SYNTHESE	3
SOMMAIRE	4
PRELIMINAIRES	7
1. INTRODUCTION	8
1.1. CONTEXTE ET HISTORIQUE	8
1.1.1. <i>Qu'est-ce qu'une étude de sûreté de fonctionnement ?</i>	8
1.1.2. <i>Vers l'automatisation</i>	8
1.1.3. <i>Démarche pour l'automatisation</i>	9
1.2. POURQUOI UN LANGAGE DE MODELISATION SPECIFIQUE ?	10
2. PRINCIPES DE MODELISATION EN FIGARO	11
2.1. LES CLASSES ET LES OBJETS	11
2.1.1. <i>Définitions</i>	12
2.1.2. <i>Structure d'un modèle Figaro</i>	12
2.2. MODELES D'ORDRE 1 ET D'ORDRE 0	13
2.3. L'HERITAGE ENTRE CLASSES	13
2.4. COMMUNICATION ENTRE LES OBJETS	15
2.4.1. <i>Champs interface d'un type</i>	15
2.4.2. <i>Champs interface d'un objet</i>	16
2.4.3. <i>Relations et ensembles</i>	16
2.4.3.1. <i>Identifiant d'une relation</i>	16
2.4.3.2. <i>Cardinalité</i>	16
2.4.4. <i>Utilisation des champs d'interface</i>	17
2.4.5. <i>Modélisation des liens matériels entre objets</i>	17
2.5. CHAMPS DE CLASSES ET D'OBJETS	17
2.6. LIEN ENTRE LES CONCEPTS FIGARO, UML ET SYSML	18
2.7. FIGARO ET LES UNITES	19
2.8. REGLES FIGARO	19
2.8.1. <i>Principe de modélisation des systèmes</i>	19
2.8.2. <i>Formalisme d'une règle Figaro</i>	20
2.8.2.1. <i>Forme d'une règle d'occurrence</i>	20
2.8.2.2. <i>Forme d'une règle d'interaction</i>	21
2.8.2.3. <i>Condition et actions d'une règle, utilisation des champs d'interface</i>	21
2.8.3. <i>Principes d'utilisation des règles</i>	23
2.8.3.1. <i>Application en chaînage avant</i>	23
2.8.3.2. <i>Application en chaînage arrière</i>	28
3. DEFINITION FORMELLE DU LANGAGE FIGARO 0	30
3.1. ELEMENTS DU MODELE	30
3.2. INFERENCE REALISEE DANS LES REGLES D'INTERACTION	31
3.3. SEMANTIQUE DU MODELE COMPLET	31
3.4. SEMANTIQUE D'UN MODELE AVEC DES LOIS PROBABILISTES QUELCONQUES	32
4. LES ELEMENTS LEXICAUX DU LANGAGE FIGARO	33
4.1. L'ALPHABET	33
4.2. LES VALEURS NUMERIQUES	35
4.3. LES IDENTIFICATEURS	35
4.4. LES MOTS-CLES	35
4.4.1. <i>Les opérateurs</i>	36
4.4.2. <i>Les mots-clés</i>	36
5. LES EXPRESSIONS	37
5.1. PRELIMINAIRES	37
5.1.1. <i>Champs valués et variables globales</i>	37
5.1.2. <i>Définir un ensemble</i>	37
5.1.2.1. <i>Notions d'ensembles et de variables liées</i>	37
5.1.2.2. <i>Utilisation d'un champ d'interface pour définir un ensemble</i>	38

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

5.1.2.3.	Utilisation d'un type pour définir un ensemble	39
5.1.2.4.	Limite de Figaro.....	40
5.1.2.5.	Restriction d'un ensemble.....	40
5.1.2.6.	Cas des ensembles vides	41
5.1.3.	Accès à un champ de type ou d'objet.....	42
5.1.3.1.	Accès aux champs du type décrit.....	42
5.1.3.2.	Accès aux champs d'une variable liée à un ensemble d'objets.....	42
5.1.3.3.	Accès au champ d'un objet défini par un ensemble de cardinal 1	43
5.1.3.4.	Accès aux champs d'un objet nommé.....	43
5.1.3.5.	Accès aux variables globales	43
5.2.	EXPRESSIONS SIMPLES	43
5.2.1.	Opérations arithmétiques	44
5.2.2.	Opérations booléennes	45
5.2.3.	Priorités entre les opérateurs	46
5.3.	LES QUANTIFICATEURS ET « ITERATEURS »	47
5.3.1.	Le quantificateur IL_EXISTE.....	47
5.3.2.	Le quantificateur QQSOIT.....	48
5.3.3.	Le quantificateur IL_EXISTE AU_MOINS	49
5.3.4.	Combinaison de quantificateurs.....	49
5.3.5.	L'itérateur POUR_TOUT	50
5.3.6.	L'itérateur SOIT.....	50
5.3.7.	Base de connaissances complète avec quantificateurs	50
5.4.	LES OPERATEURS BOOLEENS.....	51
5.4.1.	L'opérateur INCLUS DANS.....	51
5.4.2.	L'opérateur MARCHE.....	52
5.4.3.	L'opérateur PANNE	53
5.4.4.	L'opérateur AU_MOINS ... PARMI	53
5.5.	OPERATEURS NUMERIQUES COMPLEXES	54
5.5.1.	L'opérateur CARDINAL	54
5.5.2.	L'opérateur SOMME.....	54
5.5.3.	L'opérateur PRODUIT	55
5.5.4.	Les opérateurs MAXIMUM et MINIMUM.....	55
5.6.	OPERATEURS SPECIFIQUES	56
5.7.	L'OPERATEUR ?=.....	56
5.8.	PORTEE D'UNE VARIABLE	57
6.	DESCRIPTION D'UN TYPE FIGARO	58
6.1.	STRUCTURE GENERALE D'UNE DEFINITION DE CLASSE.....	58
6.1.1.	Déclaration d'une classe	58
6.1.2.	Structure générale de description d'une classe	58
6.2.	LES CHAMPS VALUES.....	59
6.2.1.	Préliminaires.....	59
6.2.1.1.	Etat initial d'un modèle Figaro	59
6.2.1.2.	Valeur par défaut.....	60
6.2.1.3.	Notion d'édition	61
6.2.1.4.	Remarque sur le calcul de l'état initial d'un modèle Figaro.....	62
6.2.2.	Les interfaces.....	62
6.2.3.	Les constantes.....	63
6.2.3.1.	CONSTANTE.....	63
6.2.3.2.	PARAMETRE_LOI	64
6.2.4.	Les variables d'état.....	65
6.2.4.1.	Notion de variable réinitialisée.....	65
6.2.4.1.1.	Variables essentielles	66
6.2.4.1.2.	Variables réinitialisées.....	66
6.2.4.2.	Catégories de variables d'état en Figaro	66
6.2.4.3.	ATTRIBUT	66
6.2.4.4.	PANNE.....	67
6.2.4.5.	EFFET.....	69
6.3.	LES REGLES.....	70
6.3.1.	Les conditions	70
6.3.2.	Les actions	71
6.3.2.1.	Affectation d'une valeur à une variable	71
6.3.2.2.	Action sur un ensemble d'objets.....	72

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

6.3.2.3.	Résolution de système d'équations.....	73
6.3.3.	<i>Les règles d'occurrence</i>	73
6.3.3.1.	Syntaxe d'une règle d'occurrence	75
6.3.3.2.	Exemple : transitions normales.....	78
6.3.3.3.	Défaillances	78
6.3.3.4.	Indisponibilités	80
6.3.3.5.	Réparations.....	80
6.3.4.	<i>Les règles d'interaction</i>	81
7.	NOTIONS AVANCEES.....	85
7.1.	LA PRE-COMPILEATION	85
7.1.1.	<i>Inclusion de fichiers</i>	85
7.1.2.	<i>Définition de macros</i>	85
7.1.3.	<i>Définition de variantes d'une base de connaissances</i>	86
7.2.	ELEMENTS GLOBAUX	87
7.3.	HERITAGE ET SURCHARGE	87
7.4.	LES GROUPES DE REGLES.....	88
7.5.	LES ETAPES.....	89
7.5.1.	<i>Déclaration des étapes</i>	90
7.5.2.	<i>Regroupement des règles par étapes</i>	91
7.5.3.	<i>Application des règles d'interaction en chaînage avant</i>	91
7.5.4.	<i>Application des règles d'interaction en chaînage arrière</i>	91
7.6.	LES EQUATIONS	91
7.6.1.	<i>Systèmes d'équations</i>	92
7.6.2.	<i>Equations</i>	92
8.	DOCUMENTATION DES MODELES FIGARO	93
9.	COHERENCE DES MODELES.....	94
9.1.	RELATION ENTRE INCOHERENCE A L'ORDRE 1 ET A L'ORDRE 0	94
9.1.1.	<i>Contrôles syntaxiques</i>	95
9.1.2.	<i>Cohérence du comportement</i>	95
9.2.	TPOLOGIE DES INCOHERENCES POSSIBLES.....	95
9.3.	GRAPHE DE DEPENDANCES ENTRE VARIABLES.....	97
9.4.	METHODES SURES D'ECRITURE DE BASES DE CONNAISSANCES	97
9.4.1.	<i>Avoir un graphe de dépendances pyramidal</i>	97
9.4.2.	<i>Raisonner en inférence monotone</i>	99
9.4.3.	<i>Bien exploiter les étapes</i>	99
9.5.	SEQUENCEMENT AUTOMATIQUE DES REGLES D'INTERACTION	100
9.5.1.	<i>Définition des interdépendances entre règles</i>	100
9.5.2.	<i>Ordonnancement des règles</i>	100
9.5.3.	<i>Utilisation de l'outil de séquençement des règles</i>	100
9.6.	DETECTION D'INCOHERENCES SUR UN MODELE FIGARO 0	101
9.7.	CONCLUSION SUR LA COHERENCE DES MODELES.....	102
10.	REFERENCES.....	103
11.	ANNEXE 1 : EVOLUTIONS DU LANGAGE FIGARO	104
11.1.	AJOUTS POUR LA SIMULATION DE MONTE CARLO.....	104
11.2.	AJOUTS POUR LA GENERATION D'ARBRES DE DEFAILLANCES	104
11.1.	REMPLACEMENT DU PSEUDO-TYPE GLOBAL PAR LES OBJETS SYSTEME.....	104
12.	ANNEXE 2 : MODELES DE FIABILITE ASSOCIABLES A UNE PANNE.....	105
13.	ANNEXE 3 : INDEX.....	108

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Préliminaires

Ce document est destiné aux personnes cherchant à se familiariser avec le langage Figaro et ses possibilités (chapitres 1 et 2) et aux personnes devant écrire ou modifier des bases de connaissances. Il décrit de façon détaillée la syntaxe du langage et la sémantique associée.

La seconde version du manuel (indice B) intégrait le retour d'expérience principalement issu de projets opérationnels, KB3 et TOPASE, et de l'enseignement de Figaro à l'Université de Rouen. Par ailleurs, elle tenait compte de la clarification de certains concepts du langage Figaro faite suite à la réécriture en C++ des traitements d'instanciation et de vérification de bases de connaissances.

La troisième version du manuel (indice C) décrivait le langage tel qu'il a été stabilisé lors du développement de la version 3 de KB3, avant 2005.

Cette cinquième version (indice E) est relative à l'état du langage en 2016 ; elle a fait l'objet de remaniements et ajouts importants par rapport à l'indice C (cf. la synthèse).

Conventions utilisées dans ce manuel

Ce manuel contient des descriptions de syntaxe pour les différentes expressions du langage Figaro. Cette notation utilise les méta-caractères suivants :

Méta-caractère	Description	Exemple
[]	Signifie que l'information entre les crochets est optionnelle.	[DE_TYPE type]
	Est utilisé pour exprimer une alternative.	UN UNE

Ce manuel utilise les conventions typographiques suivantes :

Machine à écrire

Indique que les lignes correspondantes font partie d'un programme Figaro.

Machine à écrire gras

Permet de faire la différence entre des identificateurs de l'utilisateur et les mots clés du langage dans les exemples. Les mots-clés seront toujours en gras.

Italique

Indique que le mot fait référence à un champ d'objet ou à un type d'un exemple.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

1. Introduction

Le langage Figaro est un langage spécialisé (Domain Specific Language) créé en 1989 permettant de modéliser les systèmes pour réaliser des études de sûreté de fonctionnement et, par extension, des calculs probabilistes sur des modèles stochastiques discrets à temps continu. Le but de ce chapitre est de présenter rapidement le domaine d'application du langage et ses principaux objectifs.

1.1. Contexte et historique

1.1.1. Qu'est-ce qu'une étude de sûreté de fonctionnement ?

L'objectif d'une étude de sûreté de fonctionnement d'un système est d'évaluer la fiabilité, la disponibilité ou la productivité d'un système [1], [3]. De façon classique, une telle étude se déroule en 3 phases.

Dans une première phase, le fiabiliste en charge de l'étude doit **modéliser le système** à étudier, c'est-à-dire élaborer une représentation du système qui permette de répondre aux objectifs de l'étude. L'activité de modélisation consiste principalement à décrire le système (topologie, composants, limites du système) et à formaliser les hypothèses qui sont utilisées pour l'étude (à quel niveau de détail sont décrits les composants, quels sont les modes de défaillance retenus pour chaque composant, quelle modélisation est adoptée pour décrire des comportements comme la propagation de fluide ou le flux électrique).

Dans une deuxième phase, le fiabiliste **construit les modèles de fiabilité** appropriés à l'étude du système, en utilisant le modèle et les hypothèses élaborés dans la première phase. Il existe plusieurs types de modèles de fiabilité utilisables [1], [3] (Arbre de défaillances, Graphe de Markov, Réseau de Petri, BDMP ...). Chacun de ces modèles permet de réaliser un certain type d'analyse. Par exemple :

- Une modélisation par Arbre de défaillances permet de représenter de façon très détaillée la logique d'un système, mais ne peut pas prendre en compte les dépendances entre composants ;
- Une modélisation par Graphe de Markov permet de prendre en compte les dépendances entre composants, mais est limitée par l'explosion combinatoire du nombre d'états à considérer.

Chaque type de modèle a donc ses limites et est associé à un certain nombre d'hypothèses à respecter. Par exemple, une modélisation par Arbre de défaillances suppose l'indépendance des composants du système. Le choix du type de modèle de fiabilité utilisé pour l'étude d'un système particulier est de la responsabilité du fiabiliste en charge de l'étude.

Enfin, dans une dernière phase, les modèles de fiabilité construits sont **quantifiés par des codes de calcul** appropriés afin d'obtenir les résultats attendus concernant la fiabilité, disponibilité ou productivité du système.

La réalisation d'une étude de sûreté de fonctionnement d'un système est donc une activité pluridisciplinaire, qui fait à la fois appel à des connaissances sur le fonctionnement du système et à des connaissances fiabilistes sur la modélisation des systèmes et le contenu mathématique des méthodes de traitement des modèles.

De façon classique, les activités de modélisation du système et d'élaboration des modèles de fiabilité sont réalisées "manuellement" par le fiabiliste. Les résultats de l'étude sont consignés dans un document décrivant :

- la modélisation du système réalisée (qui est décrite de façon plus ou moins détaillée) ;
- les modèles de fiabilité construits ;
- les résultats obtenus.

1.1.2. Vers l'automatisation

Depuis 1986, EDF a dû mener à bien plusieurs Etudes Probabilistes de Sûreté (EPS) de centrales nucléaires, chaque EPS nécessitant la réalisation d'un grand nombre d'études de sûreté de fonctionnement de systèmes élémentaires. Par ailleurs, depuis 1995, des demandes émergent à EDF pour introduire les techniques de sûreté de fonctionnement dans le processus de conception des systèmes.

Les travaux réalisés dans le cadre de ces projets ont montré les limites de l'approche manuelle classique pour réaliser les études de sûreté de fonctionnement des systèmes.

Manque de traçabilité et de transparence des études

Le raisonnement mené par le fiabiliste pour construire un modèle de fiabilité contient souvent beaucoup

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

d'implicite. Ce manque de transparence entraîne deux types de problèmes.

- Difficulté pour mettre à jour les études
Il est souvent difficile, pour celui qui n'a pas réalisé l'étude, de se réapproprier les modèles de fiabilité pour les faire évoluer. Ce problème se pose notamment dans le cadre des EPS pour lesquelles les modèles construits doivent évoluer dans le temps.
- Difficulté pour diffuser les études à des non fiabilistes
Comme nous l'avons déjà souligné, la réalisation d'une étude de fiabilité fait appel à des connaissances différentes, ce qui nécessite de faire appel à des interlocuteurs différents (concepteurs du système, exploitants). Il est important que ces interlocuteurs non fiabilistes puissent comprendre et valider les hypothèses et résultats sans pour autant comprendre le détail des modèles de fiabilité construits.

Difficulté pour garantir l'homogénéité des études au sein d'un même projet

Pour mener à bien des études probabilistes de grands systèmes complexes (EPS par exemple), il est nécessaire de réaliser et d'intégrer un grand nombre d'études de systèmes élémentaires. Ces études sont le plus souvent prises en charge par différents intervenants et il est difficile de garantir leur cohérence et leur homogénéité (choix du niveau de détail, méthode de prise en compte des modes de défaillances, ...).

Délais nécessaires aux études

La réalisation d'une étude de système "à la main" est un travail important, qui nécessite des délais conséquents. Ces délais rendent difficile l'intégration de telles études à des processus de conception de systèmes (processus itératif nécessitant de comparer "rapidement" des architectures de systèmes).

Ainsi, pour pallier les limites des études de sûreté de fonctionnement manuelles, des recherches ont été menées à EDF dès 1986 pour automatiser ces études. L'objectif de ces recherches était de mettre en place des outils permettant :

- de **modéliser** simplement les systèmes,
- de **formaliser, tracer et automatiser** le raisonnement du fiabiliste pour construire les modèles de fiabilité à partir de la modélisation d'un système particulier.

1.1.3. Démarche pour l'automatisation

Lorsqu'on réalise l'étude de sûreté de fonctionnement d'un système, on a besoin de collecter un certain nombre d'informations sur chaque composant du système. Ainsi, pour une pompe motorisée, on se demandera, par exemple, quel est le tableau électrique d'alimentation de la pompe, quel est l'état initial de la pompe (marche ou arrêt), etc.

Pour modéliser un composant, on lui associe donc (implicitement ou non) un certain nombre de propriétés. Il est clair que ces propriétés sont le plus souvent **communes à tous les composants d'un même type**. (Toutes les pompes motorisées des systèmes thermohydrauliques possèdent un tableau électrique d'alimentation et un état initial).

Ainsi, toutes les études de sûreté de fonctionnement relatives à une même classe de systèmes (systèmes thermohydrauliques, systèmes électriques, ...) peuvent utiliser un même **modèle générique** pour décrire les systèmes.

Il apparaît dès lors naturel, pour automatiser les études de sûreté de fonctionnement, de capitaliser les modèles génériques de composants, dans des **bases de connaissances** (cf. Figure 1). Par exemple, une base de connaissances dédiée aux systèmes thermohydrauliques contient la description des composants de type pompe, vanne, clapet, etc.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

*description de
classes de
composants*

*- description du
système
- configuration*

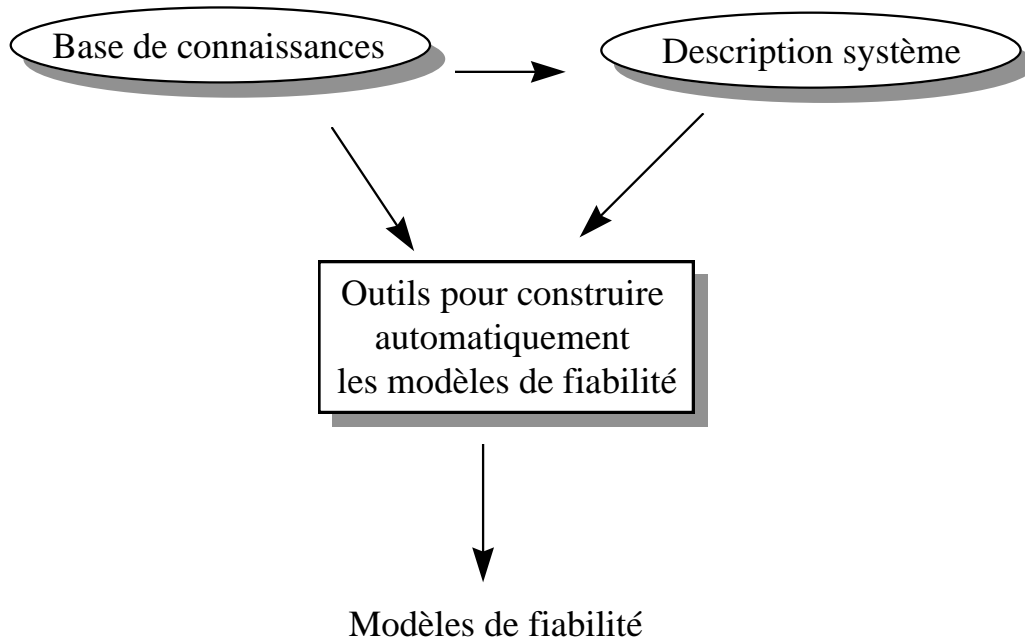


Figure 1. Démarche pour l'automatisation

Avec cette approche, la **description du système** à étudier (modélisation de la topologie du système, des composants contenus dans le système), est réalisée en s'appuyant sur les modèles génériques des composants de la base de connaissances, qu'il suffit d'instancier en fonction de leur emplacement dans la topologie du système et de leurs caractéristiques propres.

On génère ensuite automatiquement les modèles de fiabilité requis, en utilisant le **modèle du système**, constitué de la **base de connaissances** et de la **description du système** à étudier.

Remarque : La connaissance représentée dans une base de connaissances dépend bien sûr de la classe des systèmes à étudier mais aussi, le plus souvent, du type d'étude de sûreté de fonctionnement que l'on cherche à automatiser (du type de modèle de fiabilité que l'on veut générer automatiquement avec la base de connaissances).

1.2. Pourquoi un langage de modélisation spécifique ?

Pour mettre en œuvre cette démarche d'automatisation des études de sûreté de fonctionnement des systèmes, le premier problème consiste à trouver un **langage** permettant au fiabiliste de représenter simplement ses connaissances sur un système.

Ce langage doit répondre aux exigences suivantes :

- Les connaissances doivent être "**faciles**" à représenter pour le fiabiliste. Autrement dit, le langage choisi doit être simple à appréhender pour le fiabiliste. Il doit lui permettre de manipuler simplement les concepts nécessaires à la formalisation de son raisonnement (en intégrant le maximum de notions liées à la fiabilité).

- * Le langage doit permettre la description de connaissances génériques (Bases de connaissances) ;

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

- * Il doit fournir un formalisme simple pour la description de systèmes particuliers à partir de la modélisation générique de la base de connaissances. Ce formalisme doit pouvoir être associé à une interface graphique (description de la topologie et des composants du système) ;
 - * Il doit permettre de modéliser les systèmes avec le **même** formalisme, quels que soient les modèles de fiabilité à construire. Autrement dit, ce langage doit être une généralisation des modélisations fiabilistes courantes (Arbre de défaillances, Graphe de Markov, Réseau de Petri, ...).
- Les modèles construits avec ce langage (Bases de connaissances et descriptions de systèmes) doivent être **lisibles** par des non-spécialistes.
- Le formalisme utilisé doit être **évolutif** afin de permettre son extension à la modélisation de système pour d'autres types d'études que les études de sûreté de fonctionnement (maintenance, diagnostic).
- Il doit posséder une **définition formelle** afin de permettre la démonstration mathématique de propriétés : comportement du modèle défini de manière unique et indépendante des outils qui servent à le manipuler, assurance de l'absence d'incohérences, caractère fini de l'espace des états, garantie de possibilité de retour à l'état initial etc.
- Enfin, les modèles construits avec ce langage doivent être **maintenables**.
- * On doit pouvoir modifier facilement la connaissance contenue dans un modèle sans être obligé d'en avoir une vue d'ensemble, ce qui oriente vers un langage déclaratif ;
 - * Les connaissances doivent être **structurées** pour permettre l'accès rapide aux informations que l'on recherche et **factorisées** pour éviter la répétition d'informations identiques ;
 - * Le langage doit être associé à des outils de débogage.

L'analyse des langages informatiques existants et disponibles sur le marché montre qu'aucun de ces langages n'est adapté à l'ensemble de ces exigences. Les langages à vocation générale (ADA, C++, Java, Python ...) sont des langages de programmation et non de modélisation : ils ne satisfont pas directement aux contraintes de lisibilité des modèles et de simplicité d'appréhension par le fiabiliste. En outre ces langages sont de type impératif : ils exécutent les instructions d'un programme selon la séquence définie par les instructions de contrôle (if, do, while etc.) ce qui est bien adapté pour faire une simulation, mais pas pour construire des arbres de défaillances. PROLOG a un fonctionnement très différent : il est associé à un mécanisme de raisonnement en "chaînage arrière", qui cherche les valeurs à donner à des variables pour pouvoir satisfaire des prédicats. Ce type de raisonnement serait sans doute pratique pour construire des arbres de défaillances, mais la programmation en PROLOG est très particulière et n'a pas beaucoup d'adeptes. De plus en PROLOG, c'est la simulation qui risquerait d'être difficile. De manière générale, tous ces langages informatiques sont trop généraux pour pouvoir donner lieu à des démonstrations de propriétés telles que celles qui sont évoquées au chapitre 9.

Ainsi, pour répondre au problème de la modélisation des systèmes dans le domaine des études de sûreté de fonctionnement, EDF a développé son propre langage de modélisation : **le langage Figaro**. Cette initiative était unique en 1989, mais cet exemple a été suivi par de nombreux travaux depuis. Un avantage essentiel que possède le langage Figaro sur ses concurrents est sa grande stabilité, qui a permis une capitalisation continue des modèles et des outils de traitement (qui constituent la plateforme outils KB3) depuis le début des années 90.

La diffusion de ces outils à l'extérieur d'EDF ne pourra se faire que si leur support commun est connu, bien documenté et d'accès libre. C'est pourquoi le langage Figaro est libre et ce document est public. On peut ainsi espérer qu'à terme, de même que pour le langage Modelica de modélisation physique des systèmes hybrides, cohabiteront des outils exploitant le langage Figaro d'origines diverses, certains commerciaux et d'autres libres.

2. Principes de modélisation en Figaro

2.1. Les Classes et les Objets

Le langage Figaro est un langage hybride au sens où c'est à la fois un langage objet (qui gère des classes et des objets) et un langage inspiré des techniques d'intelligence artificielle (qui gère des mécanismes d'inférences sur des règles). Cette section présente de manière générale les notions principales de la partie "Objet" de Figaro. La partie "Intelligence artificielle" sera présentée ultérieurement.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

2.1.1. Définitions

Classe : Description commune à tous les composants des systèmes étudiés qui ont le même comportement et les mêmes caractéristiques. Chaque classe est identifiée par un nom et contient une série de caractéristiques qui sont les caractéristiques communes aux composants qu'elle décrit.

Objet : Tout composant d'un système. Chaque objet est identifié par un nom particulier.

En Figaro, chaque objet d'un système appartient à une classe. On dit aussi que l'objet d'une classe est une **instance de la classe**. Les objets d'un système particulier appartenant à la même classe C forment **l'ensemble** des objets de la classe C du système.

Un objet hérite des caractéristiques de sa classe, en les précisant par des informations propres à un système donné. Ces informations sont écrites en gras dans l'exemple ci-dessous.

Exemple : l'objet S002PO est une instance de la classe 'pompe' :

Objet S002PO

Caractéristiques : état initial = **arrêt**,

tableau d'alimentation = **tableau T1**

Une classe en Figaro est appelée un **TYPE**. Les caractéristiques permettant de décrire chaque TYPE ou chaque OBJET sont des champs valués ou des règles. Les **champs valués** permettent de définir les propriétés des objets, notamment les différentes variables qui caractériseront l'état d'un objet.

Exemple de TYPE :

```

TYPE      pompe ;
ATTRIBUT  (* paragraphe permettant de définir les variables d'état
              des objets du type pompe *)
              debit DOMAINE REEL ;
              etat DOMAINE 'arrêt' 'marche' ;

```

Exemples d'OBJETS :

```

OBJET S002PO EST_UNE pompe ;
ATTRIBUT

              debit = 2.5e3 ;
              etat = 'arrêt' ;

OBJET S001PO EST_UNE pompe ;
ATTRIBUT

              debit = 5000 ;
              etat = 'marche' ;

```

Comme nous le verrons plus loin dans ce chapitre, les **règles** permettent de définir le comportement dynamique des objets, c'est à dire les règles d'évolution des variables d'état des objets au cours du temps (par exemple, le démarrage d'une pompe en secours d'une autre, ...).

2.1.2. Structure d'un modèle Figaro

Maintenant que nous avons défini les notions de "classe" et d' "objet", nous pouvons préciser **comment** le langage Figaro est utilisé pour automatiser les études de sûreté de fonctionnement des systèmes. Le principe est le suivant :

- Pour une catégorie de systèmes (systèmes thermohydrauliques, systèmes électriques, ...), on décrit en Figaro dans une **base de connaissances** chaque TYPE de composants appartenant à la catégorie de systèmes étudiée. Une base de connaissances Figaro contient donc un modèle de description commun à tous les systèmes de la classe. C'est un ensemble de TYPES.

- Pour étudier un système particulier, on élabore une **description de ce système** en utilisant la base de connaissances :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

- * Chaque objet du système est créé comme une instance d'un type de la base de connaissances.
- * Pour chaque objet, des valeurs spécifiques au système sont attribuées à chaque champ valué caractéristique du type de l'objet.

L'association (base de connaissances + ensemble des objets du système) constitue un modèle complet du système à étudier. Ce modèle est appelé le **modèle Figaro** du système.

Des traitements permettent ensuite d'exploiter le modèle Figaro du système pour générer automatiquement les modèles de fiabilité requis.

2.2. Modèles d'ordre 1 et d'ordre 0

Le modèle Figaro décrit au paragraphe précédent est dit « modèle d'ordre 1 » en référence au vocabulaire utilisé en intelligence artificielle pour désigner des bases de règles s'appliquant à des ensembles d'objets non définis de manière explicite.

Une règle de comportement est le plus souvent commune à tous les objets décrits par une même classe. C'est pourquoi le langage Figaro permet d'écrire, pour chaque type d'une base de connaissances, des règles génériques à tous les objets décrits par le type (factorisation de la connaissance).

Toute exploitation de ce modèle va passer par une opération d'instanciation qui en fera un « modèle d'ordre 0 » ne contenant que des règles s'appliquant à des variables bien définies et uniques du système modélisé. Cette opération d'instanciation, possible parce que le nombre d'objets qui compose le système est fixé et n'évoluera pas au cours d'une simulation, présente plusieurs avantages :

- On va faire une seule fois un certain nombre de tests qui permettront de simplifier les règles, voire d'en supprimer complètement certaines lorsque leurs conditions sont toujours évaluées à FAUX.
- Le modèle à l'ordre 0 sera facile à compiler de manière efficace pour permettre des traitements rapides.
- La sémantique du modèle à l'ordre 0 peut être définie formellement (cf. cette définition au chapitre 3) de manière simple.
- Il est possible de faire des contrôles de cohérence supplémentaires sur un modèle d'ordre 0.

Dans un modèle Figaro d'ordre 0, il n'y a que des objets. C'est un modèle « à plat », sans aucune hiérarchie.

La suite du chapitre 2 décrit le langage d'ordre 1, sauf mention explicite de l'ordre 0.

2.3. L'héritage entre classes

L'héritage permet de factoriser des informations et ainsi d'avoir des bases de connaissances plus concises, où l'information est mieux structurée.

Prenons l'exemple d'un système thermo-hydraulique. Il contient à la fois des motopompes et des turbopompes. Ces deux types de pompe ont des caractéristiques communes qu'il est judicieux de mettre en commun. Nous allons donc définir une classe générique (celle des pompes) et deux classes spécialisées (celle des pompes motorisées et celles des turbopompes).

```

TYPE      pompe ;
ATTRIBUT
    debit  DOMAINE REEL ;
    etat   DOMAINE 'arret' 'marche' ;

TYPE      pompe_motorisee  SORTE_DE pompe ;
ATTRIBUT
    puissance_moteur  DOMAINE ENTIER ;

TYPE      turbo_pompe  SORTE_DE pompe ;
ATTRIBUT
    capacite_turbine  DOMAINE ENTIER ;

```

La première classe définit les caractéristiques communes des pompes.

Les deuxième et troisième classes héritent de la première, c'est-à-dire qu'elles disposent des mêmes caractéristiques générales (débit et état), auxquelles elles ajoutent leurs propres caractéristiques (puissance du moteur et capacité de la turbine).

La classe *pompe_motorisée* est donc composée d'une partie de *pompe* et d'une partie de *pompe_motorisée*. Une

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

instance de cette classe, c'est-à-dire un objet du type *pompe_motorisée*, devra initialiser les 3 champs (débit, état et puissance du moteur). Un objet de type *pompe* quant à lui ne dispose toujours que des champs débit et état. Etant donné qu'un objet de type *pompe_motorisée* "contient" un objet de type *pompe*, on considère qu'un objet de type *pompe_motorisée* est aussi une instance de *pompe*.

Il est possible dans une sous-classe de redéfinir certaines caractéristiques de la classe dont elle hérite. Par exemple, la classe *pompe_motorisée* pourrait redéfinir le comportement d'une pompe ou bien certaines de ses caractéristiques (pour, par exemple, particulariser le genre d'une interface ou ajouter une valeur énumérée à un domaine), etc. Cela s'appelle de la **surcharge**. Le mécanisme de surcharge permet de redéfinir un champ ou un comportement dans une classe inférieure, c'est-à-dire une classe qui hérite. Cela permet de particulariser une caractéristique générique. Par exemple, un autre état possible d'une pompe motorisée est la demande de démarrage. On aurait donc :

```

TYPE           pompe_motorisee SORTE_DE pompe ;
ATTRIBUT
    puissance_moteur DOMAINE ENTIER ;
    etat DOMAINE 'marche' 'arret' 'demande démarrage' ;

```

Dans cet exemple, on surcharge la facette DOMAINE de l'attribut *etat*. Toute caractéristique ou toute règle ayant un nom peut ainsi être surchargée.

Ce petit exemple a permis de présenter les principaux aspects de la notion d'héritage. Ceux-ci sont les suivants :

- Factorisation d'information dans des classes supérieures (celles qui sont héritées). On **met en commun** plusieurs attributs et règles dans une même classe.
- Spécification des attributs et des comportements dans les classes inférieures (celles qui héritent). On **ajoute** des attributs et des règles dans les classes spécialisées.
- Redéfinition de certains comportements dans des classes inférieures. On **surcharge** les caractéristiques des classes mères.

Cette méthode d'approche est caractéristique des langages objets.

Il est possible en Figaro qu'une classe hérite de plusieurs classes : cela s'appelle de l'héritage multiple. La classe profitera de chacune des caractéristiques des classes héritées. La nouvelle classe dispose alors de l'**union** des caractéristiques des classes héritées. Elle peut alors ajouter des caractéristiques ou bien en redéfinir comme nous venons de le voir. Dans l'exemple suivant, la classe *pompe_motorisée* est une pompe et un composant électrique.

```

TYPE           pompe ;
ATTRIBUT
    debit DOMAINE REEL ;
    etat DOMAINE 'arret' 'marche' ;

TYPE           composant_electrique ;
ATTRIBUT
    puissance_moteur DOMAINE ENTIER ;

TYPE           pompe_motorisee SORTE_DE pompe composant_electrique ;

```

Néanmoins, certains problèmes peuvent apparaître lorsque deux caractéristiques héritées de classes différentes portent le même nom : il n'est pas possible de connaître la provenance de cette caractéristique.

```

TYPE pompe ;
ATTRIBUT
    etat DOMAINE 'marche' 'arret' ;

TYPE appareil_motorise ;
ATTRIBUT
    etat DOMAINE 'marche_rapide' 'marche_lente' 'arret' ;

TYPE pompe_motorisee ;
SORTE_DE pompe appareil_motorise ;
(* Quel est le DOMAINE de l'ATTRIBUT etat ? *)

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Les règles de gestion des mécanismes de surcharge et d'héritage mises en œuvre dans Figaro sont complètement explicitées dans le document [5] qui décrit en détail la syntaxe du langage. Dans ce manuel, nous nous contenterons d'exposer les principes de ces mécanismes et les interdits associés (cf. Notions avancées).

2.4. Communication entre les objets

Un système est un ensemble de composants en interaction. Pour assurer les fonctions du système, les composants sont **reliés** entre eux. On peut distinguer trois types de liens principaux :

- Les liens topologiques : les composants sont reliés entre eux pour propager les flux principaux du système. Exemple : tuyauteries pour propager le fluide dans un système thermohydraulique, câbles pour propager le courant dans les systèmes électriques ;
- Les liens fonctionnels : le fonctionnement d'un composant du système est dépendant d'autres composants. Exemple : une pompe en secours d'une autre ne démarre que si la pompe normale est défaillante ;
- Les liens entre les composants du système et les systèmes support. Exemple : un tableau Basse Tension alimente le moteur d'une pompe.

La modélisation des liens entre les objets d'un système est un point important de l'élaboration d'un modèle, puisque le comportement de chaque objet dépend du comportement des objets qui lui sont reliés.

Lorsque l'on dispose d'un système particulier à modéliser, on connaît la structure du système, et on est donc capable de représenter facilement les liens entre les objets du système. En revanche, lorsqu'on élabore un modèle générique pour une catégorie de systèmes (description des classes dans une base de connaissances), on ne connaît que les relations (potentielles ou obligatoires) existant entre les classes. Par exemple, un objet quelconque de type *pompe_motorisée* est reliée à un et un seul objet de type *tableau_électrique*.

Le langage Figaro permet à la fois :

- de représenter, pour chaque type décrit dans une base de connaissances, les relations potentielles ou obligatoires existant entre un objet quelconque de ce type et les objets des autres types (écriture d'un modèle générique), voire des objets de même type.
- de "renseigner précisément" ces relations au niveau de chaque objet d'un système particulier (d'instancier ces relations) afin de définir précisément l'ensemble des objets du système liés à cet objet particulier.

Les liens entre les objets d'un système sont représentés en Figaro par les **champs d'interface**.

2.4.1. Champs interface d'un type

Lorsqu'on définit un type (une classe d'objets), on doit définir **les relations** existant entre un objet quelconque de ce type et d'autres objets. Plus précisément, les champs interface d'un type permettent de définir :

- les relations existant avec les autres objets, de même type ou de types différents,
- la cardinalité de chaque relation, qui permet de préciser notamment si celle-ci est obligatoire ou non (cf. §2.4.3.2)

Dans une base de connaissances (définition des types), les champs interface permettent de **définir les relations** existant entre les **types**. Chaque champ interface contient **l'identifiant d'une relation**.

Exemple : une pompe est alimentée par un tableau électrique.

```
TYPE pompe_motorisee ;
INTERFACE alim_elec GENRE tableau_electrique CARDINAL 1 ;
```

Une relation définie entre deux types n'est pas symétrique. Dans l'exemple ci-dessus, il est immédiat de savoir quel tableau électrique alimente une pompe, mais pour connaître les pompes alimentées par un tableau électrique donné, il faudra parcourir tout l'ensemble des pompes à la recherche de celles qui ont ce tableau dans leur interface *alim_elec*. Il est donc important de faire les bons choix quand on définit les interfaces dans une base de connaissances.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

2.4.2. Champs interface d'un objet

Dans une description de système (définition des objets), les champs interface permettent de mettre des **ensembles précis** d'objets en relation. Autrement dit, les **relations définies** dans les types sont **renseignées** (ou instanciées) dans les objets.

Exemple 1 : la pompe p1 est alimentée par le tableau basse tension bt1.

```
OBJET p1 EST_UNE pompe_motorisee ;
INTERFACE alim_elec = bt1.
```

Exemple 2 : la pompe p1 est connectée en amont aux deux vannes v1 et v2 et en aval à la vanne v3.

```
OBJET p1 EST_UNE pompe ;
INTERFACE      amont = v1 v2 ;
              aval = v3 ;
```

Pour un objet particulier, chaque champ interface est une liste d'objets, c'est à dire qu'il contient des noms d'objets.

Il est très important de noter que les champs interface d'un objet **doivent** être des champs constants en Figaro. Cela signifie que les liens existants entre les objets d'un système ne pourront pas évoluer au cours de la vie de ce système. Autrement dit, les règles Figaro ne permettront pas d'ajouter un objet à une interface ou d'enlever un objet d'une interface.

Un objet ne peut être mis en interface avec lui-même.

2.4.3. Relations et ensembles

2.4.3.1. Identifiant d'une relation

Il est important de noter que l'identifiant d'une relation peut être assimilé à un nom d'ensemble (au sens mathématique du terme). Reprenons les deux exemples précédents :

- * Si x est un objet quelconque de type *pompe_motorisee*, *alim_elec DE x* désigne l'ensemble des objets de type *tableau_electrique* qui sont en relation avec x dans un système quelconque décrit à l'aide de la base de connaissances. Dans l'exemple ci-dessus, *alim_elec DE p1* désigne le singleton {bt1}.
- * Si x est un objet quelconque de type *composant_hydraulique*, *amont DE x* désigne l'ensemble des objets de type *composant_hydraulique* qui sont connectés en amont à x dans un système quelconque décrit à l'aide de la base de connaissances. Dans l'exemple ci-dessus, *amont DE p1* désigne l'ensemble {v1, v2}.

2.4.3.2. Cardinalité

Lorsqu'on définit une relation entre un type A et un type B dans une base de connaissances, il est important de spécifier des contraintes sur le cardinal de l'ensemble des objets de type B qui seront en relation avec un objet quelconque de type A. Cela permet notamment de spécifier si la relation est obligatoire ou non. Plusieurs possibilités sont offertes en Figaro, que nous allons illustrer à l'aide de l'exemple suivant :

```
TYPE composant ;
INTERFACE amont GENRE composant;
```

- Si on ne spécifie aucune contrainte sur la relation *amont*, cela signifie qu'un composant peut n'avoir aucun composant en amont (la relation n'est **pas obligatoire**) ou qu'il peut en avoir plusieurs, sans limitation de nombre. Le cardinal de l'ensemble des objets de type composant en amont d'un objet de type composant est indéfini (l'ensemble peut être vide).

```
TYPE composant ;
INTERFACE amont GENRE composant;
```

Remarque : on aurait pu aussi écrire :

```
TYPE composant ;
INTERFACE amont GENRE composant CARDINAL 0 JUSQUA INFINI ;
```

- Si on spécifie que la cardinalité de la relation *amont* est égale à n, cela signifie qu'un composant doit

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

obligatoirement avoir n composants en amont. Le cardinal de l'ensemble des objets de type composant en amont d'un objet de type composant est égal à n .

```
TYPE composant ;
INTERFACE amont GENRE composant CARDINAL 3;
```

Dans ce cas, si $n=1$, on dit que l'interface est **monovaluée**. Ce cas particulier est important car il permettra d'écrire des règles avec une syntaxe plus simple que dans le cas général.

- Si on spécifie que la cardinalité de la relation *amont* est comprise dans l'intervalle p,q cela signifie qu'un composant doit obligatoirement avoir entre p et q composants en amont. Le cardinal de l'ensemble des objets de type composant en amont d'un objet de type composant est compris entre p et q .

```
TYPE composant ;
INTERFACE amont GENRE composant CARDINAL 2 JUSQUA 8;
```

Dans ce cas, si $p = 0$, la relation n'est pas obligatoire.

2.4.4. Utilisation des champs d'interface

Les champs d'interface seront utilisés pour modéliser l'influence des objets reliés à un composant sur le comportement de ce composant.

Ainsi, comme nous le verrons lors de la présentation des règles Figaro, mettre le type B en interface du type A (en relation avec le type A) permettra d'utiliser les caractéristiques du type B pour modéliser le comportement du type A.

Ainsi, la définition des interfaces existant entre les types décrits dans une base de connaissances est une étape importante de la conception d'une base de connaissances.

2.4.5. Modélisation des liens matériels entre objets

La représentation graphique d'un système associée à une description Figaro est constituée de nœuds (icônes) et de liens (traits, munis de divers attributs d'épaisseur, couleur, types de flèches etc.). Chaque objet graphique doit avoir son pendant en Figaro. Cependant, il existe deux types de situations, selon que les liens sont juste déclarés ou bien contiennent des champs valués, voire même des règles de comportement au même titre que les nœuds.

Lorsque dans la réalité physique deux objets sont reliés entre eux par un lien matériel (tuyauterie, câble, ...), il faut noter que l'on peut envisager deux types de modélisation pour ces liens, en fonction des objectifs de la modélisation.

Si ces liens physiques ont des caractéristiques propres utiles à la modélisation du comportement du système, il faut définir des types Figaro classiques pour ces liens.

Par exemple, si pour une classe de systèmes électriques, les caractéristiques des câbles (impédance, longueur) sont importantes pour la modélisation, on définira un type *cable_electrique* dans la base de connaissances. Ce type sera lui-même interfacé avec les autres types du modèle (chaque câble aura un composant en amont et en aval).

Dans le cas où les liens physiques n'ont pas de caractéristiques propres utiles à la modélisation du comportement du système, il suffira de déclarer des types « creux » (sans contenu Figaro) pour ces liens. En revanche, ils seront associés à des instructions de remplissage des interfaces des nœuds sur lesquels ils seront connectés dans le fichier XML de paramétrage de l'interface graphique de KB3.

Par exemple, si pour une classe de systèmes thermohydrauliques on considère les tuyaux comme parfaits, les liens servent uniquement à la représentation graphique du système, alors que la propagation du flux est faite directement d'un composant hydraulique représenté par un nœud vers les composants déclarés dans son champ interface *aval*. Cette interface sera remplie automatiquement par KB3 lors de la saisie graphique d'un système, quand l'utilisateur dessinera les liens.

2.5. Champs de classes et d'objets

Préambule de vocabulaire : nous utilisons le mot « champ » plutôt que attribut ou propriété pour éviter une confusion avec le vocabulaire des modèles UML (cf. § 2.6 pour la correspondance entre les concepts UML et Figaro).

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Comme nous l'avons vu précédemment, un type est décrit par des champs communs aux objets du type, et par des règles que nous étudierons dans la partie suivante.

Les champs caractéristiques d'un type ou d'un objet peuvent être de trois natures différentes :

- * Les **champs constants** représentent les caractéristiques des objets qui n'évoluent pas au cours de la vie du système (longueur d'un câble de connexion par exemple). Ces champs sont univalués.
- * Les **champs interface d'un type** permettent de définir les relations potentielles ou obligatoires existant entre un objet de ce type et des objets des autres types, voire du même type. Les **champs interface d'un objet** permettent de renseigner les objets d'un système en relation avec cet objet. Ce sont des champs d'objets, c'est-à-dire contenant des noms d'objets, qui peuvent être multivalués.
- * Les **variables d'état** caractérisent l'état des objets et sont susceptibles d'évoluer au cours de la vie du système. Ces champs sont univalués.

Il est important de noter que, en Figaro, les **champs interface sont des champs constants**. Autrement dit, on ne peut pas modéliser en Figaro des processus qui auraient pour conséquence de modifier les relations existant entre les objets d'un système.

Prenons l'exemple simple de la description d'un type "câble électrique" caractérisé par sa longueur.

Exemple :

```
TYPE cable_electrique ;
```

```
CONSTANTE
```

```
longueur DOMAINE REEL ;
```

(*le domaine de définition de la constante est l'ensemble des réels*)

Avec la définition de ce type, on peut créer une instance particulière de *câble_électrique* pour décrire un système, en affectant au champ *longueur* la valeur pertinente pour le système étudié :

Exemple :

```
OBJET cable1 EST_UN cable_electrique ;
```

```
CONSTANTE
```

```
longueur = 25 ;
```

2.6. Lien entre les concepts Figaro, UML et SysML

Il est facile de faire une correspondance entre les concepts principaux de Figaro et les concepts les plus couramment utilisés dans un méta modèle décrit en UML ou SysML. Le tableau ci-dessous résume cette correspondance.

Concept UML	Concept SysML (vocabulaire anglais)	Concept Figaro
Classe	Block	Type
Objet	Part	Objet
Héritage	Inheritance	Héritage
Attribut	Value	Constante, attribut, effet, panne
Relation	Flow and Port	Interface

Nous n'avons pas mis en correspondance les concepts d'opération présents en UML et SysML avec les règles de Figaro, bien que dans les deux cas il s'agisse de décrire un comportement des classes. En effet, les règles ne se comportent pas comme des méthodes que l'on appelle avec des arguments, à l'instar de ce qui existe dans les langages orientés objet.

Malgré cette restriction, il est facile d'utiliser un éditeur graphique UML ou SysML pour représenter tous les éléments structurels du méta modèle d'une base de connaissances Figaro. Une telle représentation graphique peut être un support de réflexion intéressant pour la construction d'une nouvelle base de connaissances ou pour représenter de manière synthétique le contenu d'une base existante.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

2.7. Figaro et les unités

Un choix très simple a été fait quant à la gestion des unités dans les modèles Figaro : elles ne sont jamais représentées explicitement. Cela a l'avantage de simplifier la syntaxe, mais du coup l'utilisateur doit bien savoir ce qu'il fait, et exprimer toutes ses expressions Figaro avec des nombres correspondant à un système d'unités cohérent. Heureusement, il n'y a dans l'immense majorité des modèles qu'une seule unité : celle prise pour le temps. Ainsi si l'utilisateur choisit l'heure comme unité, cela signifie qu'il doit exprimer les taux de défaillance et de réparation des lois exponentielles en h^{-1} , et les paramètres de lois T_C en heures, ainsi que les temps de mission qu'il précise dans les outils de calcul.

2.8. Règles Figaro

En langage Figaro, le comportement dynamique (c'est-à-dire au cours du temps) des objets est modélisé sous forme de règles. Pour bien comprendre ce qu'est une règle en Figaro, il est nécessaire de comprendre le **principe général de modélisation des systèmes** utilisé dans un modèle Figaro. Cette section donne des explications simples, avec beaucoup d'exemples. La section 3 donnera une définition formelle de la sémantique du langage, base du fonctionnement des outils fondés sur Figaro.

2.8.1. Principe de modélisation des systèmes

Etat de modèle et événement

L'état d'un objet à un instant donné se définit comme l'ensemble des valeurs prises à cet instant par les variables d'état de cet objet. **L'état d'un système** à un instant donné est complètement défini par l'état des objets qui constituent le modèle Figaro à cet instant.

L'état du système va évoluer en fonction du temps, suite à des **événements** pouvant survenir sur les objets du système.

L'occurrence d'un événement est conditionnée par l'état courant du système. Autrement dit, un événement ne peut se produire que si les variables d'état du modèle vérifient une certaine condition.

Si la pompe P1 est en marche

Alors, il peut se produire l'événement "Défaillance en fonctionnement de la pompe"

Un événement correspond à un phénomène aléatoire, qui se produit en fonction d'une certaine loi de probabilité, ou à un phénomène déterministe qui se produit au bout d'un temps fixe donné (si la condition d'occurrence de l'événement est maintenue pendant cette durée).

Si la pompe P1 est en marche,

Alors, il peut se produire l'événement "Défaillance en fonctionnement de P1" suivant une loi de probabilité exponentielle de paramètre $1e-3/h$.

Le langage Figaro permet donc de modéliser des processus markoviens (l'occurrence d'un événement ou d'une transition dépend **uniquement** de l'état courant du système) ou des processus semi-markoviens (l'occurrence d'un événement dépend de l'état courant du système et du temps écoulé dans cet état). C'est très semblable à ce qui se passe dans un réseau de Petri stochastique.

Conséquences d'un événement sur le système

Un événement survenant sur un composant du système va modifier l'état du composant, ce qui revient à modifier les variables d'état associées à ce composant. Par exemple, l'occurrence de l'événement "Défaillance en fonctionnement de P1" modifie une variable d'état de P1 qui passe de 'marche' à 'panne'.

La modification de l'état du composant touché par l'événement va entraîner toute une série de conséquences sur les autres composants du système et sur le système lui-même. (Les effets d'un événement sur un composant particulier vont se "propager" sur tout le système). On peut citer pour illustrer les deux exemples suivants :

Si la pompe P1 est en panne,

Alors la file d'alimentation passant par P1 est perdue.

Si la pompe P1 est en panne,

Alors la pompe de secours associée à P1 va être sollicitée au démarrage.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Cycle de vie d'un système

On considère que la vie d'un système est composée d'une suite de cycles, chaque cycle pouvant être décomposé en deux étapes :

- Lorsque le système est dans un état donné, certains événements peuvent se produire. Dans un premier temps, l'occurrence d'un événement sur un composant aura des conséquences **directes et locales** sur l'état de ce composant.
- Dans un deuxième temps, la modification de l'état des variables du composant va entraîner des conséquences sur les autres composants du système, **propageant** ainsi les effets de l'événement pour obtenir un nouvel état de système.

Une fois que le système est dans ce nouvel état, d'autres événements peuvent se produire.

Pour permettre de modéliser le comportement des systèmes en fonction de ce principe, le langage Figaro utilise un formalisme en règles. Deux types de règles existent en Figaro :

- Les **règles d'occurrence** permettent de modéliser les événements qui peuvent se produire sur un objet : conditions d'occurrence de l'événement en fonction de l'état du système, loi de probabilité associée, conséquences sur l'état du composant.
- Les **règles d'interaction** permettent de propager les conséquences d'un événement à tout le système en exprimant les relations existant entre l'état d'un objet et l'état des autres objets du système.

Avant d'étudier comment ces règles sont utilisées pour exploiter un modèle Figaro, nous allons définir plus précisément leur formalisme.

2.8.2. Formalisme d'une règle Figaro

Le formalisme d'une règle Figaro diffère selon qu'il s'agit d'une règle d'occurrence ou d'une règle d'interaction.

2.8.2.1. Forme d'une règle d'occurrence

Une règle d'occurrence modélise l'influence sur le système d'un événement (on dit aussi : une transition) pouvant survenir sur les objets d'un système. De façon schématique, ces règles sont de la forme :

```
SI <condition>
IL PEUT SE PRODUIRE <événement>
PROVOQUE <actions>
```

La condition est une expression booléenne qui exprime la condition que doit vérifier l'état du système pour qu'un événement se déclenche sur un objet. (Si la condition est évaluée à VRAI, l'événement peut se produire). Les informaticiens appellent ce type de condition la « garde » de l'événement (transition).

Les actions décrivent les conséquences de l'événement sur les variables d'état de l'objet touché par l'événement.

Exemple de règle d'occurrence :

```
TYPE pompe;
ATTRIBUT
  etat DOMAINE 'arret' 'sollicite' 'marche' 'panne';
CONSTANTE
  lambda PAR_DEFAULT 1e-3 ;
OCCURRENCE
  SI etat = 'marche'
  IL PEUT SE PRODUIRE
  TRANSITION defaillance_en_fonctionnement
    LOI EXP ( lambda )
    PROVOQUE etat <-- 'panne' ;
```

Cette règle d'occurrence **générique** aux objets de la classe *pompe* se lit :

Soit o un objet quelconque de type pompe,

Si etat de o est 'marche',

Il peut se produire l'événement 'défaillance en fonctionnement' de o au bout d'un temps de loi exponentielle de

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

paramètre *lambda*, qui provoque le passage à 'panne' de la variable état de *o*.

2.8.2.2. Forme d'une règle d'interaction

Une règle d'interaction modélise les relations logiques existant entre l'état d'un objet et l'état des autres objets du système. De façon schématique, ces règles sont de la forme :

```
SI <condition>
ALORS <actions>
SINON <actions>
```

Pour ces règles, la condition (expression booléenne introduite par SI) permet le déclenchement d'un ensemble d'actions affectant les variables d'état des objets du système. (Si la condition est évaluée à VRAI, les actions introduites par ALORS se déclenchent ; si la condition est évaluée à FAUX, les actions introduites par SINON se déclenchent).

Exemple de règle d'interaction :

```
TYPE pompe;
ATTRIBUT
    etat DOMAINE 'arret' 'sollicite' 'marche' 'panne';
INTERFACE
    pompe_secours GENRE pompe CARDINAL 1;
INTERACTION
    SI etat = 'panne'
    ALORS
        etat DE pompe_secours <-- 'sollicite' ;
```

Dans cet exemple, l'ensemble des objets de type pompe en relation *pompe_secours* avec une pompe quelconque est un ensemble de cardinal 1. Ainsi, *etat DE pompe_secours* désigne l'état de l'unique pompe en secours d'une pompe quelconque. Cette règle d'interaction générique aux objets de la classe *pompe* se lit :

Soit *o* un objet quelconque de type pompe,
Si état de *o* est 'panne',
Alors état de l'unique objet de type pompe en relation *pompe_secours* avec *o* passe à 'sollicité'.

2.8.2.3. Condition et actions d'une règle, utilisation des champs d'interface

En principe, le comportement des objets d'une classe dépend principalement de l'état de ces objets et de l'état des objets des classes avec lesquels ils sont en relation (classes en interface). Ainsi, pour une classe donnée :

- la condition d'une règle est une expression booléenne utilisant les propriétés de la classe et les propriétés des classes en interface,
- les actions d'une règle affectent les variables de la classe et les variables des classes en interface. (Par définition, les actions d'une règle d'occurrence affectent uniquement les variables de la classe).

Pour illustrer, nous allons donner quelques exemples qui montrent comment sont utilisés les champs interface d'un type pour écrire les règles de comportement de ce type.

Exemple 1

Prenons l'exemple des objets de type pompe. Dans cet exemple, le type pompe est spécialisé en deux sous-types : les pompes qui fonctionnent en temps normal (type *pompe_normale*) et les pompes qui fonctionnent en secours des pompes normales (type *pompe_secours*).

Chaque objet de type *pompe_secours* permet de secourir une ou plusieurs pompes normales. Autrement dit, chaque objet de type *pompe_secours* est relié à une ou plusieurs pompes normales. La règle de comportement que nous souhaitons modéliser est la suivante :

Si toutes les pompes normales secourues par la même pompe de secours sont en panne, alors le système demande le démarrage de cette pompe secours.

Le comportement des objets de type *pompe_secours* dépendant de l'état des objets de type pompe normale, il faut créer une relation entre le type *pompe_secours* et le type *pompe_normale*. Cette relation est identifiée par le champ interface *secourue* et elle est de cardinalité 1 à INFINI.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

TYPE pompe;
ATTRIBUT
    panne DOMAINE BOOLEEN ;

TYPE pompe_normale SORTE_DE pompe;

TYPE pompe_secours SORTE_DE pompe;
ATTRIBUT
    demande_demarrage DOMAINE BOOLEEN ;
INTERFACE
    secourue GENRE pompe_normale CARDINAL 1 JUSQUA INFINI;
INTERACTION
SI QUSOIT x UNE secourue ON_A panne DE x = VRAI
ALORS demande_demarrage <-- VRAI ;

```

Cette règle d'interaction se lit :

*Soit o un objet quelconque de type pompe_secours,
Si $\forall x \in \{\text{pompes normales secourues par } o\}$, x est en panne,
Alors, il y a demande de démarrage de o.*

Dans cet exemple, la condition de la règle dépend des caractéristiques des objets d'une classe en interface de la classe *pompe_secours*. L'action de la règle affecte uniquement les objets de la classe décrite. **Autrement dit, cette règle d'interaction permet de modifier l'état d'un objet de la classe en fonction de l'état des objets des classes en interface.**

Exemple 2

Dans ce nouvel exemple, on suppose que chaque pompe normale d'un système est secourue par une ou plusieurs pompes secours. La règle de comportement que nous souhaitons modéliser est la suivante :

Si une pompe normale est en panne, alors le système demande le démarrage de toutes les pompes de secours associées.

```

TYPE pompe;
ATTRIBUT
    panne DOMAINE BOOLEEN;

TYPE pompe_secours SORTE_DE pompe;
ATTRIBUT
    demande_demarrage DOMAINE BOOLEEN;

TYPE pompe_normale SORTE_DE pompe;
INTERFACE
    secours GENRE pompe_secours CARDINAL 1 JUSQUA INFINI;
INTERACTION
SI panne = VRAI1
ALORS
POUR_TOUT x UN secours FAIRE demande_demarrage DE x <--VRAI2;

```

Cette règle d'interaction se lit :

*Soit o un objet quelconque de type pompe_normale,
Si o est en panne,
Alors, $\forall x \in \{\text{pompes secours en secours de } o\}$, il y a demande de démarrage de x.*

Dans cet exemple, la condition de la règle dépend uniquement des caractéristiques des objets de la classe décrite. L'action de la règle affecte les objets de la classe *pompe_secours* en interface. **Autrement dit, cette règle d'interaction permet de modifier l'état des objets des classes en interface en fonction de l'état des objets de la classe décrite.**

¹ Cette condition panne = **VRAI** pourrait aussi s'écrire simplement panne.

² Cette affectation demande_demarrage **DE** x <--**VRAI** pourrait aussi s'écrire simplement demande_demarrage.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Problèmes potentiels

Les exemples ci-dessus montrent que la définition et l'utilisation des interfaces doivent être faites avec beaucoup de soin, de façon à ne pas écrire des incohérences. Des problèmes peuvent par exemple se poser s'il existe deux relations différentes entre deux classes.

Dans l'exemple simple suivant, le type t1 est en relation avec t2 par l'interface r12 et, réciproquement, le type t2 est en relation avec t1 par l'interface r21. La règle 'Règle_1' modifie l'état d'un objet de type t1 en fonction de l'état de l'objet de type t2 interfacé. La règle 'Règle_2' modifie l'état d'un objet de type t2 en fonction de l'état de l'objet de type t1 interfacé. Dans cet exemple, ces deux règles sont incohérentes car elles ne permettent pas de conclure sur l'état des objets de ces deux types (cf. § 8 sur la cohérence).

```

TYPE t1;
ATTRIBUT
  a DOMAINE BOOLEEN;
INTERFACE
  r12 GENRE t2 CARDINAL 1;
INTERACTION
  Règle_1
SI b(r12) = VRAI
ALORS a <-- FAUX
SINON a <-- VRAI;

TYPE t2;
ATTRIBUT
  b DOMAINE BOOLEEN;
INTERFACE
  r21 GENRE t1 CARDINAL 1;
INTERACTION
  Règle_2
SI a(r21) = VRAI
ALORS b <-- VRAI
SINON b <-- FAUX;

```

Interface monovaluée et interface multivaluée

Il est important de noter que l'utilisation d'une interface monovaluée (de cardinal 1) diffère de celle d'une interface multivaluée. Comme nous le verrons plus précisément dans la suite de ce manuel, un champ interface monovalué est utilisé comme un champ normal, alors qu'un champ interface multivalué ne peut être géré que par des quantificateurs (IL_EXISTE, IL_EXISTE AU_MOINS, QQSOIT), "itérateurs" (POUR_TOUT, SOIT) ou des opérateurs complexes (SOMME POUR_TOUT, PRODUIT POUR_TOUT...).

2.8.3. Principes d'utilisation des règles

Exploiter un modèle Figaro, c'est appliquer l'ensemble des règles de la base de connaissances aux objets décrivant le système à étudier.

Deux types d'application des règles Figaro sont possibles, correspondant à deux types de traitements : application en chaînage avant et application en chaînage arrière.

Dans tous les cas, l'exploitation se fait sur le modèle instancié à l'ordre 0, ainsi que nous l'avons vu au § 2.2.

2.8.3.1. Application en chaînage avant

Le traitement en chaînage avant est un traitement cyclique qui s'effectue en deux temps (cf. Figure 2) :

- Déclenchement d'un événement (on dit aussi déclenchement ou tir d'une transition) sur un objet du système
- Propagation des effets de l'événement sur le système par application cyclique des règles d'interaction jusqu'à stabilisation de l'état.

a) Déclenchement d'un événement

Lorsque le système est dans un état donné, certains événements peuvent se produire. Cela signifie que les

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

conditions de certaines règles d'occurrence de certains objets sont vérifiées. Le moteur choisit une³ règle d'un objet et l'applique (exécute les actions provoquées par l'événement), ce qui modifie les variables d'état de l'objet. Le système se trouve alors dans un état que nous qualifierons d'**incomplet**, puisque seules les conséquences de l'événement sur l'objet affecté sont prises en compte. Un état incomplet n'est pas un état réel du système.

b) Application des règles d'interaction

Pour propager les conséquences de l'événement à tout le système, le traitement applique ensuite les règles d'interaction. Pour cela, il procède comme suit :

Passage dans les règles d'interaction

Le traitement applique les règles d'interaction objet par objet.

Pour chaque objet, le traitement prend la "première" règle de l'objet et détermine si elle est applicable en fonction de l'état courant du système (initialement, cet état est l'état incomplet obtenu après tirage de la règle d'occurrence). Si la règle est applicable, il l'applique, calcule le nouvel état courant du système et passe à la règle suivante de l'objet.

Lorsque toutes les règles d'un objet ont été examinées, le traitement passe à l'objet suivant.

Remarque : Il est très important de noter qu'un nouvel état du système est calculé après chaque application d'une règle d'interaction pour un objet. Ainsi, si une règle d'un objet n'est pas applicable initialement, elle peut devenir applicable après un passage dans les premières règles.

Lorsque le traitement est passé une fois dans chacune des règles d'interaction, un nouvel état du système est obtenu. Mais ce n'est pas forcément un état complet correspondant à un état réel du système : les règles sont à nouveau appliquées de la première à la dernière (dans le même ordre que lors du premier « tour »). Si, à la fin de deux tours successifs, l'état obtenu est le même, on dit qu'il y a convergence et on peut arrêter l'inférence. Cela signifie que l'on a atteint un « point fixe ».

Exemple :

Prenons l'exemple simple de 3 composants thermohydrauliques A, B et C connectés en série. Dans la base de connaissances, nous avons la règle d'interaction suivante :

```

TYPE composant;
INTERFACE
    montant GENRE composant;
ATTRIBUT
    alimente DOMAINE BOOLEEN;
INTERACTION
    r1
    SI QQSUIT x UN montant ON_A alimente DE x = FAUX
    ALORS alimente <-- FAUX;

```

Cette règle générique r1 se lit :

*Soit o un objet quelconque de type composant,
Si $\forall x \in \{\text{composants en amont de } o\}$, x n'est pas alimenté,
Alors o n'est pas alimenté.*

La base de faits telle qu'initialisée par l'utilisateur est la suivante :

```

alimente DE A = VRAI
alimente DE B = VRAI
alimente DE C = VRAI

```

Supposons qu'un événement survenu sur le système ait provoqué alimenté DE A = FAUX. L'état incomplet du système après cet événement est donc :

```

alimente DE A = FAUX
alimente DE B = VRAI
alimente DE C = VRAI

```

La règle d'interaction r1 instanciée sur l'objet B est la suivante :

³ Nous verrons plus tard que, dans certains cas, plusieurs événements peuvent être appliqués simultanément.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

OBJET B;

INTERACTION

```

r1
SI alimente DE A = FAUX
ALORS alimente DE B = FAUX ;

```

La règle r1 de l'objet B est donc applicable. Après application, le nouvel état du système est :

```

alimente DE A = FAUX
alimente DE B = FAUX
alimente DE C = VRAI

```

Compte tenu de ce nouvel état, la règle r1 instanciée sur l'objet C est applicable (elle ne l'était pas initialement). Le nouvel état obtenu pour le système est :

```

alimente DE A = FAUX
alimente DE B = FAUX
alimente DE C = FAUX

```

Application des règles d'interaction jusqu'à stabilisation

Les règles d'interaction permettent de propager les effets d'un événement de proche en proche. Il faut donc **faire autant de passages dans ces règles qu'il est nécessaire** pour que toutes les conséquences d'un événement soient prises en compte. Lorsque le point fixe est atteint⁴, cela signifie que toutes les conséquences de l'événement ont été prises en compte.

Après passage dans les règles d'interaction, on obtient un nouvel état du modèle (état complet) pour lequel **d'autres événements peuvent éventuellement se produire**.

Exemple :

Reprenons l'exemple précédent en supposant que le système examine l'objet C avant l'objet B. Dans ce cas, lors du premier passage, seule la règle instanciée sur l'objet B est appliquée et l'état obtenu après premier passage est :

```

alimente DE A = FAUX
alimente DE B = FAUX
alimente DE C = VRAI

```

Lors du deuxième passage dans les règles d'interaction, la règle instanciée sur l'objet C sera appliquée et on obtiendra le même état final que précédemment.

Remarque sur l'ordre d'application des règles d'interaction

Il est très important de noter que **l'ordre d'application des règles d'interaction ne doit avoir aucune conséquence sur le résultat obtenu**. (C'est le cas dans notre exemple où, quel que soit l'ordre de prise en compte des objets, le résultat obtenu est le même – seul diffère le nombre de « tours » des règles nécessaire pour stabilisation de l'état).

Ainsi, le concepteur de bases de connaissances devra veiller à la **commutativité des règles** d'interaction, car il ne maîtrise pas l'ordre de tirage des règles par le moteur.

Pour des cas complexes, nous verrons plus loin dans ce manuel (cf. § 7.5) qu'il est possible d'introduire un ordre partiel dans l'application des règles d'interaction, en regroupant celles-ci en sous-ensembles de règles (étapes) et en appliquant chaque sous-ensemble l'un après l'autre. Le chapitre 8 consacré à la cohérence des modèles explique les précautions à prendre dans l'utilisation des étapes.

Exploitation d'un modèle Figaro en chaînage avant

L'exploitation d'un modèle Figaro en chaînage avant, la plus « naturelle », permet d'explorer tous les états d'un système ainsi que les événements reliant ces états. Ce type d'exploitation peut donc être utilisé :

- En simulation interactive : l'utilisateur choisit lui-même la suite de transitions qu'il veut simuler et il peut ainsi voir les effets sur le système à chaque transition

⁴ Un tel point fixe n'existe pas toujours. Cf. le § 9 sur la cohérence des modèles pour assurer son existence.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

- Pour construire la matrice du **graphe d'états** du système et calculer ainsi la probabilité de chaque état (cela n'est possible que si le modèle est markovien, c'est-à-dire si toutes les transitions sont de loi instantanée ou exponentielle).
- Pour explorer les **séquences** du graphe de manière à calculer les probabilités des séquences menant à un état de panne du système. (Une séquence est une suite d'événements menant de l'état initial du système à un état particulier).
- Pour faire une simulation de Monte Carlo (cf. §3.4).

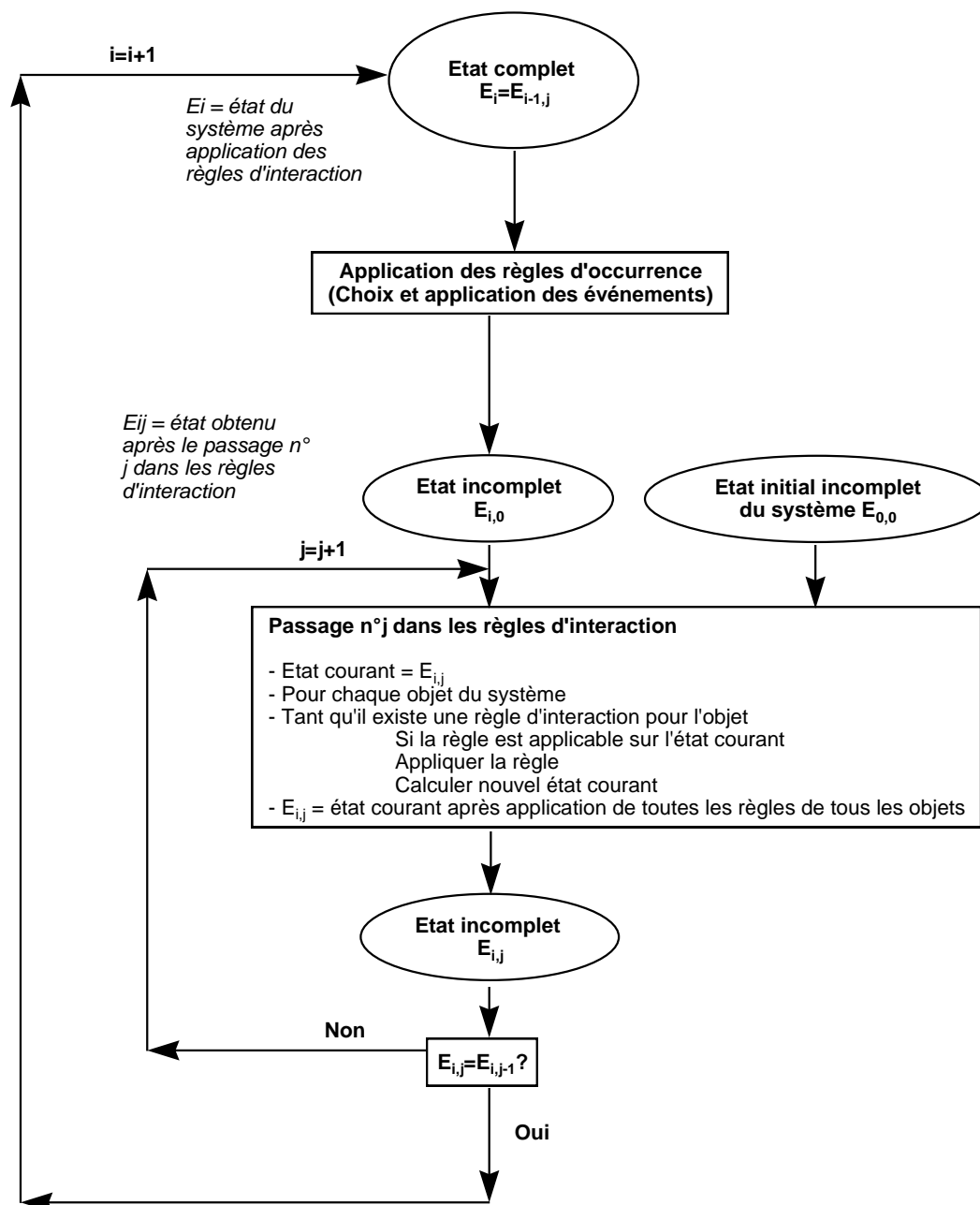


Figure 2. Schéma de principe du chaînage avant en Figaro

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

2.8.3.2. Application en chaînage arrière

L'application des règles Figaro en chaînage arrière est utilisée pour générer automatiquement un **Arbre de défaillances** à partir :

- du modèle Figaro 0 associé au système à étudier,
- de la description de l'événement indésirable que l'on cherche à étudier, qui est défini par un fait du type : égalité d'une variable du modèle avec une valeur donnée.

Exemple : Un modèle Figaro représentant un système thermohydraulique contient des objets représentant des composants thermohydrauliques. Chaque objet est décrit notamment par une variable booléenne *alimenté* qui définit si l'objet est alimenté ou non en fluide. Si l'objet B1 représente le but du système à alimenter, l'événement indésirable peut être décrit par le fait *alimenté (B1) = FAUX*.

Pour construire l'Arbre de défaillances, le traitement procède en deux étapes :

- construction d'un "arbre des causes" (composé uniquement de portes logiques) par chaînage arrière sur les règles d'interaction du modèle Figaro,
- transformation de "l'arbre des causes" en Arbre de défaillances en utilisant les règles d'occurrence du modèle ou bien les modèles de fiabilité associés aux pannes pour créer des modèles de défaillance probabilistes au niveau des feuilles de l'arbre.

a) Construction de "l'arbre des causes" par chaînage arrière sur les règles d'interaction

Pour construire "l'arbre des causes" associé à l'événement indésirable, on applique **les règles d'interaction** du modèle en chaînage arrière.

"L'arbre des causes" associé à un événement indésirable est un arbre explicitant les relations logiques existant entre l'événement indésirable et des événements de base. Un événement de base est un fait que l'on ne peut pas expliquer sous forme d'autres faits figurant dans les règles d'interaction. Il correspond :

- * soit à une défaillance d'un composant du système (par exemple "défaillance à la sollicitation de la pompe p1 = VRAI"),
- * soit à une expression représentative de l'état du système (par exemple "position de la vanne v1= ouvert").

Pour construire l'arbre des causes associé à un événement indésirable, on applique le processus suivant.

Partant du fait à expliquer (de l'événement indésirable à expliquer), on sélectionne toutes les règles d'interaction (RV1, ..., RVn) concluant sur ce fait (dont les actions provoquent le fait), ainsi que toutes les règles d'interaction (RF1, ..., RFp) dont la conclusion est incompatible avec ce fait (dont les actions donnent une valeur différente à la variable).

Le raisonnement appliqué est alors le suivant. Pour que le fait soit vérifié, il faut et il suffit qu'une des règles RV1...RVn (de conditions CV1...CVn) s'applique et qu'aucune des règles RF1...RFp (de conditions CF1...CFp) ne s'applique. On construit alors l'élément d'arbre suivant :

Evénement indésirable = (CV1 ou CV2...ou CVn) et non (CF1 ou CF2 ... ou CFp)

Le chaînage arrière continue en appliquant le même raisonnement sur chacune des conditions non développées de l'arbre.

Remarque : Il est important de noter que, pour que le traitement en chaînage arrière soit applicable, les règles d'interaction doivent notamment vérifier les contraintes suivantes :

- la condition d'une règle doit être une expression booléenne simple combinant par des opérateurs des expressions booléennes atomiques du type :
"variable (objet) = valeur",
- les actions d'une règle doivent être du type "variable (objet) <-- valeur".

Le chaînage s'arrête lorsque, pour chaque feuille de l'arbre :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

- aucune règle d'interaction ne conclut sur le fait représenté par la feuille ou sur un fait incompatible,
- la feuille a déjà été explicitée dans l'arbre des causes.

"L'arbre des causes" obtenu est donc un arbre logique comprenant des portes ET, OU, k/n et éventuellement des négations, et où les feuilles de l'arbre sont des expressions booléennes simples (tests sur la valeur d'une variable).

b) Transformation de l'Arbre des causes en Arbre de défaillances

Pour transformer l'arbre des causes en Arbre de défaillances, le traitement utilise les règles d'occurrence du modèle et les modèles de fiabilité associés aux pannes. Pour chaque feuille de l'arbre des causes :

- Si il existe une règle d'occurrence applicable (c'est-à-dire dont la condition est vérifiée pour l'état initial du modèle) dont les actions provoquent le fait associé à la feuille, la feuille est remplacée par l'événement de la règle d'occurrence et suivant son type (instantané ou temporisé) un modèle de feuille adéquat est créé.

Exemple :

Une feuille de l'arbre contient le fait "défaillance en fonctionnement de la pompe P1 = VRAI" (*def DE P1 = VRAI*) . Le modèle Figaro contient la règle d'occurrence suivante pour le type pompe.

```

TYPE pompe ;
ATTRIBUT
  def DOMAINE BOOLEEN ;
CONSTANTE
  etat_initial DOMAINE 'marche' 'arret' ;
OCCURRENCE
  SI etat_initial = 'marche'
  IL PEUT SE PRODUIRE
    DEFAILLANCE def
      LIBELLE "defaillance en fonctionnement"
      LOI EXP (lambda_pompe)
      PROVOQUE def <-- VRAI;
```

Cette règle d'occurrence est applicable pour l'objet P1 qui est initialement en marche. Le traitement remplace donc la feuille de l'arbre des causes par l'événement de base décrit par la règle ci-dessus, auquel est associé un taux de défaillance *lambda_pompe*.

- Si la feuille de l'arbre correspond au fait qu'une PANNE (cf. §6.2.4.4) est à VRAI et si il existe un modèle de fiabilité associé à cette panne, ce modèle est directement mis dans la feuille. Cette deuxième possibilité a été introduite pour permettre de générer des arbres de défaillances avec des modèles de feuilles variés, éventuellement complexes, qu'il n'aurait pas été possible d'inférer à partir des règles d'occurrence. Ce mécanisme est en quelque sorte en concurrence avec le précédent. Il faut donc choisir lequel on veut exploiter, et éventuellement utiliser des groupes de règles (cf. § 7.4) distincts pour l'exploitation du modèle en chaînage avant et en chaînage arrière ; on peut ainsi utiliser les règles d'occurrence pour le chaînage avant et les modèles de fiabilité associés aux pannes pour le chaînage arrière.
- Si il n'existe pas de règle d'occurrence provoquant le fait associé à la feuille et si ce fait n'est pas une panne associée à un modèle de fiabilité, l'expression booléenne associée est évaluée en fonction de l'état initial du modèle et l'arbre des causes est simplifié en fonction du résultat obtenu. Les règles de simplification utilisées sont simples : une feuille VRAI sous une porte ET est éliminée, une feuille VRAI sous une porte OU provoque la mise à VRAI de la porte OU, etc.

Exemple :

Reprenons l'exemple ci-dessus en supposant qu'aucune règle d'occurrence du modèle ne provoque *def (P1) = VRAI*. Dans ce cas, si la valeur de la variable *def* est VRAI pour *P1* dans l'état initial du système, la feuille est mise à VRAI et l'arbre est simplifié en conséquence.

L'arbre de défaillance construit permet donc d'expliquer un **état indésirable** du système (état défini par une

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

contrainte de la forme « une variable donnée a une valeur donnée ») par les événements de base (défaillances de composants) pouvant survenir dans le système à partir d'un **état initial** donné. Il s'agit de l'état initial complet, obtenu par une application des règles d'interaction en chaînage avant à partir de l'état initial défini par l'utilisateur, considéré dans tous les cas comme incomplet.

3. Définition formelle du langage Figaro 0

Le langage Figaro d'ordre 1 est complexe et il est donc difficile d'en donner une définition formelle : celle-ci serait d'une longueur déraisonnable, et illisible. En revanche, il est possible de donner une définition formelle simple du langage Figaro d'ordre 0.

Dans la mesure où le langage d'ordre 1 n'est qu'un moyen commode de générer des modèles d'ordre 0 qui sont le point de départ de tout traitement, il est suffisant de définir formellement le langage d'ordre 0.

Dans un premier temps, afin de garder un exposé simple, nous écarterons la possibilité, offerte par le langage Figaro, de définir des lois probabilistes de type Dirac, servant à modéliser des déclenchements déterministes d'événements au bout de temps fixés, ou des lois (comme la loi uniforme sur un intervalle ou la loi triangle) dont la mesure est nulle sur certains intervalles inclus dans $[0, +\infty[$. En effet, la majorité des modèles ne font pas appel à ce type de lois, qui ont un effet sur le comportement qualitatif (existence ou non de certaines séquences de transitions entre états). Grâce à cette restriction, nous n'aurons pas besoin de détailler le comportement temporel de l'automate défini par un modèle Figaro 0 : nous nous contenterons de définir les états vers lesquels l'automate peut évoluer à partir d'un état donné, et si le changement d'état est instantané ou a lieu après un temps non nul. Ce n'est que dans la section 3.4 que nous traiterons des cas particuliers en introduisant un échéancier, une notion relativement complexe et inutile pour les cas courants. En revanche, la distinction entre états instantanés et états non instantanés ou "tangibles" est dans tous les modèles une **nécessité** pour donner au langage un pouvoir de modélisation suffisant. En effet, ces deux types d'états ne sont pas traités de la même manière, ainsi qu'on le verra dans le paragraphe définissant comment l'automate peut évoluer à partir d'un état donné.

3.1. Eléments du modèle

Soit L un ensemble de lois probabilistes, appartenant à l'une des deux catégories suivantes : lois continues sur $[0, +\infty[$, lois discrètes associant des probabilités à un nombre fini de modalités.

Un modèle en langage Figaro 0 est un nuplet $(\mathcal{E}, O, T, I, \sigma, Y_0, V_0)$ composé des éléments suivants :

\mathcal{E} est le produit cartésien $E_1 \otimes E_2 \dots \otimes E_n$ des domaines des composantes de X , un vecteur fini de variables d'état (x_1, x_2, \dots, x_n) dont la valeur définit l'état du modèle à l'instant t . Chaque variable d'état x_i est soit un booléen, soit un entier, soit un réel, soit une variable dite « énumérée » pouvant prendre ses valeurs dans un ensemble E_i fini. X est en fait la concaténation V, Y de deux vecteurs V et Y . V regroupe les variables dites "**essentiels**" et Y les variables dites "**déduites**".

V_0 est la valeur initiale de V , et Y_0 est la valeur dite "de réinitialisation" de Y . **Y est une fonction de V et de Y_0 définie à l'aide de la fonction I décrite plus loin.**

T est l'ensemble des **groupes de transitions** du modèle. Une **transition** est une application de \mathcal{E} dans \mathcal{E} , qui à tout vecteur d'état fait correspondre un autre vecteur d'état, en général obtenu par la modification d'un faible nombre de variables essentielles. Un groupe de transitions, quant à lui, est un couple dont les éléments sont un ensemble de transitions dites "liées", et une loi de probabilité de L . Le groupe de transitions contient une seule transition si sa loi n'est pas de type loi discrète. Si la loi est de type discret, le groupe de transitions contient autant de transitions que la loi a de modalités. Une transition représente un changement local de l'état d'un composant du système modélisé, par exemple l'apparition d'un mode de défaillance. *Un groupe de transitions associé à une loi discrète modélise les diverses issues possibles d'un processus non déterministe considéré comme instantané* ; par exemple, les divers résultats d'un jet de dé, ou le choix entre démarrage et refus de démarrage d'un composant. Par abus de langage, on confondra la notion de groupe de transitions et de transition pour les groupes ne contenant qu'une transition. La notion de groupe de transitions est une originalité du langage Figaro par rapport à l'ensemble des formalismes destinés à décrire des automates non déterministes, à commencer par les réseaux de Petri. Son importance apparaîtra plus loin, lorsque nous décrirons comment l'automate peut évoluer à partir d'un état pour lequel *plusieurs groupes de transitions instantanées* sont applicables.

O est une fonction de \mathcal{E} dans $\mathcal{P}(T)$. En pratique, O est définie par l'ensemble des **règles dites "d'occurrence"**. Ces règles permettent d'associer un ensemble (éventuellement vide) de groupes de transitions à n'importe quel

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

état X du modèle. Les transitions appartenant aux groupes de $O(X)$ sont dites "**valides**" dans l'état X .

I est une fonction de \mathcal{E} dans \mathcal{E} , qui à tout vecteur d'état X **pour lequel I est définie** fait correspondre un autre vecteur d'état. La fonction I est en pratique définie par les **règles dites "d'interaction"** du modèle Figaro (et éventuellement un **système d'équations linéaires**), et cette fonction **dépend éventuellement de l'ordre σ des règles**.

La fonction I est définie comme la composée d'un ensemble fini de fonctions de \mathcal{E} dans \mathcal{E} , notées I_0, I_1, \dots, I_P . Autrement dit, $I(X) = I_P(I_{P-1}(\dots I_0(X) \dots))$.

Les règles d'interaction du modèle sont regroupées en "étapes" (cf. § 7.5), correspondant aux différentes fonctions I_1, \dots, I_P . Quant à I_0 , c'est une fonction particulière de réinitialisation des variables déduites : $I_0(V, Y) = V, Y_0$

3.2. Inférence réalisée dans les règles d'interaction

La fonction correspondant à chaque étape est définie par un **automate déterministe** dont le fonctionnement est celui d'un moteur d'inférence simple. Soit $\{R_k\}$ l'ensemble des règles d'une étape donnée (pour alléger les notations, nous omettons l'indice d'étape). La règle R_k est une fonction de \mathcal{E} dans \mathcal{E} , qui à tout vecteur d'état X fait correspondre un autre vecteur d'état $R_k(X)$. Chaque règle est en pratique composée d'une **condition** calculée par une fonction booléenne de l'état X , permettant son déclenchement, et **d'actions**, qui sont le plus souvent des instructions d'affectation de variables d'état, par des valeurs constantes ou par des valeurs calculées à partir de X . Il existe un autre type d'action possible, employé uniquement pour l'instant dans certaines bases de connaissances pour les systèmes électriques : la résolution d'un système d'équations linéaires, qui permet de calculer une nouvelle valeur de X .

Le moteur d'inférence applique cycliquement les règles (R_k) *ordonnées suivant l'ordre σ* jusqu'à obtenir la même valeur pour X à la fin de deux "tours" successifs des règles.

Appliquer une règle consiste à évaluer sa condition, et, si celle-ci est vraie, réaliser l'action correspondante. En particulier, si la condition n'est pas réalisée, le vecteur X n'est pas modifié.

Ce type de fonctionnement des règles d'interaction permet une grande souplesse de modélisation, notamment pour tout ce qui est propagation de flux dans un système. Il a même permis, dans une première version de l'outil Topase, de calculer les courants et tensions en tout point d'un circuit électrique selon les lois d'Ohm, avant que ce calcul puisse être réalisé de manière bien plus performante par la résolution d'un système d'équations linéaires.

3.3. Sémantique du modèle complet

Les différents éléments d'un modèle Figaro 0 étant décrits, il est facile de définir la sémantique du langage Figaro 0. Cette sémantique équivaut au fonctionnement de l'automate **aléatoire** suivant :

Initialisation :

Un état initial **incomplet** est défini par l'utilisateur par la donnée de V_0 (seule contrainte : respect du domaine de V). Puis calcul de $X_0 = I(V_0, Y_0)$ qui est l'état initial complet du modèle.

Notons au passage que cette façon de définir l'état initial du système permet de résoudre un certain nombre de problèmes épineux pour un modèle complexe : le risque de la définition d'un état incohérent (ex : on a deux composants électriques en série, l'intensité de courant est nulle dans l'un mais pas dans l'autre), voire l'impossibilité pour l'utilisateur de définir l'état initial (ce serait le cas, par exemple, s'il lui fallait donner les courants et tensions en tous points d'un circuit électrique). Ainsi, la définition de l'état initial est facile et sûre pour l'utilisateur : il a peu de variables à initialiser, et ce sont des variables à la signification claire et faciles à déterminer (comme la position d'une vanne, le nombre de composants mis en service à l'instant initial...).

Evolution à partir d'un état courant X :

$O(X)$ définit les groupes de transitions applicables (on dit aussi valides) à partir de X .

- Si cet ensemble est vide, l'état courant est absorbant.
- S'il est réduit à un groupe contenant une seule transition t , l'unique état que peut prendre l'automate, à partir de l'état X , est $I(t(X))$.
- S'il contient plusieurs groupes de transitions, deux cas sont à considérer (la décomposition ci-après constitue une partition de l'ensemble des situations possibles) :
 - $O(X)$ contient **uniquement** des transitions de lois continues. Dans ce cas, l'état X est dit "tangibile" et le

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

prochain état que prendra l'automate sera $I(t(X))$, t étant l'une quelconque des transitions de lois continues (non déterminisme).

- $O(X)$ contient les groupes $G1, G2, \dots, Gn$ de transitions de loi discrète. L'état X est alors dit "instantané", et les transitions de lois continues sont ignorées. Dans ce cas, le prochain état que prendra l'automate sera $I(t1 \circ t2 \circ \dots \circ tk(X))$, expression dans laquelle ti représente un choix quelconque (non déterminisme) de transition dans le groupe $Gi : ti \in Gi$. L'ordre d'application des transitions (autrement dit le choix des indices pour les groupes) n'est pas précisé. Ceci peut produire des problèmes de cohérence, qui sont discutés au chapitre 8.

3.4. Sémantique d'un modèle avec des lois probabilistes quelconques

Il est facile de voir que lorsque l'on a en concurrence dans un modèle deux transitions A et B dont les conditions sont validées en même temps, et avec des lois de Dirac de paramètres $\text{Delai_A} < \text{Delai_B}$, seule la séquence A, B pourra être observée. Les paramètres des lois ont donc une incidence sur le comportement **qualitatif** du modèle, ce qui n'était absolument pas le cas dans les sous-sections précédentes.

Dans le cas où l'on a affaire à des lois probabilistes quelconques, le seul moyen d'expliquer la sémantique du modèle consiste à décrire l'algorithme qui permet de faire une simulation de Monte Carlo à partir de ce modèle. Cet algorithme fait appel aux fonctions définies précédemment pour les règles d'occurrence et d'interaction.

Il repose sur la gestion d'un échéancier : c'est une liste de transitions associées à des temps précis. Formellement, un échéancier est défini comme $S = ((\tau_1, t_1), (\tau_2, t_2), \dots, (\tau_n, t_n))$. Les seconds éléments des couples sont les transitions, et les premiers éléments sont les temps associés. Les événements sont ordonnés par temps croissants : $\tau_1 \leq \tau_2 \leq \dots \tau_n$.

Pour simuler une « histoire » du modèle sur l'intervalle de temps $[0, T_m]$, on utilise l'algorithme suivant :

```
Créer un échéancier vide S
Initialiser la variable qui représente le temps simulé : t = 0
Calculer l'état initial complet  $X = I(V0, Y0)$ 
Tant que  $t \leq T_m$  :
    1 Calculer  $O(X)$  : liste des groupes de transitions valides
    2 Si des groupes de transitions instantanées sont valides :
        tirer au hasard une transition par groupe (en tenant compte des
        lois discrètes) et l'appliquer. X devient X'
        Calculer  $X = I(X')$  pour propager les effets des transitions
        instantanées
        Revenir en 1
    4 Mettre l'échéancier à jour :
        Tirer au hasard les instants de tir des transitions valides qui ne
        figurent pas déjà dans S, et les insérer dans S
        Supprimer de S les transitions qui ne sont plus valides
    5 Trouver la transition  $(\tau_{\min}, t_{\min})$  correspondant à la plus petite date
        > t dans S (s'il n'y en a pas, l'histoire se termine par un état
        absorbant)
    6 Appliquer  $t_{\min}$ . X devient X'
    7 Calculer  $X = I(X')$  pour propager les effets de  $t_{\min}$ 
    8 Faire  $t = \tau_{\min}$  // la date de la transition devient le temps courant
```

En regardant cet algorithme en détail, on s'aperçoit qu'il existe des situations qui demandent plus de précisions.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

De même que plusieurs groupes de transitions instantanées peuvent être valides en même temps, à l'étape 5 de l'algorithme, plusieurs transitions temporisées peuvent tomber exactement au même instant si elles sont définies par des Dirac. On calcule alors X' en cumulant les effets des transitions avant de propager les effets à l'étape 7.

Que se passe-t-il pour une transition dont les paramètres de loi changent, sans qu'il y ait interruption des conditions de validité de la transition ? Plusieurs choix sont défendables, ainsi que cela est expliqué dans l'article [12]. Les choix faits pour le langage Figaro sont les suivants :

- Pour toute autre loi que la distribution de Dirac, une nouvelle date est tirée en fonction des nouveaux paramètres et remplace celle qui était précédemment associée à la transition dans l'échéancier
- Pour la distribution de Dirac, deux choix sont laissés à l'utilisateur (ce choix est à faire dans les outils de calcul)
 - a) la date présente dans l'échéancier est maintenue.
 - b) la date présente dans l'échéancier est remplacée par la date du changement de paramètre + la nouvelle valeur du paramètre.

Ces options ne sont pas forcément adaptées à ce que l'utilisateur veut modéliser. Il a toujours la possibilité de faire ses propres calculs en utilisant la variable DATE_COURANTE et les fonctions mathématiques associées aux distributions de probabilité pour calculer dans les règles d'interaction un délai à mettre dans une distribution de Dirac (avec l'option b) mais c'est un peu compliqué.

Heureusement, dans le cas très fréquent de la loi exponentielle, il n'y a pas de question à se poser : comme cette loi modélise l'absence de mémoire, il n'y a qu'une seule option possible, qui consiste à recalculer la date de la transition (c'est ce qui est spécifié pour Figaro).

Les lois « à mémoire »

Imaginons qu'un objet soit mis sur un tapis roulant à t=0. D'après la vitesse du tapis, on peut calculer qu'il arrivera à destination à la date t1. On peut donc modéliser son arrivée par une transition avec une distribution de Dirac. Mais si le fonctionnement du tapis est interrompu par moments ? On voit ici tout l'intérêt d'une transition munie d'une loi à mémoire, qui se déclenchera lorsque la *somme des temps pendant lesquels la transition était valide* atteint t1.

```

TYPE objet_sur_tapis ;
PARAMETRE_LOI t1 PAR DEFAULT 10 ;
ATTRIBUT en_mouvement DOMAINE BOOLEEN PAR_DEFAULT VRAI ;
OCCURRENCE
SI en_mouvement (* cet attribut peut être alternativement à VRAI et FAUX en
fonction de l'état du système *)
IL_PEUT_SE_PRODUIRE
TRANSITION arrivee LOI T_C_M (t1)
PROVOQUE en_mouvement <-- FAUX ;

```

Ce cas facile à comprendre peut être généralisé à toute loi **autre que la loi exponentielle** (qui par définition est sans mémoire). C'est pourquoi tous les noms de lois sauf la loi EXP ont un « double » désignant la loi à mémoire correspondante, obtenu en ajoutant le suffixe « _M ». Le fonctionnement de toutes ces lois est le même : la transition est tirée lorsque la somme des temps pendant lesquels la transition est valide atteint le nombre tiré au hasard lors de sa première validation. Lors du tir, le compteur est remis à zéro. C'est ainsi que l'on peut modéliser facilement un moteur qui ne s'use que lorsqu'il est en marche par une loi de Weibull à mémoire.

4. Les éléments lexicaux du langage Figaro

Ce chapitre décrit les différentes règles de base à suivre pour écrire un modèle Figaro correct du point de vue lexicographique en Figaro. Le présent document ne décrit que la version française de Figaro. Pour la version anglaise, il faut se référer au document compagnon qui décrit le détail de toute la syntaxe, en français et en anglais.

4.1. L'alphabet

L'ensemble des caractères de la table ASCII (7 bits) sont acceptés. Les caractères accentués de la langue française ne sont pas autorisés par l'analyseur syntaxique (sauf dans les commentaires).

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Voici quelques définitions de base :

- Les **séparateurs** de mots sont : l'espace, la tabulation, le retour chariot et le saut de page. Ils peuvent apparaître en nombre quelconque.
- Une **lettre** est un des caractères suivants : a...z A...Z _ (souligné).
- Un **chiffre** est un des caractères suivants : 0...9.
- La **fin d'une description de champ** et la fin de **description d'une règle** sont représentées par un point-virgule ';'
- Le point-virgule est aussi utilisé comme séparateur de haut niveau pour :
 - * marquer la fin de la description du "champ" TYPE (cf. TYPE),
 - * séparer les étapes définies pour une base de connaissances (cf. ORDRE_DES_ETAPES),
 - * séparer les groupes définis pour une base de connaissances (cf. NOMS_DES_GROUPES),
 - * séparer les systèmes d'équations définis pour une base de connaissances (cf. NOMS_DES_SYSTEMES).
- Le séparateur des éléments d'une **liste** est la virgule ','. Les listes sont principalement utilisées :
 - * pour définir plusieurs actions d'une même règle (cf. Action),
 - * pour attribuer un ensemble de valeurs à la facette EDITION d'un champ (cf. EDITION),
 - * pour définir un ensemble d'identificateurs pour une même ETAPE (cf. ETAPE dans Notions avancées).
- Les séparateurs des éléments d'une **énumération** sont l'espace ou la tabulation. Les énumérations sont principalement utilisées :
 - * pour définir les valeurs d'un DOMAINE énuméré (cf. DOMAINE),
 - * pour définir l'ensemble des GROUPES auxquels appartiennent une même règle (cf. GROUPE dans Notions avancées),
 - * pour définir, dans une base de faits, l'ensemble des objets connectés à un même objet dans une interface,
 - * pour définir, dans le cas d'un héritage multivalué, l'ensemble des types dont hérite un même TYPE (cf. SORTE_DE et Héritage).
- Une **chaîne de caractères** s'écrit entre guillemets (") et peut contenir des espaces. Les chaînes de caractères sont uniquement utilisées pour définir le libellé d'un champ (cf. LIBELLE).
- Une **valeur symbolique** (ou alphanumérique) s'écrit entre apostrophes (') et peut contenir des espaces. Les valeurs symboliques sont uniquement utilisées pour définir les différentes valeurs d'un domaine énuméré (cf. DOMAINE).
- Un **commentaire** commence par '(*' et finit par '*)'. Tout caractère est permis entre les deux délimiteurs. Les commentaires peuvent être imbriqués.

Exemple :

```

TYPE vanne
    SORTE_DE composant; (*le ; désigne la fin du champ TYPE*)
ATTRIBUT
    etat    EDITION    VISIBLE, MODIFIABLE, NON OBLIGATOIRE
              (* liste *)
              LIBELLE    "etat de la vanne"
              (* chaîne de caractères *)
              DOMAINE    'ouvert' 'ferme' 'inconnu'
              (* enumeration de valeurs symboliques*)

```


EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

PAR_DEFAULT    'ouvert' ;    (*le ; désigne la fin de
                                description du champ etat*)
(* commentaire
   (* commentaire imbriqué *)
*)

```

Remarque : Il est interdit de mettre un retour chariot dans une valeur symbolique (entre '). En revanche, le retour chariot est autorisé dans les commentaires et dans les chaînes de caractères (entre ").

Remarque : La taille maximale d'une chaîne ou d'une valeur symbolique est de 200 caractères.

4.2. Les valeurs numériques

- Les **constantes booléennes** sont : VRAI et FAUX (en majuscule).

- Les **constantes numériques** (ou nombres) sont de deux types : entier ou réel. Un entier est un nombre à plusieurs chiffres (précédé du signe '-' si besoin). Un réel a la forme suivante :

```
<partie entière>.<partie réelle>e<puissance de dix>
```

La puissance de dix est un entier positif ou négatif. La partie entière est un entier positif ou négatif. La partie réelle est un entier sans signe.

Exemples de constantes numériques:

```

25          (* un entier *)
-135        (* un entier négatif *)
256.012     (* un réel *)
-58.5e-5    (* un réel négatif avec une puissance de dix *)

```

4.3. Les identificateurs

Un identificateur permet de donner un nom à un type, un objet, un champ, un opérateur spécifique ou une règle.

On distingue deux possibilités pour décrire un identificateur :

- * Soit par une suite de lettres et de chiffres commençant toujours par une lettre et ne contenant pas d'espaces (on utilisera souvent le caractère _ pour assurer la lisibilité de l'identificateur).
- * Soit par une suite de lettres et de chiffres encadrée par des barres verticales '|'. Dans ce cas, le nom de l'identificateur peut commencer par un chiffre et il peut contenir des espaces et des tabulations (le retour chariot est interdit).

La taille maximale d'un identificateur est de 200 caractères.

Exemples :

```

ABCDEF (* autorisé *)
abcdef (* autorisé. Ces deux symboles sont différents*)
A1234  (* autorisé *)
12ABCD (* non autorisé !!! Sera lu comme 12 ABCD *)
|12 AbDEF| (* autorisé *)

```

Remarque : Pour assurer la lisibilité du modèle Figaro, il est préférable de ne pas utiliser des identificateurs identiques à des mots clés, et de ne pas utiliser deux identificateurs identiques pour représenter des données différentes.

Exemple à ne pas suivre :

```

TYPE composant ;
ATTRIBUT
    attribut DOMAINE ENTIER ; (* DECONSEILLE : attribut (le champ)
                                et ATTRIBUT (le mot-clé) *)

```

4.4. Les mots-clés

Les **mots-clés de Figaro** doivent être écrits en majuscules, sinon ils seront considérés comme étant des

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

identificateurs de variables.

4.4.1. Les opérateurs

sens Figaro	terme Français	terme Anglais
opérateur d'assignation	<--	<--
opérateur booléen égalité	=	=
assignation en Figaro 0		
opérateur booléen différent	<>	<>
opérateur booléen inférieur strictement	<	<
opérateur booléen inférieur ou égal	<=	<=
opérateur booléen supérieur strictement	>	>
opérateur booléen supérieur ou égal	>=	>=
opérateur de multiplication	*	*
opérateur de division	/	/
opérateur d'addition	+	+
opérateur de soustraction	-	-
modulo (a % b désigne le reste de la division entière de a par b)	%	%
puissance (a**b désigne a puissance b)	**	**
négation booléenne	NON	NOT
opérateur booléen ET	ET	AND
opérateur booléen OU	OU	OR

4.4.2. Les mots-clés

Le tableau ci-dessous est une liste exhaustive triée par ordre alphabétique. La longueur de cette liste ne doit pas faire peur, car elle contient pêle-mêle des mots utilisés très fréquemment et faisant vraiment partie du cœur du langage, et des mots d'utilisation très spécifique, en particulier ceux servant à définir des lois de probabilité et leurs paramètres. Afin d'aider le lecteur, le code couleur suivant est utilisé :

Bleu : fonctions mathématiques, par exemple COS pour cosinus ;

Violet : noms des lois de probabilités, par exemple WEI ou WEIBULL pour la loi de Weibull ;

Vert : noms de paramètres de lois de probabilité.

Tableau de tous les mots-clés Figaro (hors opérateurs définis par des symboles) :

ALORS	FRECHET_M	MODELE_REMPLACE	RESOUDRE_SYSTEME
ALPHA	FREQUENCE	MODELE_TA	ROLE
ATTRIBUT	GAMMA	MODELE_TPE	SI
AU_MOINS	GAMMA_NON_DETECTION	MODELE_WB	SIN
BETA	GAMMA_RECONFIG	MODIFIABLE	SINON
BOOLEEN	GAMMA_TEST	MOI_MEME	SOIT
CARDINAL	GENRE	MU	SOMME
CEIL	GROUPE	NINT	SORTE_DE
CONCEPTION	GUMBEL	NOMS_DES_GROUPE	SQRT
CONDITION	GUMBEL_M	NOMS_DES_SYSTEMES	STANDARD
CONFIGURATION	IL_EXISTE	NON	SUPPRIMER
CONSTANTE	IL_PEUT_SE_PRODUIRE	NORMAL	SYSTEME_EQUATIONS
COS	INCLUS_DANS	NORMAL_M	T_C
CYCLE	INDISPONIBILITE	OBJET	T_C_M
CYCLE_M	INFINI	OBJET_SYSTEME	T_INIT_TEST
DATE_COURANTE	INS	OBLIGATOIRE	T_INTER_TEST

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

règles génériques modélisant le comportement d'un objet quelconque d'un type.

Un **quantificateur logique** se définit comme un symbole qui permet d'utiliser une variable définie sur un ensemble particulier. Il existe deux types de quantificateurs en Figaro :

- * le quantificateur universel : $\forall x \in \text{ensemble } E,$
- * le quantificateur existentiel : $\exists x \in \text{ensemble } E.$

De même, les **opérateurs numériques complexes** somme, produit, minimum et maximum permettent d'effectuer des calculs sur un ensemble de variables :

- $\sum_{x \in \text{ensemble}} f(x)$
- $\prod_{x \in \text{ensemble}} f(x)$
- même principe pour le minimum et maximum.

En Figaro, les quantificateurs et les opérateurs numériques complexes s'utilisent uniquement sur des **ensembles d'objets**. Le quantificateur (ou l'opérateur) sera utilisé avec une **variable liée** représentant un objet quelconque de l'ensemble.

Exemple

```

TYPE pompe;
ATTRIBUT
    panne DOMAINE BOOLEEN ;

TYPE pompe_normale SORTE_DE pompe;

TYPE pompe_secours SORTE_DE pompe;
ATTRIBUT
    demande_demarrage DOMAINE BOOLEEN ;
INTERFACE
    secourue GENRE pompe_normale CARDINAL 1 JUSQUA INFINI;
INTERACTION
SI QOSOIT x UNE secourue ON_A panne DE x = VRAI
ALORS demande_demarrage <-- VRAI ;

```

Cette règle d'interaction se lit :

Soit o un objet quelconque de type pompe_secours,

Si $\forall x \in \{\text{pompes normales secourues par } o\}$, x est en panne, (x est une variable liée)

Alors, il y a demande de démarrage de o.

Pour utiliser des quantificateurs ou des opérateurs numériques complexes en Figaro, il est donc nécessaire de définir des ensembles d'objets. Dans la suite de cette partie, nous allons voir qu'il existe deux moyens principaux de définir un ensemble d'objets en Figaro.

5.1.2.2. Utilisation d'un champ d'interface pour définir un ensemble

Un premier moyen de définir un ensemble d'objets en relation avec un objet d'un type est d'utiliser un **champ interface** du type.

Exemple 1 :

```

TYPE composant ;
ATTRIBUT
    relie DOMAINE BOOLEEN;
INTERFACE
    amont GENRE composant ;

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

INTERACTION

```
SI IL_EXISTE x UN mont TEL_QUE relie DE x = VRAI
ALORS relie <-- VRAI ;
```

Si *o* est un objet quelconque de type composant, *mont* désigne l'ensemble des composants reliés à *o* par la relation *mont*.

Dans cet exemple, l'ensemble de définition de la variable liée au quantificateur 'IL_EXISTE' est défini par **l'identificateur d'un champ interface**. Ce moyen de définir un ensemble d'objets avec un champ interface est le moyen le plus couramment utilisé en Figaro. Les deux exemples suivants illustrent des définitions plus complexes d'ensembles.

Exemple 2 :

```
TYPE X;

TYPE Y;
INTERFACE
  i1 GENRE X;

TYPE Z;
INTERFACE
  i2 GENRE Y CARDINAL 1;
INTERACTION
  SI IL_EXISTE x UN i1 DE i2 ....;
```

Si *o* est un objet quelconque de type Z, l'expression *i1 DE i2* désigne l'ensemble des objets de type X en relation *i1* avec l'**unique** objet de type Y en relation *i2* avec *o*.

Dans cet exemple, l'ensemble de définition de la variable liée au quantificateur IL_EXISTE est défini par **'identificateur de champ interface' DE 'ensemble de cardinal 1'**.

Exemple 3 :

```
TYPE X;

TYPE Y;
  i1 GENRE X;

TYPE Z;
INTERFACE
  i2 GENRE Y;
INTERACTION
  SI IL_EXISTE y UN i2 TEL_QUE (IL_EXISTE x UN i1 DE y TEL_QUE...)....;
```

Si *o* est un objet quelconque de type Z :

- * l'expression *i2* désigne l'ensemble des objets de type Y en relation *i2* avec *o*,
- * l'expression *i1 DE y* désigne l'ensemble des objets de type X en relation *i1* avec *y*, un élément de l'ensemble précédent.

Dans cet exemple, l'ensemble de définition de la variable liée au deuxième quantificateur IL_EXISTE est défini par **'identificateur de champ interface' DE variable liée**.

5.1.2.3. Utilisation d'un type pour définir un ensemble

Il existe un autre moyen d'accéder à un ensemble d'objets. En effet, il est parfois intéressant en Figaro de pouvoir adresser tous les objets d'un **type particulier**. C'est pourquoi Figaro permet d'utiliser un nom de type pour définir un ensemble.

Pour définir un ensemble à l'aide d'un nom de type, on utilise une syntaxe de la forme :

$\forall x$ UN OBJET DE_TYPE 'nom de type'

Exemple :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

TYPE composant ;

TYPE pompe ;
INTERACTION
SI QQS0IT x UN OBJET DE_TYPE composant ON_A ...
ALORS ... ;

```

L'expression *UN OBJET DE_TYPE composant* désigne l'ensemble des objets d'un système instance du type composant.

5.1.2.4. Limite de Figaro

Les ensembles d'objets manipulés en Figaro sont des **ensembles constants**.

En effet, comme nous l'avons évoqué précédemment, les liens entre les objets d'un système (champs interface des objets) n'évoluent pas au cours de la vie du système. Autrement dit, les règles Figaro ne pourront pas ajouter un objet dans une interface ou enlever un objet d'une interface.

De même, il est impossible en Figaro d'ajouter au système un objet instance d'un type (ou de supprimer un objet instance d'un type).

Ainsi, que l'on utilise des champs interface ou des noms de types pour définir un ensemble d'objets, l'ensemble sera constant au cours de la vie du système.

5.1.2.5. Restriction d'un ensemble

Il arrive parfois que l'on veuille **restreindre** l'ensemble des objets d'une interface ou l'ensemble des objets d'un type. Ainsi, il est possible en Figaro d'utiliser des contraintes pour réduire le nombre d'objets de l'ensemble.

Il existe deux contraintes optionnelles en Figaro introduites par les mots-clés `DE_TYPE` et `VERIFIANT`. Ces deux contraintes peuvent être utilisées ensemble ou séparément. Elles s'appliquent sur l'ensemble de définition du quantificateur ou de l'opérateur complexe.

La contrainte `DE_TYPE` :

Cette contrainte permet de restreindre le type des objets de l'ensemble. **Cette contrainte ne peut être utilisée que lorsque l'ensemble est défini avec un champ interface.** En effet, les objets d'une interface sont tous d'un type particulier précisé lors de la déclaration par le mot-clé `GENRE`. Cependant, il peut être intéressant de réduire cet ensemble d'objets à un sous-type. Prenons comme exemple, la hiérarchie de classes suivante :

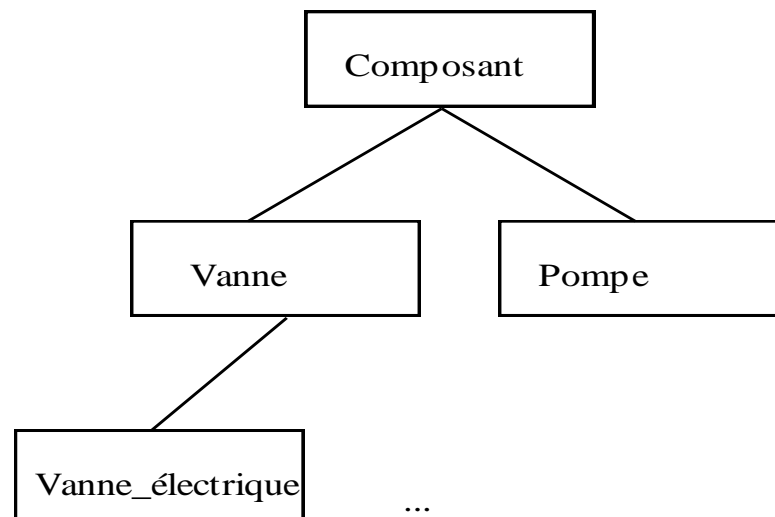


Figure 3. Exemple de hiérarchie de classes

Composant est un type générique hérité par les types *vanne* et *pompe*. *Vanne_électrique* est une classe qui spécifie le type *vanne*. Aussi, si le type de l'interface est *composant*, toute instance des 4 types ci-dessus peut être incluse dans l'interface. En effet, puisque ces types héritent de *composant*, ce sont des composants. L'intérêt de la contrainte `DE_TYPE` apparaît alors.

L'exemple suivant met en évidence l'utilisation de cette contrainte : le type *vanne* est un sous type de *composant*

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

et les objets interfacés avec le type *pompe* sont d'un type *composant*. Dans la règle, on veut uniquement s'intéresser aux objets de type *vanne* qui sont interfacés :

```

TYPE composant ;
TYPE vanne SORTE_DE composant;
TYPE pompe SORTE_DE composant;
INTERFACE
    amont GENRE composant ;
INTERACTION
SI QQSOIT x UN amont DE_TYPE vanne ON_A ...
ALORS ... ;

```

Si o est un objet quelconque de type pompe l'expression *amont DE_TYPE vanne* désigne l'ensemble des objets de type vanne (sous-type de composant) en relation amont avec o.

La contrainte VERIFIANT :

Cette contrainte permet de restreindre les objets d'un ensemble d'après une condition. Le principe est simple : l'ensemble est réduit au sous-ensemble ne contenant que les objets vérifiant cette condition.

La condition suivant le mot-clé VERIFIANT doit donc être une expression booléenne (qui renvoie VRAI ou FAUX).

Pour que l'ensemble défini soit un ensemble d'objets constant, l'expression suivant le mot-clé VERIFIANT doit être une **expression constante**, c'est-à-dire ne faisant appel qu'à des champs constants.

La condition peut contenir un nombre quelconque d'expressions booléennes (y compris complexes) reliées par des opérateurs logiques (OU, ET ou NON).

Dans l'exemple suivant, on ne s'intéresse qu'aux objets de l'interface dont le débit dépasse une certaine valeur :

```

TYPE pompe ;
CONSTANTE
    debit DOMAINE REEL;

TYPE composant ;
INTERFACE
    amont GENRE pompe ;
INTERACTION
SI QQSOIT x UN amont VERIFIANT debit DE x > 3000 ON_A ...
ALORS ...

```

Si o est un objet quelconque de type composant, *amont VERIFIANT debit DE x > 3000* désigne l'ensemble des pompes reliées à o par la relation amont, dont le débit est strictement supérieur à la valeur 3000.

5.1.2.6. Cas des ensembles vides

Il est très important de remarquer que, pour assurer une souplesse maximale d'utilisation du langage, les expressions faisant appel à des quantificateurs ou opérateurs numériques complexes doivent donner un résultat (plutôt qu'une erreur) même quand l'ensemble sur lequel portent ces constructions est vide.

Pour les opérateurs QQSOIT, IL_EXISTE, SOMME et PRODUIT le principe est le même : le résultat est l'élément neutre de l'opération, ce qui est cohérent avec le fait que si l'on passe de 0 à un élément dans l'ensemble, on passe de l'élément neutre à la valeur correspondant à l'unique élément de l'ensemble.

Pour le minimum et le maximum si l'on appliquait le même principe, il faudrait prendre respectivement $+\infty$ et $-\infty$. Les valeurs infinies n'étant pas représentables, on a choisi arbitrairement de rendre le résultat 0 dans les deux cas. En résumé, on a donc les valeurs suivantes :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

QQSOIT x UN elt d'un ensemble vide ON_A...	VRAI
IL_EXISTE x UN elt d'un ensemble vide TEL_QUE	FAUX
SOMME POUR_TOUT x UN elt d'un ensemble vide DES_TERMES	0
PRODUIT POUR_TOUT x UN elt d'un ensemble vide DES_TERMES	1
MINIMUM POUR_TOUT x UN elt d'un ensemble vide DES_TERMES	0
MAXIMUM POUR_TOUT x UN elt d'un ensemble vide DES_TERMES	0

Tableau 1 : valeurs retournées par les opérateurs complexes appliqués à des ensembles vides

Il faut donc prendre garde à ces cas particuliers des ensembles vides, et pour éviter tout comportement imprévu d'un modèle, utiliser l'une des techniques suivantes :

- S'assurer que les modèles de systèmes ne contiendront pas ce cas particulier en précisant par exemple que le cardinal d'une interface est au moins égal à 1 (cf. 2.4.3.2).
- Compléter l'expression de façon à prévoir explicitement le cas où l'ensemble est vide. Dans l'exemple suivant, on ne veut mettre à vrai l'effet relié du type que l'on décrit que s'il a des objets dans son interface amont et si ils sont eux-mêmes tous reliés. Dans le cas particulier où il n'y a pas d'amont, relié doit rester à faux.

Exemple :

INTERACTION

```
SI ( CARDINAL DE amont <> 0 ) ET
  ( QQSOIT x UN amont ON_A relie DE x = VRAI )
ALORS relie <-- VRAI ;
```

5.1.3. Accès à un champ de type ou d'objet

5.1.3.1. Accès aux champs du type décrit

Lorsque, pour décrire un TYPE, on a besoin d'écrire une expression utilisant un champ des objets du type, il suffit de nommer ce champ.

```
TYPE pompe;
ATTRIBUT
  etat DOMAINE 'arret' 'marche';
INTERACTION
  SI etat = 'arret'
  ALORS ....;
```

Si o est un objet quelconque de type pompe, l'expression *etat = 'arret'* signifie *etat de o = 'arret'*.

5.1.3.2. Accès aux champs d'une variable liée à un ensemble d'objets

Pour accéder aux objets d'un ensemble, on utilise toujours une variable liée qui représente un objet de l'ensemble (cette variable prendra successivement toutes les valeurs de l'ensemble). Pour accéder aux champs des objets représentés par la variable liée, on utilise l'opérateur **DE** ou des parenthèses.

Syntaxe :

```
champ DE variable_liee
champ (variable_liee)
```

Le mécanisme de l'opérateur est simple. Pour chaque nom d'objet substitué à la variable, il renvoie la valeur du champ de l'objet. Si le champ n'existe pas, il provoque une erreur.

```
TYPE pompe;
INTERFACE
  secours GENRE composant;
ATTRIBUT
  etat DOMAINE 'marche' 'arret';
INTERACTION
  SI IL_EXISTE x UN secours TEL_QUE etat DE x = 'arret'
```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

ALORS;

5.1.3.3. Accès au champ d'un objet défini par un ensemble de cardinal 1

Lorsque l'on veut accéder à l'objet d'un ensemble de cardinal 1, l'utilisation d'une variable liée n'est pas nécessaire. On accède à l'objet en nommant simplement l'ensemble, comme *secours* dans l'exemple suivant :

```

TYPE pompe;
INTERFACE
    secours GENRE composant CARDINAL 1;
ATTRIBUT
    etat DOMAINE 'marche' 'arret';
INTERACTION
    SI etat DE secours = 'arret'
    ALORS .....;

```

5.1.3.4. Accès aux champs d'un objet nommé

Dans une base de faits (description du système), pour accéder aux champs d'un objet nommé on utilise l'opérateur **DE**.

Syntaxe :
 champ **DE** nom_objet
 champ (nom_objet)

Cet opérateur permet d'accéder à un champ d'un objet grâce à son nom. Le mécanisme de l'opérateur est simple :

- Il vérifie que l'objet est bien défini dans la base de faits.
- Si l'objet existe, il renvoie la valeur du champ de l'objet en question. Si le champ n'existe pas, il provoque une erreur.
- Si l'objet n'existe pas, l'opérateur provoque une erreur.

Exemple :
 debit **DE** v1

5.1.3.5. Accès aux variables globales

C'est un cas particulier de ce qui est décrit au paragraphe précédent. La grande différence entre les champs des objets système et les autres champs d'objets est que l'on peut faire directement référence à eux dans tous les **types** du modèle.

```

TYPE global ;
ATTRIBUT
    reparable DOMAINE BOOLEEN PAR_DEFAULT VRAI;

OBJET_SYSTEME parametres_globaux EST_UN global ;

TYPE pompe;
ATTRIBUT
    etat DOMAINE 'marche' 'arret' 'panne';
OCCURRENCE
    SI etat = 'panne' ET reparable(parametres_globaux)
    IL_PEUT_SE_PRODUIRE REPARATION.....;

```

5.2. Expressions simples

On entend par expressions simples, des expressions sans opérateurs complexes spécifiques au langage Figaro. Rappelons certaines définitions :

1. Une expression est une combinaison **d'expressions élémentaires**.

Exemple d'une expression :
 a = (b + d) * ((r + z) >= 3.5 ET NON q)

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

2. D'une façon générale, une expression élémentaire comporte un opérateur et un ou deux opérandes.

Exemples d'expressions élémentaires :

```
b + d
r + z
NON q
a = e
```

3. Une expression élémentaire est dite numérique si l'opérateur est arithmétique. Sinon, l'expression est dite logique.

Exemples d'expressions numériques :

```
b + d
r + z
```

Exemples d'expressions logiques :

```
NON q
a = e
```

4. On appelle *domaine* d'un opérande **l'ensemble des valeurs possibles** de l'opérande. En Figaro, il existe 4 domaines possibles : ENTIER, REEL, BOOLEEN et énuméré. Si un champ se voit affecter une valeur en dehors de son domaine, Figaro provoque une erreur.

Exemples de champs avec domaine :

TYPE pompe ;

CONSTANTE

a **DOMAINE ENTIER**

PAR_DEFAULT 3 ;

b **DOMAINE REEL**

PAR_DEFAULT -12e-3 ;

c **DOMAINE BOOLEEN**

PAR_DEFAULT VRAI ;

d **DOMAINE** 'marche' 'arret' (* domaine énuméré *)

PAR_DEFAULT 'arret' ;

Nous allons étudier les différentes opérations disponibles en Figaro.

5.2.1. Opérations arithmétiques

Opérateurs :

Les opérateurs arithmétiques disponibles sont les opérateurs classiques :

```
+ addition
- soustraction
* multiplication
/ division
** exponentiation
% modulo
```

Opérandes :

Ces opérateurs sont binaires, c'est-à-dire qu'ils nécessitent deux opérandes. Ces opérandes sont soit des entiers, soit des réels. L'opérateur de soustraction est unaire lorsque l'opérande à gauche est un opérateur ou qu'il n'y a pas d'opérande à gauche (par exemple, -3). Dans ce cas, il n'agit que sur l'opérande de droite.

Exemples :

```
5 - 15          (* opérateur - binaire *)
5 + -15        (* opérateur - unaire : le résultat est le même *)
1e-4 * 123
5 % 3          (* 5 modulo 3 : le résultat est 2 *)
```

Domaines :

Lorsque deux opérandes n'ont pas le même domaine dans une expression arithmétique, le résultat de l'expression est une valeur qui a pour domaine le plus "grand" domaine des deux opérandes sachant que :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

domaine(réel) > domaine(entier) > domaine(booléen)

Ainsi, le domaine de l'expression suivante :

`1.8e-4 * 123`

est réel car 1.8e-4 est un réel et 123 est un entier. Donc, l'expression est du plus grand domaine, soit réel.

Il peut être intéressant d'utiliser des opérandes booléens dans une opération arithmétique. Dans ce cas, les valeurs booléennes sont converties en valeurs entières (ou réelles le cas échéant) :

VRAI est remplacé par **1**
FAUX est remplacé par **0**

Ainsi :

`VRAI * (5 > 6) * 3`
est égal à (à cause de *, on convertit VRAI)
`1 * FAUX * 3`
est égal à (à cause de *, on convertit FAUX)
`1 * 0 * 3`

La transformation d'un booléen en entier ou réel est particulièrement utile pour définir des expressions compliquées à l'aide de « fonctions indicatrices », une astuce bien connue des mathématiciens.

Par exemple, l'expression suivante permet de déterminer le montant de l'amende à payer suivant que l'on a commis une petite_infraction (17€), une infraction_moyenne (100€), ou une infraction ni petite ni moyenne (1000€) :

`17 * petite_infraction + 100 * infraction_moyenne +
1000 * (NON (petite_infraction OU infraction_moyenne))`

5.2.2. Opérations booléennes

Les opérateurs booléens (c'est-à-dire qui retournent un résultat booléen VRAI ou FAUX) sont divisés en deux catégories : les opérateurs de comparaison et les opérateurs logiques.

Opérateurs de comparaison :

= égal
<> différent
< inférieur
<= inférieur ou égal
> supérieur
>= supérieur ou égal

Opérandes :

Les opérateurs de comparaison sont binaires. Leurs opérandes sont soit des entiers, soit des réels, soit uniquement pour les opérateurs = et <>, des valeurs symboliques (entre '), des booléens ou des objets.

Exemples :

`5 >= 6 (* toujours FAUX *)`
`a < 12.4 (* a ne peut être qu'entier ou réel *)`
`etat = 'arret' (* VRAI ou FAUX selon la valeur de la variable etat*)`

(* Dans l'exemple suivant, x est une variable liée représentant un objet contenu dans l'interface amont. On veut tester si x est différent d'un objet système appelé source *)
... x **UN** amont **VERIFIANT** x <> source ...

Domaines :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Le résultat est toujours booléen. Si les opérandes ont des domaines différents, le domaine de l'opérande de plus petit domaine sera transformé en domaine de l'opérande de plus grand domaine uniquement si les domaines sont ENTIER ou REEL. Sinon, Figaro provoquera une erreur.

Exemple :

```
5 >= 6.2e-3      (* 5 est transformé en réel *)
```

La comparaison entre des opérandes symboliques et numériques est interdite. Aucune conversion ne peut être effectuée.

Exemples :

```
'arrêt' < 'marche'  (* interdit *)
'arrêt' < 5          (* interdit *)
```

Opérateurs logiques :

OU
NON
ET

Opérandes :

Seul l'opérateur de négation est unaire (il ne prend qu'un seul opérande à droite). Les autres sont binaires. Les opérandes ne peuvent être que des booléens. Chacun de ces opérateurs renvoie un booléen.

Exemples :

```
rupture OU court_circuit
(* les variables doivent être booléennes *)
NON ouvert
```

Domaines :

Le calcul des domaines ne pose aucun problème car tout est booléen. Il est donc clair qu'une opération logique entre des opérandes non booléens provoquera une erreur. En effet, Figaro ne peut effectuer de conversion d'un non booléen vers un booléen.

Exemples :

```
a < 10 OU 123 (* interdit *)
a < 10 ET b > 4  (* OK *)
rupture OU NON ( a < 13)
(* OK si rupture est une variable booléenne
et a une variable entière ou réelle *)
```

5.2.3. Priorités entre les opérateurs

Le tableau suivant présente les priorités d'évaluation qui s'appliquent sur les opérateurs. Un calcul d'expression en Figaro est effectué dans l'ordre des priorités des opérateurs. C'est à dire que Figaro commencera par évaluer les expressions élémentaires avec des opérateurs plus prioritaires jusqu'à évaluer les expressions avec les opérateurs moins prioritaires. En cas d'égalité sur les priorités, la priorité est à l'opérateur situé le plus à gauche.

**	Plus prioritaire
* /	
+ -	.
= > >= < <= <>	.
NON	.
ET	.
OU	Moins prioritaire

L'utilisation des parenthèses permet de gérer l'ordre d'évaluation de l'expression. Mieux vaut en mettre trop que pas assez et se tromper sur la signification d'une expression !

Exemples :

```
10 * a - 3 / 2
(* est évaluée comme (10*a) - (3/2) *)
```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

NON p OU a <= 3 * 2
(* est évaluée comme (NON p) OU (a <= (3*2)) *)
2/3*4
(* est évalué comme (2/3)*4 *)
2-3+4
(* est évalué comme (2-3)+4 *)

```

5.3. Les quantificateurs et « itérateurs »

Une grande force du langage Figaro est sa capacité à définir des modèles très génériques grâce à des expressions qui sont valables sur des ensembles d'objets dont le nombre d'éléments n'est pas connu avant l'instanciation sur un système particulier. C'est possible grâce aux quantificateurs et « itérateurs » (ce néologisme désigne un opérateur qui joue un peu le même rôle qu'une boucle dans un langage de programmation).

Comme nous l'avons déjà vu, un **quantificateur logique** se définit comme un symbole qui permet d'utiliser une variable définie sur un ensemble d'objets. En Figaro, on trouve les quantificateurs correspondant aux mots clés suivants : IL_EXISTE, IL_EXISTE AU_MOINS et QQSOIT. Les quantificateurs s'utilisent dans des expressions booléennes définissant les **conditions** de règles.

Lorsqu'on a besoin de définir une **action** sur tous les éléments d'un ensemble, on utilisera l'« itérateur » POUR_TOUT.

Enfin, il arrive (assez rarement il est vrai) qu'on ait besoin de créer (lorsqu'on instanciera un modèle à l'ordre 0) un ensemble de règles portant sur les objets d'un ensemble, voire sur tous les nuplets d'objets que l'on peut constituer en faisant le produit cardinal de plusieurs ensembles. Pour ce faire, on utilisera l'« itérateur » SOIT.

5.3.1. Le quantificateur IL_EXISTE

Syntaxe :

```

IL_EXISTE variable
    UN | UNE      nom_d_ensemble
    [ DE_TYPE     type ]
    [ VERIFIANT   condition ]
    TEL_QUE | TELLE_QUE expression_bouleanne

```

Cet opérateur correspond au quantificateur \exists de la logique des prédicats. Il permet de tester si **au moins un** objet de l'ensemble vérifie une condition particulière (facette TEL_QUE ou TELLE_QUE) qui est représentée par une expression booléenne :

```
IL_EXISTE x UN <ensemble d'objets> TEL_QUE <condition>
```

Le mécanisme de cet opérateur est le suivant :

- Si l'ensemble est non vide, on évalue l'expression booléenne de la condition TEL_QUE pour chaque objet de l'ensemble. Si au moins un résultat est VRAI (OU logique entre les résultats), le résultat de l'opérateur est VRAI. Sinon (tous les résultats sont FAUX), le résultat de l'opérateur est FAUX.
- Si l'ensemble est vide, le résultat est FAUX.

L'expression booléenne peut être une combinaison d'expressions booléennes simples ou complexes.

Remarque : La variable d'ensemble est liée à l'opérateur. Elle peut donc être utilisée uniquement :

- dans les contraintes pour restreindre l'ensemble de définition du quantificateur,
 - dans la condition TEL_QUE.
- mais pas dans le reste de la règle.

Exemple :

```

TYPE vanne ;
CONSTANTE
    debit DOMAINE REEL ;
ATTRIBUT
    position DOMAINE 'ouvert' 'ferme' PAR_DEFAUT 'ouvert' ;

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

TYPE pompe ;
INTERFACE
    montant GENRE vanne ;
INTERACTION
SI IL_EXISTE x UN montant
        TEL_QUE (debit DE x > 1000 ET position DE x = 'ouvert')
ALORS ...

```

L'exemple précédent peut s'écrire différemment grâce à la contrainte VERIFIANT. En effet, le champ débit du type vanne est une constante (si c'était un attribut, ce serait refusé par le *parser*).

Exemple :

```

TYPE vanne ;
CONSTANTE
    debit DOMAINE REEL ;
ATTRIBUT
    position DOMAINE 'ouvert' 'ferme' PAR_DEFAULT 'ouvert' ;

TYPE pompe ;
INTERFACE
    montant GENRE vanne ;
INTERACTION
SI IL_EXISTE x UN montant
        VERIFIANT debit DE x > 1000
        (* utilisation de la contrainte VERIFIANT *)
        TEL_QUE (position DE x = 'ouvert')
ALORS ...

```

5.3.2. Le quantificateur QSOIT

Syntaxe :

```

QSOIT variable
    UN | UNE      nom_d_ensemble
    [ DE_TYPE      type ]
    [ VERIFIANT    condition ]
    ON_A expression_booléenne

```

Cet opérateur correspond au quantificateur \forall de la logique des prédicats. Il permet de tester si **tous** les objets de l'ensemble vérifient une condition particulière (facette ON_A) qui est représentée par une expression booléenne :

```

QSOIT x UN <ensemble d'objets> ON_A <condition>

```

Le mécanisme de cet opérateur est le suivant :

- Si l'ensemble est non vide, on évalue l'expression booléenne de la condition ON_A pour chaque objet de l'ensemble. Si tous les résultats sont VRAIs (ET logique entre les résultats), le résultat de l'opérateur est VRAI. Sinon (au moins un des résultats est FAUX), le résultat de l'opérateur est FAUX.
- Si l'ensemble est vide, le résultat est VRAI. Attention à ce cas particulier, si on l'oublie, cela peut être la source d'une erreur !

Exemple :

```

TYPE pompe ;
ATTRIBUT
    etat DOMAINE 'arrêt' 'marche' PAR_DEFAULT 'marche' ;

TYPE pompe_secours ;
INTERFACE
    pompe_secourue GENRE pompe;
INTERACTION
SI QSOIT x UNE pompe_secourue
        ON_A etat DE x = 'arrêt'

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

ALORS ...

5.3.3. Le quantificateur IL_EXISTE AU_MOINS

Syntaxe :

```

IL_EXISTE AU_MOINS      expression_entière
                        variable
                        INCLUS_DANS  nom_d_ensemble
                        [ DE_TYPE    type ]
                        [ VERIFIANT   condition ]
                        TEL_QUE | TELLE_QUE expression_booléenne

```

Cet opérateur correspond au quantificateur k sur n . Il permet de tester si **au moins** k (exprimé par l'expression entière) objets de l'ensemble vérifient une condition particulière (facette TEL_QUE ou TELLE_QUE) qui est représentée par une expression booléenne :

IL_EXISTE AU_MOINS k x **INCLUS_DANS** <ensemble d'objets> **TEL_QUE** <condition>

Le mécanisme de cet opérateur est le suivant :

- Si l'ensemble est non vide (c'est-à-dire $n \neq 0$), on évalue l'expression entière représentant la valeur k .

- * Si $k = 0$, le résultat est VRAI.

- * Si $k > n$, le résultat est FAUX.

- * Si $0 < k \leq n$, on évalue l'expression booléenne de la condition TEL_QUE pour chaque objet de l'ensemble. Si au moins k expressions sont évaluées à VRAI, le résultat est VRAI. Sinon (au moins $n - k$ expressions sont évaluées à FAUX) le résultat est FAUX.

- Si l'ensemble est vide (c'est-à-dire $n = 0$), le résultat est FAUX.

Exemple :

```

TYPE pompe ;
ATTRIBUT
    etat DOMAINE 'arret' 'marche' PAR_DEFAULT 'marche' ;

TYPE pompe_secours ;
ATTRIBUT
    K      DOMAINE ENTIER
          PAR_DEFAULT 2 ;
INTERFACE
    pompe_secourue GENRE pompe ;
INTERACTION
    (* La pompe de secours ne se déclenche que si AU_MOINS K pompes
       sont à l'arret *)
    SI IL_EXISTE AU_MOINS K x      INCLUS_DANS pompe_secourue
                                   TELLE_QUE etat DE x = 'arret'
ALORS ...

```

5.3.4. Combinaison de quantificateurs

Lors de l'utilisation d'opérateurs complexes imbriqués, il n'est pas évident de se rendre compte du résultat que Figaro produira. Aussi, il est possible de considérer que tout quantificateur correspond à une succession de conditions liées par un opérateur logique. En effet, les résultats des évaluations des conditions des quantificateurs sont liés :

- par un OU pour le quantificateur IL_EXISTE
- par un ET pour le quantificateur QQSOIT.

Lors d'imbrications de quantificateurs, il peut être intéressant de substituer les quantificateurs par leur combinaison d'expressions sur un exemple d'objets pour se faire une idée du résultat (cela revient à faire mentalement une instanciation en Figaro d'ordre 0).

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Par exemple, si on considère les ensembles d'objets suivants :

```
X = { x1 x2 }
Y = { y1 y2 y3 }
```

et la règle suivante :

```
SI  $\exists x$  UN X / ( $\forall y$  UN Y ON_A x <> y) ALORS ...
```

cette règle équivaut à :

```
SI ( $\forall y$  UN Y ON_A x1 <> y) OU ( $\forall y$  UN Y ON_A x2 <> y) ALORS ...
```

qui est équivalent à

```
SI (x1 <> y1 ET x1 <> y2 ET x1 <> y3) OU
(x2 <> y1 ET x2 <> y2 ET x2 <> y3) ALORS ...
```

Grâce à cette substitution, il est facile de se faire une idée du résultat.

Il faut particulièrement faire attention à la portée des variables liées lors des imbrications d'opérateurs complexes. En effet, la portée d'une variable liée est réduite à l'intérieur du quantificateur qui définit l'ensemble auquel appartient cette variable. L'exemple suivant provoque une erreur car la variable x (définie dans l'opérateur IL_EXISTE) n'est pas définie dans le quantificateur QQSOIT :

Exemple :

```
SI IL_EXISTE x UN X (* fin de l'opérateur IL_EXISTE *)
ET QQSOIT y UN Y ON_A x <> y (* ERREUR : x n'est pas défini ! *)
```

Il faut écrire :

```
SI IL_EXISTE x UN X TEL_QUE
QQSOIT y UN Y ON_A x <> y (* x est défini dans IL_EXISTE *)
```

5.3.5. L'itérateur POUR_TOUT

Cet itérateur permet de spécifier une série d'actions dans les facettes PROVOQUE d'une règle d'occurrence, ALORS et SINON d'une règle d'interaction. Il sera présenté dans le paragraphe consacré aux actions des règles Figaro (cf. § 6.3.2.2).

Exemple :

```
SI etat DE reseau = 'marche'
ALORS POUR_TOUT x UN tableau_electrique
FAIRE etat DE x <-- 'sous_tension' ;
```

5.3.6. L'itérateur SOIT

Cet itérateur permet de créer (lorsqu'un modèle est instancié à l'ordre 0) un ensemble de règles portant sur tous les objets d'un ensemble, ou tous les nuplets d'objets d'un produit cartésien d'ensembles. Il sera présenté dans le paragraphe consacré aux règles d'interaction Figaro (cf. § 6.3.4).

Exemple :

```
SOIT x UNE pompe_secours
SI etat(x) = 'arret'
ALORS etat(x) <-- 'sollicite' ;
```

5.3.7. Base de connaissances complète avec quantificateurs

La mini base de connaissances suivante, de 45 lignes seulement, définit de manière complète la sémantique des réseaux de Petri stochastiques généralisés. Elle présente l'avantage d'illustrer l'utilisation des quantificateurs, de l'accès à une variable par une chaîne d'interfaces et de la fonction CARDINAL. On voit aussi comment on peut, comme c'est le cas dans les réseaux de Petri stochastiques généralisés, avoir une transition dont le paramètre de

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

loi est un attribut, susceptible de changer lorsque l'état du système change. La sémantique de ce type de transition est décrite au §3.4. Enfin, cette base de connaissances est représentative des bases dites « abstraites » parce que les classes y correspondent, non directement à des composants physiques comme dans tous les exemples pris jusqu'ici, mais à des éléments d'un modèle graphique.

```
(* Définition des réseaux de Petri stochastiques généralisés en Figaro *)
TYPE place ;
  ATTRIBUT marquage DOMAINE ENTIER PAR_DEFAULT 0 ;

TYPE arc ;
  CONSTANTE poids DOMAINE ENTIER PAR_DEFAULT 1 ;

TYPE arc_ament SORTE_DE arc ;
  INTERFACE depart GENRE place          CARDINAL 1 ;
               arrivee GENRE transition  CARDINAL 1 ;

TYPE arc_aval SORTE_DE arc ;
  INTERFACE depart GENRE transition  CARDINAL 1 ;
               arrivee GENRE place    CARDINAL 1 ;

TYPE arc_inhibiteur SORTE_DE arc_ament ;

TYPE transition ;
  INTERFACE amount GENRE arc_ament ;
               aval GENRE arc_aval ;
               inhibe GENRE arc_inhibiteur ;

TYPE transition_exponentielle SORTE_DE transition ;
  INTERFACE pl GENRE place CARDINAL 0 JUSQUA 1;

PARAMETRE_LOI lambda PAR_DEFAULT 1.e-3;
ATTRIBUT lambda_calcule DOMAINE REEL PAR_DEFAULT 0.0 ;

OCCURRENCE
SI ( QQSOIT x UN amount ON_A marquage ( depart DE x ) >= poids DE x )
  ET
  ( QQSOIT y UN inhibe ON_A marquage ( depart DE y ) < poids DE y )
IL PEUT_SE PRODUIRE TRANSITION tir LOI EXP ( lambda_calcule )
PROVOQUE
  ( POUR_TOUT x UN aval FAIRE marquage ( arrivee DE x )
    <-- marquage ( arrivee DE x ) + poids DE x ) ,
  ( POUR_TOUT x UN amount FAIRE marquage ( depart DE x )
    <-- marquage ( depart DE x ) - poids DE x ) ;

INTERACTION
(* Si une place est déclarée en interface pl, le
   lambda_calcule est égal à son marquage * lambda.
   sinon, il est fixe, égal à lambda *)
SI CARDINAL(pl) = 1
ALORS POUR_TOUT x UN pl FAIRE lambda_calcule <-- marquage(x) * lambda
SINON lambda_calcule <-- lambda;
```

5.4. Les opérateurs booléens

Ils sont au nombre de quatre : un opérateur de gestion d'ensemble (INCLUS_DANS), un opérateur de gestion d'expression (AU_MOINS ... PARMI) et deux opérateurs de test de panne (MARCHE et PANNE).

5.4.1. L'opérateur INCLUS_DANS

Syntaxe :

```
variable INCLUS_DANS nom_d_ensemble
```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Cet opérateur permet de tester l'appartenance d'un objet à un ensemble. Pour cela, on précise à gauche de l'opérateur la variable objet et, à droite, un ensemble d'objets représenté par une variable d'interface ou un nom de type. Dans le cas d'un nom de type, on teste simplement si l'objet est du type précisé ou bien si l'objet est d'un type qui hérite du type spécifié.

Si l'objet apparaît dans l'ensemble des objets, le résultat sera VRAI. Sinon il sera FAUX. Si l'ensemble est vide, le résultat est FAUX.

Dans l'exemple suivant, on teste si toutes les pompes du système sont incluses dans l'interface de l'objet.

Exemple :

```

TYPE composant ;
INTERFACE
    montant GENRE pompe ;
INTERACTION
SI QQS0IT x UN_OBJET_DE_TYPE pompe
                ON_A x INCLUS_DANS montant
ALORS ...

```

Remarque : Il n'est pas possible d'imposer des contraintes d'ensemble sur l'opérateur INCLUS_DANS.

5.4.2. L'opérateur MARCHE

Syntaxe :

```

MARCHE [ DE objet ]
MARCHE [ ( objet ) ]

```

L'opérateur MARCHE teste si un objet est en marche, c'est-à-dire si toutes les variables PANNE de l'objet sont à FAUX (cf. PANNE (en tant que variable d'état), § 6.2.4.4). L'opérateur MARCHE permet d'accéder à un objet comme suit :

- Si aucun objet n'est précisé, l'opérateur considère qu'il s'agit de l'objet dans lequel l'opérateur est utilisé.
- L'objet sur lequel doit agir l'opérateur peut être identifié soit par une variable liée, soit par un nom d'ensemble de cardinal 1, soit par un nom d'objet particulier.

L'opérateur effectue les opérations suivantes :

- Il récupère tous les champs PANNE de l'objet.
- Il teste si tous ces champs (booléens) sont à FAUX. Si c'est le cas, le résultat de l'opération est VRAI. Sinon (au moins un des champs PANNE est VRAI), le résultat est FAUX.
- Si l'objet n'a aucun champ PANNE, le résultat est VRAI.

L'exemple suivant teste si une pompe est alimentée par au moins une source et ce, uniquement si la pompe fonctionne.

Exemple :

```

TYPE pompe;
INTERFACE
    montant GENRE source ;
PANNE
    panne_pompe ;
EFFET alim ;
INTERACTION
    SI MARCHE ET
        IL_EXISTE x UN montant TEL_QUE alim DE x
    ALORS
        alim ;

```


EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

5.4.3. L'opérateur PANNE

Syntaxe :

```
PANNE [ DE objet ]
PANNE [ ( objet ) ]
```

L'opérateur PANNE est le complémentaire de l'opérateur MARCHE. Il teste si au moins un des attributs PANNE (cf. PANNE (en tant que variable d'état), § 6.2.4.4) de l'objet est à VRAI. Dans ce cas, il renvoie VRAI. L'opérateur PANNE permet d'accéder à un objet comme suit :

- Si aucun objet n'est précisé, l'opérateur considère qu'il s'agit de l'objet dans lequel l'opérateur est utilisé.
- L'objet sur lequel doit agir l'opérateur peut être identifié soit par une variable liée, soit par un nom d'ensemble de cardinal 1, soit par un nom d'objet particulier.

L'opérateur effectue les opérations suivantes :

- Il récupère le nom de tous les champs PANNE de l'objet
- Il teste si au moins un des champs (booléens) est à VRAI. Si c'est le cas, le résultat de l'opération est VRAI. Sinon (tous les champs PANNE sont FAUX), le résultat est FAUX.
- Si l'objet n'a aucun champ PANNE, le résultat est FAUX.

L'exemple suivant concerne une pompe de secours qui ne démarre que si la pompe secourue est en panne.

Exemple :

```
TYPE pompe_secours;
INTERFACE
  pompe_secourue GENRE pompe CARDINAL 1 ;
ATTRIBUT
  etat DOMAINE 'arret' 'marche' 'demande_demarrage'
  PAR_DEFAULT 'arret' ;
INTERACTION
  (* On essaie de démarrer si la pompe secourue est en panne *)
  SI MARCHE ET etat = 'arret'
    ET PANNE( pompe_secourue )
  ALORS
    etat <-- 'demande_demarrage' ;
```

5.4.4. L'opérateur AU_MOINS ... PARMI

Syntaxe :

```
AU_MOINS expression_entière PARMI ( liste_expressions_booléennes )
```

Cet opérateur permet de tester si au moins k expressions de la liste sont vérifiées. Pour cela, on précise la valeur de k par une expression entière et la liste des expressions booléennes à tester (les expressions sont séparées par des virgules). Si au moins k expressions sont évaluées à VRAI, le résultat sera VRAI. Sinon il sera FAUX. Si la valeur de k est égale à 0, le résultat est VRAI. Si la valeur de k est supérieure au nombre d'expressions, le résultat est FAUX.

Dans l'exemple suivant, on teste si au moins 2 des composants en interface sont en panne.

Exemple :

```
TYPE composant ;
INTERFACE
  composant1 GENRE composant CARDINAL 1;
  composant2 GENRE composant CARDINAL 1;
  composant3 GENRE composant CARDINAL 1;
INTERACTION
  SI AU_MOINS 2 PARMI ( PANNE(composant1),
                        PANNE(composant2),
                        PANNE(composant3) )
```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

ALORS ...

5.5. Opérateurs numériques complexes

Figaro dispose aussi d'opérateurs numériques dits complexes parce qu'ils agissent sur des ensembles, c'est-à-dire des champs d'interface ou des noms de type. Ces opérateurs sont principalement utilisés pour écrire des équations en Figaro (cf. Chapitre Equations).

Remarque : Comme pour les opérateurs booléens complexes, il est possible de préciser des contraintes sur l'ensemble d'objets traités par l'opérateur (DE_TYPE et VERIFIANT) pour ne traiter qu'un sous-ensemble d'objets.

Les opérateurs numériques complexes sont au nombre de cinq : CARDINAL, SOMME, PRODUIT, MINIMUM, MAXIMUM.

5.5.1. L'opérateur CARDINAL

Syntaxe :

```
CARDINAL  DE champ_d_interface
CARDINAL  ( champ_d_interface )
```

L'opérateur CARDINAL retourne le nombre d'éléments d'un ensemble. Appliqué à une interface, il renvoie le nombre d'objets que contient le champ interface.

- Si l'interface est monovaluée, le résultat est 1.
- Si l'interface ne contient pas d'objets, le résultat est 0.

Il est possible de récupérer le CARDINAL d'une interface d'un objet voisin grâce à l'opérateur DE ou ().

Dans l'exemple suivant, on modélise le comportement d'une porte logique ET. On vérifie que la porte logique ET est reliée à au moins deux composants en *amont*. On vérifie aussi que tous les objets de l'interface amont de type *porte_logique*, ont une interface *aval* à un seul élément.

Exemple :

```
TYPE composant ;

TYPE porte_logique SORTE_DE composant ;
INTERFACE
    aval GENRE composant;

TYPE porte_ET SORTE_DE porte_logique;
INTERFACE
    amont GENRE composant ;
ATTRIBUT
    etat DOMAINE 'valide' 'pas valide' PAR_DEFAULT 'valide';
INTERACTION
    SI ( CARDINAL DE amont < 2 )
    OU ( IL_EXISTE x UN amont DE_TYPE porte_logique
        TEL_QUE CARDINAL DE aval DE x <> 1 )
    ALORS etat <-- 'pas valide' ;
```

Remarque : Cet opérateur est à différencier de la facette CARDINAL accessible pour la description de l'interface d'un objet.

5.5.2. L'opérateur SOMME

Syntaxe :

```
SOMME POUR_TOUT variable UN | UNE nom_d_ensemble
                                [ DE_TYPE type ]
                                [ VERIFIANT condition ]
DES_TERMES expression_numérique
```

L'opérateur SOMME permet d'effectuer une somme d'expressions sur un ensemble d'objets.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Il utilise une variable liée définie sur un ensemble d'objets (facette POUR_TOUT) et il additionne successivement, pour chaque objet de l'ensemble, la valeur de l'expression numérique (facette DES_TERMES) pour cet objet :

SOMME POUR_TOUT x **UN** <ensemble d'objets> **DES_TERMES** <expression numérique>

Exemple :

```

TYPE macro_composant ;
ATTRIBUT
    taux_defaillance DOMAINE REEL PAR_DEFAULT 0 ;
INTERFACE
    element GENRE composant ;
INTERACTION
    ALORS taux_defaillance <-- SOMME POUR_TOUT x UN element DES_TERMES
        (lambda DE x* MARCHE DE x) ;

```

Le mécanisme de l'opérateur est le suivant :

- Si l'ensemble est non vide, on évalue l'expression numérique DES_TERMES pour chaque objet de l'ensemble. Le résultat de l'opérateur est la somme des résultats des expressions.
- Si l'ensemble est vide, le résultat est 0. Attention à ne pas oublier ce cas particulier !

La facette DES_TERMES permet de préciser quels sont les calculs que nous voulons effectuer et additionner. L'expression doit être numérique pour pouvoir être additionnée.

5.5.3. L'opérateur PRODUIT

Syntaxe :

```

PRODUIT POUR_TOUT variable UN | UNE nom_d_ensemble
    [ DE_TYPE type ]
    [ VERIFIANT condition ]
DES_TERMES expression_numérique

```

L'opérateur PRODUIT permet d'effectuer un produit d'expressions sur un ensemble d'objets.

Il utilise une variable liée définie sur un ensemble d'objets (facette POUR_TOUT) et il calcule le produit, pour tous les objets de l'ensemble, des valeurs de l'expression numérique donnée dans la facette DES_TERMES :

PRODUIT POUR_TOUT x **UN** <ensemble d'objets> **DES_TERMES** <expression numérique>

L'exemple suivant montre une utilisation de cet opérateur dans l'initialisation par défaut d'un attribut.

Exemple :

```

TYPE calcul ;
INTERFACE
    montant GENRE composant ;
ATTRIBUT
    valeur DOMAINE ENTIER
        PAR_DEFAULT PRODUIT POUR_TOUT x UN montant
            DES_TERMES ( a DE x ) + 5 ;

```

Le mécanisme de l'opérateur est le suivant :

- Si l'ensemble est non vide, on évalue l'expression numérique DES_TERMES pour chaque objet de l'ensemble. Le résultat de l'opérateur est le produit des résultats des expressions.
- Si l'ensemble est vide, le résultat est 1. Attention à ne pas oublier ce cas particulier !

La facette DES_TERMES permet de préciser quels sont les calculs que nous voulons effectuer et multiplier. L'expression doit être numérique pour pouvoir être multipliée. Elle peut aussi être booléenne car l'opérateur utilisé pour lier les résultats est l'opérateur de multiplication qui peut effectuer une conversion des booléens en entier.

5.5.4. Les opérateurs MAXIMUM et MINIMUM

Syntaxe :

MAXIMUM | **MINIMUM**

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

POUR_TOUT variable UN | UNE nom_d_ensemble
                [ DE_TYPE type ]
                [ VERIFIANT condition ]
DES_TERMES expression_numérique

```

Ces opérateurs permettent de calculer le maximum (respectivement : minimum) d'expressions sur un ensemble d'objets.

Ils utilisent une variable liée définie sur un ensemble d'objets (facette POUR_TOUT) et ils calculent le maximum (respectivement : minimum), pour tous les objets de l'ensemble, des valeurs de l'expression numérique donnée dans la facette DES_TERMES :

```

MAXIMUM POUR_TOUT x UN <ensemble d'objets> DES_TERMES <expression numérique>

```

L'exemple suivant montre une utilisation de cet opérateur dans l'initialisation par défaut d'une constante.

Exemple :

```

TYPE calcul ;
INTERFACE
    montant GENRE composant ;
CONSTANTE
    plus_gros_debit DOMAINE ENTIER
        PAR_DEFAUT MAXIMUM POUR_TOUT x UN montant
                                DES_TERMES debit_nominal DE x ;

```

Le mécanisme de l'opérateur est le suivant :

- Si l'ensemble est non vide, on évalue l'expression numérique DES_TERMES pour chaque objet de l'ensemble. Le résultat de l'opérateur est le maximum (respectivement : minimum) des résultats des expressions.
- Si l'ensemble est vide, le résultat est 0. Attention à ne pas oublier ce cas particulier !

La facette DES_TERMES permet de préciser quels sont les calculs que nous voulons effectuer et comparer. L'expression doit être numérique pour pouvoir être comparée. Elle peut aussi être booléenne car l'opérateur maximum (respectivement : minimum) peut effectuer une conversion des booléens en entier ou réel.

5.6. Opérateurs spécifiques

Figaro contient des opérateurs spécifiques tels que les fonctions mathématiques (exemples : exp, sin, cos, tan, max, ...) ou les générateurs de nombres aléatoires suivant différentes lois.

La liste des opérateurs spécifiques s'allonge au gré des besoins. Cette liste est actualisée dans la note définissant la syntaxe Figaro.

L'utilisation de tels opérateurs suit les règles suivantes :

```

opérateur ( expression )
ou
opérateur ( liste_expressions )

```

Remarque : Dans une liste d'expressions, les expressions sont séparées par des virgules.

Exemples :

```

EXP ( 2 + ( a * 3 ) )
MAX ( a, b, c ) (* attention ici le mot-clé est MAX et non MAXIMUM *)
RAND_UNI (a, b) (* chaque appel à cette fonction produit un nombre aléatoire
équiréparti entre les réels a et b *)

```

5.7. L'opérateur ?=

L'opérateur "?=" est très puissant, car il permet de faire la même chose qu'un opérateur de type "case" ou "switch" dans un langage de programmation, ou bien des instructions if then else imbriquées.

Syntaxe :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

`N ?= (Valeur si N=1, valeur si N=2..., valeur sinon)`
 (* N est une expression entière *)

Le résultat de cette expression a le même type que celui des valeurs entre parenthèses (qui doivent avoir toutes le même type).

Exemples :

```
taille ?= ('petit', 'moyen', 'grand', 'inexistant') (* taille va de 1 à 3 *)

(taille = 1) ?= ('petit', 'tout_sauf_petit') (* si l'opérande de gauche est
booléen, il est considéré comme l'entier 0 ou 1 *)

R <-- A ?= (2, B ?= (3, 4))
```

Le résultat R de cette dernière expression est calculé comme suit :

Si A = 1, R vaut 2 (quelle que soit la valeur de B)

Si A est différent de 1, R vaut 3 si B = 1 et 4 si B est différent de 1.

On a donc un équivalent de deux expressions if then else imbriquées.

5.8. Portée d'une variable

Comme dans tout langage de programmation, les variables en Figaro ont une portée.

Suivant sa nature (variable d'OBJET_SYSTÈME, champ d'objet, variable liée), chaque variable a une portée différente et une priorité d'évaluation différente (c'est-à-dire l'ordre dans lequel la valeur de la variable est recherchée dans chacun des contextes : global, d'objet ou local à un opérateur complexe).

- La portée d'une variable d'OBJET_SYSTÈME est **globale** : tous les types de la base de connaissances pourront y accéder.
- Un champ de type est associé à un type. Sa portée est **locale au type**, c'est-à-dire qu'elle est inconnue et inaccessible par les autres types autrement que via une interface ou une chaîne d'interfaces du genre `a_gauche` (`en_haut` (`a_droite`)).
- Enfin, une variable liée est une variable définie dans un opérateur complexe comme QQSOIT, IL_EXISTE, IL_EXISTE_AU_MOINS, SOMME, PRODUIT, MINIMUM, MAXIMUM, POUR_TOUT et SOIT. En effet, ces opérateurs permettent de définir une variable qui prend successivement certaines valeurs afin d'effectuer certains calculs. Cette variable est liée à l'opérateur, c'est-à-dire que sa portée est **interne à l'opérateur**. En dehors de celui-ci, la variable n'est plus connue.

Maintenant que la notion de portée est plus précise, il nous faut définir l'ordre dans lequel Figaro recherche la valeur d'une variable. En effet, il est possible de surcharger le nom d'une variable. C'est à dire que l'on peut nommer une variable liée comme le champ d'un type (bien que ce ne soit pas recommandé !). Dans ces cas, la priorité est donnée à la variable qui a la portée la plus réduite. Ainsi, Figaro recherchera d'abord (grâce au nom de la variable) la valeur de la variable dans le contexte des variables liées. Ensuite, s'il échoue, il passera au contexte des champs du type (contexte local).

Exemple:

```
TYPE vanne
ATTRIBUT
  x DOMAINE BOOLEEN PAR_DEFAULT VRAI;
INTERACTION
  SI QQSOIT x UN composant
    ON_A propriété(x)
  ALORS x <-- VRAI ;
```

La variable `x` de la condition de la règle est liée à l'instruction QQSOIT. Dans l'instruction propriété de `x`, le `x` trouvé est celui du QQSOIT. Dans le ALORS, il s'agit du champ de l'objet.

Remarque : Il est conseillé de ne pas déclarer de variables qui ont le même nom qu'un type pour faciliter la lecture de la base de connaissances.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

6. Description d'un type Figaro

Comme nous l'avons vu au chapitre 2, une base de connaissances est un ensemble de définitions de classes. Dans ce chapitre, nous présentons de façon détaillée comment décrire une classe en Figaro.

6.1. Structure générale d'une définition de classe

6.1.1. Déclaration d'une classe

Syntaxe :

```
TYPE nom_type[SORTE_DE énumération_de_types];
```

La description d'une classe commence par le mot-clé **TYPE** suivi du nom (de l'identifiant) de la classe. Etant donné le mot clé utilisé, nous utiliserons aussi TYPE comme synonyme de classe.

```
TYPE moto_pompe ;
```

L'héritage est spécifié par une facette **SORTE_DE** qui peut être multivaluée (héritage multiple), et dans laquelle on énumère les surclasses de la classe décrite.

```
TYPE moto_pompe
  SORTE_DE composant pompe ;
```

Remarque : Contrairement aux types, un objet ne peut hériter que d'un seul type.

6.1.2. Structure générale de description d'une classe

Les caractéristiques associées à un TYPE se décomposent en deux grands groupes :

- Les champs valués représentent les caractéristiques **statiques** d'un TYPE. Ce sont les interfaces, les constantes et les variables d'état (attributs, effets, pannes).
- Les règles de fonctionnement (règles d'occurrence et d'interaction) permettent de décrire le comportement dynamique d'un TYPE, c'est-à-dire :
 - Les événements pouvant survenir pour un objet d'un type (changement d'état, défaillance, réparation, ...)
 - Les conditions d'occurrence de ces événements.
 - Les conséquences de ces événements sur le système.

Le schéma suivant présente l'organisation des caractéristiques associées à un TYPE.

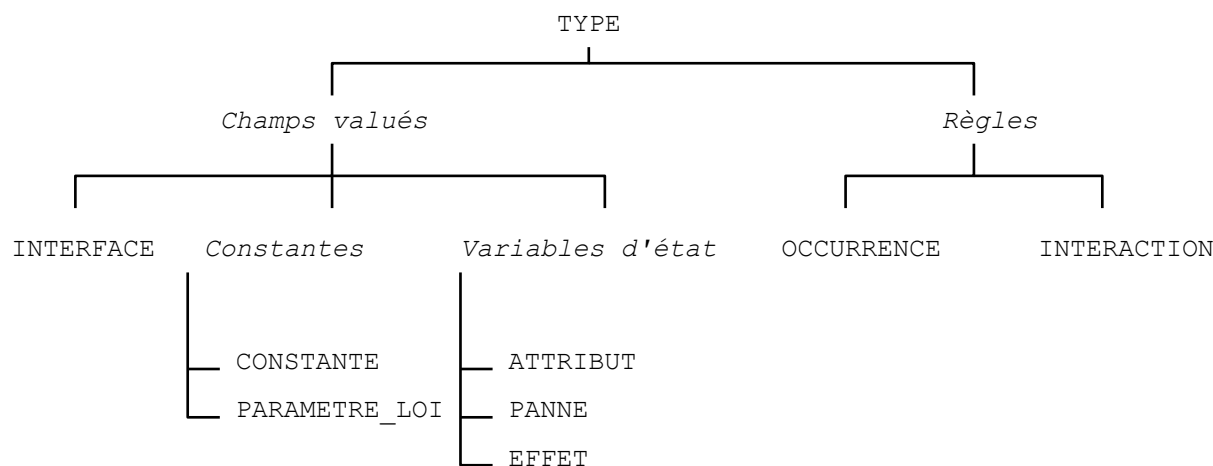


Figure 4. Caractéristiques d'un TYPE

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

6.2. Les champs valués

On distingue trois catégories de champs valués pour un type :

- * Les **champs constants** représentent les caractéristiques des objets qui n'évoluent pas au cours de la vie du système (longueur d'un câble de connection par exemple). Ces champs sont univalués.
- * Les **champs d'interface** sont des champs contenant des noms d'objets. La valeur de ces champs n'évolue pas au cours de la vie du système (les relations entre les objets d'un système sont définies lors de la description du système et ne peuvent évoluer). Ces champs peuvent être multivalués.
- * Les **variables d'état** caractérisent l'état des objets et sont susceptibles d'évoluer au cours de la vie du système. Ces champs sont univalués.

En Figaro, tout champ appartient à un type de champ identifié par un mot-clé (CONSTANTE et PARAMETRE_LOI pour les champs constants, INTERFACE pour les champs d'interface, ATTRIBUT, PANNE et EFFET pour les variables d'état). Chaque champ est décrit par son nom puis par des facettes de description. On peut résumer la structure générale de la description d'un champ de type comme suit :

Syntaxe :

```
<type_de_champ>
  champ1 <nom_de_facette>    <liste_de_valeurs>
    ...
    <nom_de_facette>    <liste_de_valeurs> ;

  champ2 <nom_de_facette>    <liste_de_valeurs>
    ...
    <nom_de_facette>    <liste_de_valeurs> ;

  ...
<type_de_champ>    ...
```

Exemples de déclarations de champ :

```
ATTRIBUT
  fonc DOMAINE 'arret' 'demande_demarrage'
        'refus_demarrage' 'marche'
  PAR_DEFAULT 'arret' ;

CONSTANTE
  poids DOMAINE ENTIER ;
```

Avant de décrire plus en détail chaque type de champ possible, il nous faut préciser certaines notions communes à toutes les catégories de champs.

6.2.1. Préliminaires

6.2.1.1. Etat initial d'un modèle Figaro

Lors de l'écriture d'une base de connaissances, l'utilisateur décrit des types génériques et réutilisables pour différentes études. Pour étudier un système particulier, ces types sont utilisés pour décrire le système. La description complète d'un système consiste à créer les objets du système et à affecter des valeurs à chaque champ caractéristique du type de l'objet. Ces valeurs sont :

- soit une valeur initiale pour les variables d'état du modèle,
- soit une valeur pour les constantes du modèle.

Exemple :

```
TYPE vanne ;
CONSTANTE
  position DOMAINE 'ouvert' 'ferme';

(* base de faits *)
```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```
OBJET v1 EST_UNE vanne ;
    position = 'ouvert' ;
```

L'ensemble des valeurs attribuées aux champs des objets lors de la description du système constitue l'état initial incomplet du modèle. L'état initial complet du modèle sera calculé en appliquant les règles d'interaction sur cet état initial incomplet.

Remarque : l'attribution d'une valeur initiale à chaque variable d'état du modèle n'est pas obligatoire. Avant d'appliquer les règles d'interaction, le traitement de calcul de l'état initial d'un modèle attribue la valeur par défaut déclarée dans la base de connaissances à chaque variable non initialisée.

6.2.1.2. Valeur par défaut

Il est parfois intéressant d'indiquer dans la base de connaissances une **valeur par défaut** pour les champs génériques caractérisant les types. Cela permet d'alléger la phase de description d'un système. En effet, les valeurs par défaut attribuées aux champs d'un type seront automatiquement attribuées aux champs de chaque objet créé pour ce type. Ces valeurs pourront bien sûr être modifiées par l'utilisateur.

Une valeur par défaut représente :

- soit une valeur initiale par défaut pour les champs "variables d'état",
- soit une valeur par défaut pour les champs "constantes".

Pour un champ donné d'un type, une valeur par défaut est créée en utilisant la facette PAR_DEFAULT. **L'attribution d'une valeur par défaut à un champ de type n'est jamais obligatoire** (les règles d'interaction peuvent être prévues pour calculer la valeur de la variable dans toutes les situations).

L'exemple suivant met en évidence cette notion en définissant un type *vanne* avec un attribut *position* qui aura pour valeur par défaut 'ouvert'. Deux objets de ce type seront alors créés : un utilisant la valeur par défaut, l'autre la redéfinissant.

Exemple :

```
TYPE vanne ;
ATTRIBUT
    position DOMAINE 'ouvert' 'ferme' PAR_DEFAULT 'ouvert' ;

(* base de faits *)
OBJET v1 EST_UNE vanne ;
(* la valeur initiale de position de v1 est 'ouvert' *)

OBJET v2 EST_UNE vanne ;
ATTRIBUT
    position = 'ferme' ;
(* la valeur initiale de position de v2 est 'ferme' *)
```

Remarque : Pour un champ donné d'un objet, une valeur initiale est créée en utilisant la facette "=". Cette facette n'est pas autorisée pour les caractéristiques des types. Réciproquement la facette PAR_DEFAULT est interdite pour les caractéristiques des objets.

La valeur par défaut utilisée avec le mécanisme d'héritage de Figaro permet de définir plus précisément un élément d'un type au niveau de ses sous-types. Dans notre cas, on pourra par exemple définir dans un type le domaine d'une variable d'état, et donner dans les sous-types la valeur par défaut de cette variable.

Dans l'exemple suivant, on divise les composants électriques en deux familles : les sources de courant (lignes, diesels, ...) et les éléments passifs (traversés par un courant comme les disjoncteurs et les tableaux). Par défaut, on dit qu'une source est alimentée, et qu'un élément passif ne l'est pas.

Exemple :

```
TYPE composant_electrique ;
ATTRIBUT
    alimente DOMAINE 'oui' 'non' ;

TYPE source_de_courant SORTE_DE composant_electrique;
ATTRIBUT
    alimente PAR_DEFAULT 'oui' ;
```


EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

TYPE element_passif SORTE_DE composant_electrique ;
ATTRIBUT
    alimente PAR_DEFAULT 'non' ;

```

6.2.1.3. Notion d'édition

En principe, lorsqu'un utilisateur décrit un système en utilisant une base de connaissances, chaque fois qu'un objet est créé, il a la possibilité d'éditer l'ensemble des champs caractéristiques du type de l'objet ainsi que leur valeur par défaut qu'il peut modifier.

Cependant, le langage Figaro permet au concepteur de base de connaissances :

- de restreindre l'ensemble des champs visualisables et éditables par l'utilisateur chargé de décrire un système (lors de l'écriture d'une base de connaissances, il peut être utile de définir des variables utiles à la spécification du modèle, mais inutiles à sa compréhension),
- de contrôler l'édition des champs de la base (rendre obligatoire l'initialisation de certains champs, rendre non modifiables les valeurs par défaut de certains champs, etc.).

La facette EDITION (facultative) permet de préciser les caractéristiques d'édition d'un champ d'un type dans la base de connaissances. Les options d'éditions sont les suivantes. Elles peuvent être regroupées dans la même facette d'EDITION si elles sont liées par des virgules (liste de valeurs).

VISIBLE	NON VISIBLE
MODIFIABLE	NON MODIFIABLE
OBLIGATOIRE	NON OBLIGATOIRE

*Remarque : La négation d'une option est réalisée par l'ajout du mot-clé NON à cette option. Exemple : Négation de l'option VISIBLE : **NON VISIBLE**.*

Voici le détail des options (les options en **gras** sont les options par défaut d'un champ quelconque) :

Option **VISIBLE**

Le champ sera visualisé par l'utilisateur lors de la création d'un objet du type.

Option **NON VISIBLE**

Le champ ne sera pas visualisé par l'utilisateur lors de la création d'un objet du type.

Option **MODIFIABLE**

La valeur par défaut attribuée au champ dans la base de connaissances pourra être modifiée par l'utilisateur lors de la description d'un système particulier. Pour les interfaces, cf. la précision importante donnée ci-dessous, pour l'option NON MODIFIABLE.

Option **NON MODIFIABLE**

La valeur par défaut attribuée au champ dans la base de connaissances ne pourra pas être modifiée par l'utilisateur lors de la description d'un système particulier.

Cette option a une signification particulière en ce qui concerne les **interfaces**. Pour ces dernières, il est impossible de donner des valeurs dans une base de connaissances. Lorsqu'une interface est associée à l'option NON MODIFIABLE cela signifie que l'utilisateur n'a pas d'autre moyen pour modifier cette interface que l'utilisation de tracés de liens dans l'outil KB3. **Pour la robustesse des applications d'une base de connaissances, il est donc important de préciser cette option pour les interfaces remplies automatiquement par le tracé de liens.**

Option **OBLIGATOIRE**

L'utilisateur doit obligatoirement renseigner une valeur initiale pour le champ (cette option n'est pas pertinente pour les champs constants qui doivent être obligatoirement renseignés).

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Option **NON OBLIGATOIRE**

La valeur initiale de la variable n'est pas obligatoire (elle pourra par exemple être calculée par application des règles d'interaction pour le calcul de l'état initial complet).

Remarque : Les options s'appliquent dans l'ordre de priorité suivant : 1. Visible (non visible), 2. Modifiable (non modifiable), 3. Obligatoire (non obligatoire). Ainsi, un champ déclaré non visible ne pourra être modifié car rendu non accessible à l'utilisateur (non visible).

Exemples :

ATTRIBUT

```
champ1 DOMAINE ENTIER
      PAR_DEFAULT 12
      EDITION NON VISIBLE ;
(* ce champ ne pourra être modifié lors de la création d'un objet du type *)

champ2 DOMAINE REEL
      EDITION VISIBLE, MODIFIABLE, OBLIGATOIRE ;
(* ce champ devra être affecté lors de la création d'un objet du type *)
```

Remarque : Seuls les types peuvent définir des champs avec une facette EDITION. Cette facette n'est en aucun cas reconnue dans les bases de faits, elle ne doit donc pas être utilisée dans les objets.

6.2.1.4. Remarque sur le calcul de l'état initial d'un modèle Figaro

Avant toute exploitation d'un modèle Figaro pour générer un modèle de fiabilité (Arbre de défaillances, Graphe d'états, ...), le traitement devra calculer l'état initial complet du modèle.

Le concepteur de base de connaissances devra donc définir, pour chaque variable d'état essentielle (non réinitialisée à chaque passage des règles d'interaction) de la base de connaissances, la méthode d'initialisation retenue :

- attribution d'une valeur initiale par défaut dans la base de connaissances (facette PAR_DEFAULT) modifiable ou non (facette EDITION) par l'utilisateur,
- attribution obligatoire (facette EDITION) d'une valeur initiale lors de la description d'un système,
- calcul de la valeur initiale par les règles d'interaction (cette dernière option n'est pas recommandée pour une variable essentielle).

6.2.2. Les interfaces

Syntaxe :

```
INTERFACE nom_interface
  [GENRE nom_type]
  [CARDINAL nombre_entier]
  | [CARDINAL nombre_entier JUSQUA nombre_entier | INFINI ]
```

Comme nous l'avons vu au chapitre 2, les champs interface permettent de représenter les relations entre les objets des classes. Pour un type donné, les champs introduits par le mot clé INTERFACE permettent d'indiquer les types avec lesquels le type décrit aura une relation.

- Un champ interface est identifié par le nom de l'interface (nom de la relation)
- La facette GENRE permet de préciser le nom du type en relation pour cette interface avec le type décrit (elle est bien sûr obligatoire). L'utilisation de la facette GENRE est interdite dans les objets.

```
TYPE tableau_electrique ;

TYPE pompe ;
INTERFACE alim GENRE tableau_electrique ;
```

- La facette CARDINAL permet de spécifier des contraintes sur la cardinalité de la relation définie par le champ interface. Cette facette permet notamment de spécifier si la relation est obligatoire ou non.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

* Facette **CARDINAL** non définie

Si la facette **CARDINAL** n'est pas définie (*cette facette n'est pas obligatoire*), cela signifie qu'un objet quelconque du type décrit peut être interfacé avec 0 ou plusieurs objets du type en interface (sans limitation de nombre). Dans ce cas, la relation n'est pas obligatoire.

* **CARDINAL** nombre_entier

Si la facette **CARDINAL** est égale à un nombre entier n, cela signifie que tout objet du type décrit sera **obligatoirement** interfacé avec n objets du type en interface (le cardinal de l'ensemble des objets en interface est fixé à n). Si n=1, on dit que l'interface est monovaluée.

* **CARDINAL** nombre_entier **JUSQUA** nombre_entier | **INFINI**

Si la facette **CARDINAL** est un intervalle p,q, cela signifie que tout objet du type décrit sera relié à entre p et q objets du type en interface (le cardinal de l'ensemble des objets en interface est inclus au sens large dans l'intervalle défini). Deux cas particuliers peuvent se présenter :

- . Si p=0, cela signifie que la relation n'est pas obligatoire,
- . Si q = INFINI , cela signifie que tout objet du type décrit peut être relié à plusieurs objets du type en interface, sans limitation de nombre).

Exemple :

```

TYPE vanne_electrique;
INTERFACE      amont GENRE composant;
                  alim_220 GENRE tableau_electrique CARDINAL 1 ;
                  voie_de_commande GENRE commande CARDINAL 1 JUSQU'A 2 ;

```

*Remarque : L'utilisation de la facette **CARDINAL** est interdite dans la base de faits, donc dans les objets.*

6.2.3. Les constantes

Les constantes sont des champs dont la valeur ne varie pas au cours de la vie du système. De par leur nature, ces champs ne pourront donc jamais être modifiés au cours du fonctionnement du système.

On distingue deux types de valeurs constantes : les paramètres loi utilisés par les lois de probabilité (type de champ PARAMETRE_LOI), et les constantes proprement dites (type de champ CONSTANCE).

6.2.3.1. CONSTANCE

Syntaxe :

```

CONSTANCE      nom_constante
[ LIBELLE        "description" ]
[ DOMAINE        ENTIER | REEL | BOOLEEN |
                           énumération_de_symboles
[ PAR_DEFAULT    valeur | expression ]
[ EDITION        option_edition ]

```

Les constantes sont les caractéristiques des objets d'une classe qui n'évoluent pas au cours de la vie d'un système. Autrement dit, les événements survenant sur le système n'ont pas de conséquences (ne modifient pas) sur ces caractéristiques. Les constantes sont donc principalement utilisées pour représenter les caractéristiques fixes d'un système (longueur d'un câble, impédance d'une ligne, ...).

Exemple :

```

TYPE cable ;
CONSTANCE
  longueur      DOMAINE REEL
                 LIBELLE "longueur de l'arc"
                 PAR_DEFAULT 25.0

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

impedance **EDITION OBLIGATOIRE ;**
DOMAINE REEL ;

Plusieurs facettes permettent de définir précisément le type et la valeur d'une constante. Voici la description de chacune d'entre elles :

Facette LIBELLE (facultative)

Permet de renseigner entre guillemets une description plus complète de la constante. Elle rend la base de connaissances et la description du système plus lisibles. Cette facette est autorisée pour les types et les objets.

Facette DOMAINE (obligatoire)

Précise **l'ensemble des valeurs possibles** pour la constante.

Il existe trois types prédéfinis de domaines en Figaro :

- **BOOLEEN** : contient uniquement les valeurs VRAI et FAUX.
- **ENTIER** : contient toutes les valeurs entières.
- **REEL** : contient toutes les valeurs réelles.

Il est aussi possible de définir son propre domaine de valeurs sous la forme d'une énumération. Ces valeurs doivent être des valeurs constantes (il n'est pas possible d'utiliser des variables dans l'énumération). Chaque valeur est un *symbole* matérialisé par une chaîne de caractères entre quotes ('chaîne'). La déclaration d'un domaine énuméré permet de préciser quelles sont les valeurs que l'on peut donner au champ sans introduire d'incohérence dans le modèle.

Exemple :

```
TYPE transformateur ;
CONSTANTE
  tension_primaire      DOMAINE '400kv' '220kv'
                       PAR_DEFAULT '400kv'
                       EDITION OBLIGATOIRE ;
```

Facette PAR_DEFAULT (facultative)

Cette facette accepte une valeur précise appartenant au domaine de définition ou une expression constante, c'est-à-dire dans laquelle ne sont utilisés que des champs constants (l'ensemble des valeurs possibles pour l'expression doit être inclus dans le domaine de définition de la constante).

Remarque sur l'affectation d'une valeur par défaut sous forme d'expression :

Lors de l'exploitation d'un modèle Figaro, le traitement de calcul de l'état initial détectera une erreur s'il trouve dans le modèle une constante dont la valeur n'est pas définie (toutes les constantes doivent être résolues au moment du calcul de l'état initial). Ainsi, lorsque dans une base de connaissances, on définit une valeur par défaut sous forme d'expression pour une constante, il faudra veiller à ce que l'utilisation de cette base rende obligatoire la saisie d'une valeur pour les champs constants intervenant dans l'expression (en utilisant l'option OBLIGATOIRE pour la facette EDITION par exemple). En effet, si un champ de l'expression n'a pas de valeur, le traitement ne pourra évaluer l'expression.

Facette EDITION (facultative)

(cf. § 6.2.1.3)

6.2.3.2. PARAMETRE_LOI

Syntaxe :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

PARAMETRE_LOI nom_parametre_loi
[ LIBELLE      "description" ]
[ PAR_DEFAULT valeur | expression ]
[ EDITION     option_edition ]

```

Comme nous l'avons vu précédemment, l'occurrence d'un événement sur un objet est fonction d'une loi de probabilité, qui s'exprime en fonction d'un certain nombre de paramètres.

Exemple : L'occurrence de la défaillance en fonctionnement d'une pompe est régie par une loi de probabilité exponentielle de paramètre *lambda_pompe*.

La catégorie de champs **PARAMETRE_LOI** d'une classe permet de regrouper tous les paramètres qui sont utilisables pour exprimer les lois de probabilité des événements pouvant survenir sur les objets de la classe.

Exemple :

```

TYPE pompe ;
PARAMETRE_LOI
    lambda_pompe ;

```

Les paramètres loi sont des constantes qui seront **toujours des réels**, ce qui nous dispense de déclarer leur domaine de définition.

Les facettes associées à la déclaration d'un champ **PARAMETRE_LOI** sont :

Facette LIBELLE (facultative)

(cf. CONSTANCE)

Facette PAR_DEFAULT (facultative)

Cette facette accepte une valeur réelle ou une expression constante, c'est-à-dire dans laquelle ne sont utilisés que des champs constants (l'ensemble des valeurs possibles pour l'expression doit être inclus dans l'ensemble des réels).

Cf. CONSTANCE pour la remarque sur l'affectation d'une valeur par défaut sous forme d'expression.

Facette EDITION (facultative)

Exemple :

```

TYPE pompe ;
PARAMETRE_LOI
    lambda      PAR_DEFAULT 1.e-8 ;
    beta_gamma PAR_DEFAULT 1.e-7 ;

```

6.2.4. Les variables d'état

A la définition d'un type de composant, on sait que certaines caractéristiques, contrairement à celles qui sont fixes et que l'on déclarera en constantes, pourront très bien changer au cours du temps en fonction des événements survenant sur le système (par exemple, la position ouvert ou fermé d'un disjoncteur, la présence ou non d'une panne).

Pour cela, toutes les caractéristiques susceptibles de changer sont décrites dans des **variables d'état**. Ces caractéristiques sont représentées en interne par des variables, **toujours univaluées**, et l'état du système à un moment donné sera défini par une instanciation possible de ces variables.

6.2.4.1. Notion de variable réinitialisée

Pour faciliter la modélisation d'une classe de systèmes en Figaro, il est souvent utile de distinguer deux types de variables :

- les variables **essentielles** de l'état d'un système (variables non réinitialisées),
- les variables dont la valeur, **pour un état du système**, est **déduite** de la valeur des autres variables dans cet état par application des règles d'interaction (variables réinitialisées).

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

6.2.4.1.1. Variables essentielles

Ce sont des variables dont la valeur évolue simplement en fonction de l'occurrence des événements.

Par exemple, au TYPE pompe on peut associer la variable d'état booléenne "défaillant" qui est FAUX initialement et deviendra VRAI si l'un des deux événements "défaillance en fonctionnement" ou "défaillance à la sollicitation" se produit.

6.2.4.1.2. Variables réinitialisées

Ce sont des variables dont la valeur sera réinitialisée à une valeur donnée (valeur de réinitialisation) avant chaque passage dans les règles d'interaction. Autrement dit, pour un état du système donné, la valeur de ces variables sera toujours **déduite** de la valeur de réinitialisation et de la valeur des autres variables du modèle.

Prenons l'exemple d'un réseau maillé composé de sources, d'arcs et de nœuds. Cette classe de systèmes est régie par les lois suivantes :

- Une source est toujours alimentée.
- Un événement "défaillance arc" peut se produire sur chaque objet du type arc.
- Un nœud est alimenté s'il existe un nœud voisin alimenté tel que l'arc reliant les deux nœuds soit non défaillant.

Pour modéliser une telle classe de systèmes, deux variables d'état sont particulièrement intéressantes. La variable booléenne "alimenté" associée aux objets de type "noeud". La variable booléenne "défaillant" associé aux objets du type "arc".

La variable "défaillant" évolue pour un arc en fonction de l'occurrence de l'événement "défaillance arc" sur cet arc. En revanche, pour un nœud donné, la variable "alimenté" doit être recalculée, après chaque occurrence d'événement, en fonction des nœuds voisins et du nouvel état des arcs du réseau (elle dépend uniquement des connexions du réseau). Cette variable est donc déduite par application des règles d'interaction. On la déclarera donc en Figaro comme une variable réinitialisée, la valeur de réinitialisation étant FAUX.

La notion de variable d'état réinitialisée présente un intérêt important du point de vue de l'exploitation des modèles Figaro. En effet, la valeur d'une telle variable étant déduite de la valeur des autres variables du modèle, il n'est pas nécessaire de stocker sa valeur lorsque l'on veut mémoriser un état du système. La section 9.4 explique des raisons plus profondes pour lesquelles il est important d'avoir un maximum de variables déduites dans un modèle Figaro.

La valeur de réinitialisation d'une variable doit être, obligatoirement, une constante.

6.2.4.2. Catégories de variables d'état en Figaro

On distingue trois formes de variables d'état en Figaro : les ATTRIBUTS, les PANNES et les EFFETS.

Les **attributs** regroupent toutes les variables d'état qui ne sont ni des pannes ni des effets. On peut définir des attributs simples (non réinitialisés) ou des attributs réinitialisés (facette REINITIALISATION).

Les **pannes** sont des variables d'état booléennes non réinitialisées qui seront utilisées pour détecter la présence ou l'absence d'une défaillance sur un objet. Ces variables d'état sont associées aux événements déclarés dans les règles d'occurrence et aux modèles de fiabilité qui seront exploités pour la génération d'arbres de défaillances.

Les **effets** sont des variables d'état booléennes réinitialisées à FAUX. Ils ont été créés pour simplifier la déclaration de telles variables, parce qu'elles sont très employées, surtout dans les bases de connaissances destinées à être exploitées pour générer des arbres de défaillances.

6.2.4.3. ATTRIBUT

Syntaxe:

```
ATTRIBUT      nom_attribut
[ LIBELLE      "description" ]
[ DOMAINE_ENTIER | REEL | BOOLEEN | énumération_de_symboles ]
[ PAR_DEFAULT valeur | expression ]
[ REINITIALISATION valeur | expression ]
[ EDITION      option_edition ] ;
```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Les attributs sont des variables d'état caractéristiques de la classe. Chaque attribut est associé à un domaine de définition et peut être associé à une valeur initiale par défaut.

Exemple :

```

TYPE disjoncteur ;
ATTRIBUT
    position DOMAINE 'ouvert' 'ferme'
        PAR_DEFAULT 'ferme' ;

```

A chaque attribut peuvent être associées plusieurs facettes :

Facette DOMAINE (obligatoire)

Permet de définir l'ensemble des valeurs possibles pour l'attribut. Cette facette est identique à celle des CONSTANTES.

Exemple :

```

TYPE vanne SORTE_DE composant_thermohydraulique;
ATTRIBUT
    debit DOMAINE REEL ;
    pression DOMAINE 'faible' 'moyenne' 'forte' ;

```

Facette LIBELLE (facultative)

(cf. CONSTANCE)

Facette PAR_DEFAULT (facultative)

Cette facette accepte une valeur précise appartenant au domaine de définition de l'attribut, ou une expression (l'ensemble des valeurs possibles pour l'expression doit être inclus dans le domaine de définition de l'attribut).

Remarque sur l'affectation d'une valeur initiale par défaut sous forme d'expression :

Lorsque dans une base de connaissances, on définit une valeur initiale par défaut sous forme d'expression pour un attribut, il faudra veiller à ce que l'utilisation de cette base rende obligatoire la saisie d'une valeur pour les champs intervenant dans l'expression (en utilisant l'option OBLIGATOIRE pour la facette EDITION par exemple). En effet, si un champ de l'expression n'a pas de valeur initiale, le traitement ne pourra évaluer l'expression.

Facette REINITIALISATION (facultative)

Cette facette permet de déclarer qu'un attribut est une variable réinitialisée et de lui affecter une valeur de réinitialisation (cf. Paragraphe "Variables réinitialisées" de ce chapitre). Cette facette n'est autorisée que pour les variables déclarées comme attributs dans la base de connaissance.

La valeur de réinitialisation doit obligatoirement être une constante.

6.2.4.4. PANNE

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Syntaxe :

```

PANNE
nom_panne
[ LIBELLE      "description" ]
[ PAR_DEFAULT VRAI | FAUX ]
[ EDITION      option_edition ]
[ DONNEES_FIABILISTES
  [SI expression booléenne constante]
  [GROUPE nom(s) de groupe(s)]
  [MODELE_PAR_DEFAULT nom_de_modele]
  [modele_fiabiliste]
  ...
  [modele_fiabiliste]
] (fin de la rubrique optionnelle DONNEES_FIABILISTES)

; (un unique ";" termine la déclaration de chaque panne)

```

Lorsqu'on réalise l'étude de sûreté de fonctionnement d'un système, on s'intéresse principalement aux conséquences des dysfonctionnements des composants sur le fonctionnement du système. C'est pourquoi de nombreux événements modélisés dans une classe d'objets sont des événements représentatifs des pannes pouvant survenir sur les objets de cette classe.

Exemple : l'événement "défaillance en fonctionnement" peut survenir sur les objets de type pompe.

Pour décrire les conséquences d'un "événement panne" sur le système, il est souvent utile d'associer aux classes d'objets des variables d'état booléennes représentatives de l'absence ou de la présence d'une panne.

Le langage Figaro permet de regrouper ces variables particulières sous la catégorie de champs **PANNE**.

Exemple :

```

TYPE pompe ;
PANNE
  defaillance_en_fonctionnement ;

```

Remarque : Il ne faut pas confondre "l'événement panne", qui sera modélisé dans les règles d'occurrence, avec la "variable panne" qui mémorise si cet événement s'est produit ou non.

Les variables d'état de la catégorie **PANNE** sont toujours des variables booléennes (**VRAI** si la panne s'est produite, **FAUX** sinon). Leur valeur initiale est **FAUX** si elle n'est pas spécifiée, comme dans l'exemple ci-dessus.

La modélisation des pannes s'est fortement enrichie au moment où on a souhaité pouvoir capitaliser dans les bases de connaissances Figaro différents modèles de petits processus stochastiques qui pourraient être associés aux événements de base des arbres de défaillances générés (cf. [3] pour la définition mathématique des arbres de défaillances de ce genre). C'est alors qu'on a ajouté au langage la rubrique optionnelle **DONNEES_FIABILISTES** pour décrire une panne.

Dans cette rubrique on trouve :

- Une condition (facultative) qui ne doit dépendre que de constantes (par exemple : pour considérer la défaillance refus d'ouverture d'une vanne, on mettra la condition `position_initiale = 'fermee'`) ;
- Un ensemble de groupes de règles (cf. §7.4) dans lesquels les modèles fiabilistes dont il est question ci-après doivent être considérés ;
- Un ensemble de modèles fiabilistes avec leurs paramètres et la mention d'un modèle par défaut : tous ces modèles seront présents dans l'arbre de défaillances généré, mais un seul sera « actif » : le modèle par défaut.

Exemple typique de déclaration de **PANNE** avec deux modèles de fiabilité :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

PANNE def_fonc
LIBELLE "défaillance en fonctionnement"
DONNEES_FIABILISTES
  SI etat_initial = 'marche'
    GROUPE groupe_add
    MODELE_PAR_DEFAUT MODELE_GLM
    MODELE_GLM
      GAMMA gamma
      LAMBDA lambda
      MU mu
    MODELE_GLTM
      GAMMA gamma
      LAMBDA lambda
      TM temps_de_mission ;

```

Les modèles fiabilistes associés à une PANNE peuvent être utilisés de deux manières dans l'exploitation d'un modèle Figaro :

- En chaînage avant, ils sont automatiquement remplacés par un ensemble de règles d'occurrence écrites de façon à produire le comportement théorique décrit en annexe 2.
- En chaînage arrière, lorsque l'algorithme décrit au § 2.8.3 doit expliquer un fait du type `nom_panne = VRAI`, il crée un événement de base de l'arbre de défaillances en cours de construction et lui affecte les modèles associés à la PANNE `nom_panne`.

L'ensemble de tous les modèles possibles est en annexe 2. Cette liste des modèles disponibles en Figaro a été obtenue par compilation des modèles exploitables dans différents outils de calcul des arbres de défaillances.

Voici à titre d'exemple le comportement spécifié par le modèle fiabiliste le plus utilisé, appelé **MODELE_GLM**. Ce modèle représente un composant réparable qui peut tomber en panne à $t=0$ ou en fonctionnement.

Ses paramètres sont : **GAMMA** (taux de défaillance à la sollicitation), **LAMBDA** (taux de défaillance en fonctionnement), **MU** (taux de réparation).

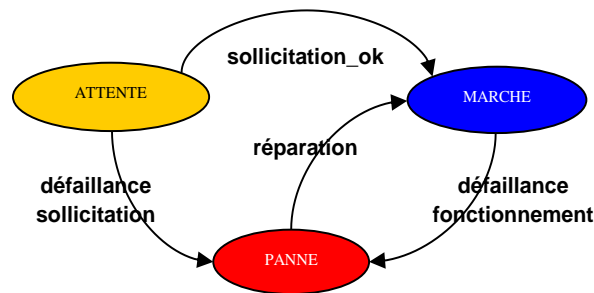


Figure 5. Graphe d'états du modèle **MODELE_GLM**

6.2.4.5. EFFET

Syntaxe:

```

EFFET nom_effet
  [ LIBELLE "description" ]
  [ EDITION [ [NON] VISIBLE ] ] ;

```

Les effets **sont équivalents à des variables booléennes réinitialisées à FAUX**, mais étant donné leur emploi très fréquent (surtout dans les modèles destinés à la construction d'arbres de défaillances), il est utile de les distinguer, car cela allège beaucoup leur déclaration (en général il suffit de déclarer le nom de l'effet). Un effet peut être déclaré **NON VISIBLE** si on estime que sa valeur n'a pas d'intérêt pour l'utilisateur final de la base de connaissances.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Lorsqu'une variable est une variable réinitialisée à FAUX, cela signifie que, si aucune règle d'interaction ne permet de **déduire** la valeur de cette variable, celle-ci est FAUSSE. C'est pourquoi un effet correspond à un comportement d'un composant, généralement de nature fonctionnelle (obstruction, fuite, etc.), observé à la suite de la réalisation d'un ensemble de causes. Il disparaît dès que ses causes disparaissent : on dit qu'*un effet est non rémanent*. Les effets sont à rapprocher de la notion d'événement intermédiaire dans les arbres de défaillances.

6.3. Les règles

Comme nous l'avons vu au chapitre 2, les règles d'un type permettent de définir le comportement dynamique des objets d'un type. Les règles décrivent les événements pouvant survenir sur les différents types modélisés et leurs conséquences sur le système. Il existe deux types de règles en Figaro.

Les **règles d'occurrence** permettent de modéliser l'occurrence des événements qui peuvent se produire sur un objet. Elles sont de la forme :

SI <condition>

IL_PEUT_SE_PRODUIRE <événement>

PROVOQUE <actions>

- * La condition exprime la condition que doit vérifier l'état du système pour qu'un événement se déclenche sur un objet d'une classe. Cette condition peut dépendre à la fois de variables **essentiels** et **déduites**.
- * Les actions décrivent les conséquences de l'événement sur les variables d'état **essentiels** de la classe.

Les **règles d'interaction** modélisent la propagation des changements d'états provoqués par le déclenchement d'un événement. Elles sont de la forme :

SI <condition>

ALORS <actions>

SINON <actions>

Pour ces règles, la condition (SI) permet par sa réalisation (ALORS) ou par sa non réalisation (SINON) le déclenchement d'un ensemble d'actions affectant les variables **essentiels** et **déduites** du modèle.

Exploiter un modèle Figaro (le faire tourner), c'est appliquer les règles de la base de connaissances à la base de faits décrivant le système à étudier. Les règles d'un modèle Figaro peuvent être exploitées en chaînage avant (pour générer le graphe d'états du système par exemple) ou en chaînage arrière (pour générer des arbres de défaillances par exemple).

6.3.1. Les conditions

Les conditions des règles Figaro sont des expressions booléennes complexes ou non. Elles correspondent à des tests sur l'état du système qui vont être évalués par les traitements (chaînage avant ou arrière) pour tester si la règle est applicable. Si la condition d'une règle d'un objet est évaluée à VRAI pour les valeurs des variables dans l'état courant du système, alors la règle est déclenchable et les actions peuvent être appliquées.

La syntaxe des expressions utilisables pour définir les conditions des règles a été complètement définie précédemment. Rappelons simplement que, pour un type donné, on peut utiliser, dans l'expression associée à la condition d'une règle :

- les champs du type,
- les champs des objets en interface en définissant un ensemble d'objets par une interface et en utilisant les opérateurs IL_EXISTE, IL_EXISTE AU_MOINS, QQSOIT ou SOIT,
- les champs des objets d'un type particulier du modèle en définissant l'ensemble d'objets par le nom du type et en utilisant les opérateurs IL_EXISTE, IL_EXISTE AU_MOINS, QQSOIT ou SOIT,
- les champs des OBJET_SYSTEME (variables globales).

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

L'exemple suivant décrit une règle d'interaction pour une pompe de secours qui se déclenche (*etat* passe à 'demande_demarrage') uniquement si la pompe secourue est en panne.

Exemple :

```

TYPE pompe ;
PANNE
    def_soll ;
    def_fonc ;

TYPE pompe_secours SORTE_DE pompe ;
ATTRIBUT
    etat DOMAINE 'arret' 'demande_demarrage' 'marche'
        PAR_DEFAULT 'arret' ;
INTERFACE
    pompe_secourue GENRE pompe CARDINAL 1;
INTERACTION
    SI PANNE( pompe_secourue ) ET MARCHE
        ET etat = 'arret'
    ALORS
        etat <-- 'demande_demarrage' ;

```

6.3.2. Les actions

Les actions sont les opérations qui permettent de modifier les variables d'état du modèle, faisant ainsi évoluer l'état du système. Une action peut donc modifier uniquement des attributs, des pannes ou des effets. Pour un type donné, on peut modifier par une action les **variables d'état** :

- du type,
- des objets en interface en définissant un ensemble d'objets par une interface et en utilisant les opérateurs POUR_TOUT ou SOIT,
- des objets d'un type particulier du modèle en définissant l'ensemble d'objets par le nom du type et en utilisant les opérateurs POUR_TOUT ou SOIT,
- des champs des OBJET_SYSTEME (variables globales).

Une règle peut provoquer plusieurs **actions élémentaires** (liste d'actions). Dans ce cas, il faut les lier par des virgules. Il existe trois sortes d'actions élémentaires.

6.3.2.1. Affectation d'une valeur à une variable

L'affectation d'une nouvelle valeur à un attribut, une panne ou un effet est réalisée grâce à la facette <--.

```
<variable> <-- <nouvelle valeur>
```

La nouvelle valeur doit bien sûr appartenir au domaine de valeurs possibles spécifié pour la variable.

Afin d'alléger certaines bases de connaissances, la gestion des variables booléennes et notamment des variables PANNE et EFFET est simplifiée. Ainsi, **pour une variable booléenne** :

```

<variable> <-- VRAI      est équivalent à      <variable>
<variable> <-- FAUX     est équivalent à      NON <variable>

```

Ces deux notations sont strictement équivalentes.

Exemples :

```

... ALORS position DE x <-- 'ouvert' ,
    valeur DE y <-- valeur DE y + poids DE y ;
(* la règle exécute deux actions élémentaires *)

... ALORS refus_demarrage ;
(* refus_demarrage est un attribut booléen, une panne ou un effet *)

... ALORS NON alimente ;

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

(* alimente est un attribut booléen, une panne ou un effet *)

6.3.2.2. Action sur un ensemble d'objets

L'opérateur POUR_TOUT permet de modifier successivement les attributs d'un ensemble d'objets. Il agit comme les opérateurs complexes IL_EXISTE, IL_EXISTE AU_MOINS, QQSOIT, SOMME et PRODUIT.

POUR_TOUT x **UN** <ensemble d'objets>

Syntaxe :

```

POUR_TOUT variable
    UN | UNE      nom_d_ensemble
    [ DE_TYPE      type ]
    [ VERIFIANT    condition ]
    FAIRE action

```

POUR_TOUT définit une variable *liée* à l'opérateur qui prendra successivement toutes les valeurs de l'ensemble. L'ensemble est soit un nom d'interface (multivaluée) soit un nom de type. Comme pour les opérateurs complexes, il est possible de préciser des contraintes optionnelles (DE_TYPE et VERIFIANT) pour effectuer l'action (facette FAIRE) uniquement sur un sous-ensemble d'objets.

Le mécanisme de cet opérateur est le suivant :

- Si l'ensemble est non vide, on effectue pour chaque objet de l'ensemble, l'action précisée par la facette FAIRE.
- Si l'ensemble est vide, aucune action n'est effectuée.

Exemples :

```

POUR_TOUT x UN <ensemble d'objets>
    FAIRE <affectation>

ALORS POUR_TOUT x UNE commande
    FAIRE position DE x <-- 'ouvert' ;

(* Imbrication d'opérateurs POUR_TOUT *)
ALORS POUR_TOUT x UN amont
    FAIRE ( POUR_TOUT y UN aval DE x
            VERIFIANT MOI_MEME <> y
            FAIRE valeur DE y <-- valeur DE x ) ;

```

Remarque : La portée de la variable liée est réduite à l'opérateur POUR_TOUT. Elle n'est donc accessible que dans l'action précisée par FAIRE. Pour combiner plusieurs actions sous une même facette FAIRE, il faut utiliser des parenthèses et non des virgules. En effet, la virgule a une priorité plus élevée que le mot-clé FAIRE. Les exemples suivants éclairent cet aspect :

Exemple :

```

ALORS POUR_TOUT x UN amont
    FAIRE etat DE x <-- 'arret', position DE x <-- 'ouvert' ;

```

Cet exemple provoque une erreur car Figaro assimile cette phrase comme deux actions successives après le mot-clé ALORS et non pas après le mot-clé FAIRE :

```

(* Voici comment Figaro lit la phrase précédente *)
ALORS POUR_TOUT x UN amont
    FAIRE etat DE x <-- 'arret',
    position DE x <-- 'ouvert' ; (* x est inconnu !!! *)

```

Il aurait fallu écrire :

```

ALORS POUR_TOUT x UN amont
    FAIRE ( etat DE x <-- 'arret', position DE x <-- 'ouvert' ) ;
(* x est connu : les parenthèses sont nécessaires *)

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

6.3.2.3. Résolution de système d'équations

Syntaxe :

... **ALORS RESOUDRE_SYSTEME** (nom_de_systeme) ,

L'action RESOUDRE_SYSTEME permet de préciser le système d'équations à résoudre (cf. § 7.6). Le mécanisme de résolution est le suivant :

Si la règle est déclenchable, Figaro résout le système d'équations.

- Les valeurs des inconnues sont alors calculées.
- Comme ces inconnues représentent des attributs, Figaro leur affecte leur nouvelle valeur.

La valeur des attributs/inconnues a donc été modifiée. L'état du système est modifié.

La principale difficulté que l'on rencontre lorsqu'on utilise des systèmes d'équations est le fait que suivant la configuration du système au moment où on lance la résolution, il est possible de tomber sur une résolution impossible ou donnant une infinité de valeurs possibles. Par exemple dans un système électrique, à partir du moment où un sous-ensemble est isolé des sources de tension, il devient impossible de calculer sa tension.

Pour éviter cet écueil, il faut conditionner les équations grâce à leur facette SI (cf. § 7.6), de façon à faire complètement disparaître les équations problématiques. Pour l'exemple du système électrique, la condition d'une équation sur les tensions doit vérifier que le composant concerné est bien relié à une source de tension (ce qui peut aisément être fait par des règles d'interaction propageant un flux de proche en proche).

6.3.3. Les règles d'occurrence

Les règles d'occurrence d'un type modélisent l'occurrence des événements possibles et leurs conséquences sur ce type (et quelquefois sur des types accessibles via les interfaces, comme dans l'exemple de base de connaissances du § 5.3.7, où le tir d'une transition affecte les places amont et aval de la transition).

L'occurrence d'un événement est fonction d'une certaine loi de probabilité. On distingue plusieurs types d'événements possibles, en fonction de la loi de probabilité associée à l'événement :

- Les événements associés à une loi instantanée : ces événements se produisent instantanément, dès lors que leur condition d'occurrence est vérifiée. Le paramètre loi associé est un taux d'occurrence sans dimension.
- Les événements dont l'occurrence est différée dans le temps (par rapport au moment où leurs conditions d'occurrence sont vérifiées). Parmi ceux-ci, deux cas particuliers sont fréquemment rencontrés :
 - o les événements associés à une loi exponentielle : ces événements peuvent se produire au bout d'un temps aléatoire. Le paramètre de la loi est un taux d'occurrence horaire.
 - o les événements associés à une loi temps constant : ces événements se produisent de façon certaine au bout d'un temps défini T, si les conditions d'occurrence de l'événement sont maintenues pendant toute la durée de T.

Une règle d'occurrence peut être associée à un seul événement s'il est temporisé, mais à un ou plusieurs événements s'ils sont instantanés.

Règles d'occurrence instantanées

Lorsqu'un système est dans un état donné, il peut arriver qu'il ait le choix entre plusieurs évolutions incompatibles. C'est le cas par exemple lorsqu'un composant sollicité peut démarrer ou refuser de démarrer. Le système a le choix entre ces deux évolutions, mais l'une des deux **doit forcément** se produire. Ainsi, les règles d'occurrence instantanées permettent de modéliser l'occurrence d'événements :

- incompatibles entre eux (exclusifs),
- dont l'un au moins doit se produire instantanément.

Une règle d'occurrence instantanée aura donc une syntaxe de la forme :

SI <condition>

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

IL_PEUT_SE_PRODUIRE      <événement>
                           PROVOQUE <actions>
                           OU_BIEN
                           <événement>
                           PROVOQUE <actions>
                           OU_BIEN ...

```

Chaque événement est associé à une loi instantanée. **La somme des paramètres loi des événements de la règle doit obligatoirement être égale à 1** (un des événements doit obligatoirement se produire).

*Remarque : pour l'événement déclenché par la **dernière** facette OU_BIEN, il n'est pas obligatoire de renseigner le paramètre loi associé à l'événement. Si celui-ci n'est pas renseigné, Figaro ajoutera automatiquement un paramètre loi instantanée de valeur égale à :*

1 - \sum (paramètres loi des autres événements déclenchés par la règle)

Ordre d'application des règles d'occurrence en chaînage avant

Dans les chapitres 2 et 3, nous avons étudié le principe d'exploitation des règles d'un modèle Figaro en chaînage avant. Nous avons vu que, lorsque le système est dans un état dans lequel aucune règle d'occurrence avec des transitions instantanées ne s'applique, le traitement trouve les règles d'occurrence applicables, choisit une de ces règles et l'applique pour simuler un événement.

Lorsque dans un état du système, le traitement trouve des règles d'occurrence instantanées applicables, cela signifie qu'un événement de chaque règle et un seul doit être déclenché instantanément. Ainsi, le traitement ne choisit pas **un** événement mais une **combinaison** d'événements formée d'un événement de chaque règle applicable.

Ensuite, le traitement applique successivement chaque événement de la combinaison en déclenchant les actions provoquées par les événements. Il est important de noter que l'ordre d'application des événements de la combinaison doit être indifférent. Tout se passe alors comme si les événements étaient appliqués en parallèle. Ainsi, le concepteur de base de connaissances devra veiller à ce que les règles d'occurrence instantanées du système susceptibles d'être applicables simultanément ne concluent pas sur des faits incompatibles. Pour plus de détail, cf. le type d'incohérence **Inc2**, au § 9.2.

Variables Panne et Événements

De façon qualitative, on peut classer les événements susceptibles de survenir sur un système en quatre catégories :

- Les **transitions** sont des événements décrivant le comportement normal des composants. Exemples : démarrage d'une pompe après sollicitation, transition jour-nuit...
- Les **défaillances** sont des événements décrivant les défaillances des composants. Exemples : défaillance à la sollicitation d'une pompe, ouverture intempestive d'un disjoncteur...
- Les **indisponibilités** représentent un type particulier de défaillance à la sollicitation. Ce sont le plus souvent des événements qui se sont produits dans le passé du système et qui se révèlent à la première sollicitation des composants. (Exemple : oubli fermé d'une vanne dont la position est normalement ouvert). Le traitement des événements de type INDISPONIBILITE sera exactement le même que le traitement des défaillances à la sollicitation. Autrement dit, l'information INDISPONIBILITE est une information purement qualitative par rapport à une simple défaillance à la sollicitation.
- Les **réparations** représentent des changements d'états dus aux opérations de maintenance sur un système. Exemple : réparation d'une pompe suite à une défaillance à la sollicitation ou en fonctionnement de la pompe.

Ainsi, pour décrire un événement dans une règle d'occurrence on pourra écrire :

```

IL_PEUT_SE_PRODUIRE TRANSITION <événement>
ou
IL_PEUT_SE_PRODUIRE DEFAILLANCE <événement>
ou

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

IL_PEUT_SE_PRODUIRE **INDISPONIBILITE** <événement>

ou

IL_PEUT_SE_PRODUIRE **REPARATION** <événement>

L'intérêt de cette classification est double : tout d'abord, elle permet de mieux présenter les sorties de codes de calcul (par exemple en affichant les défaillances en rouge et les réparations en vert dans un affichage de séquences). Ensuite, elle permet des raccourcis d'écriture grâce à l'association à des variables PANNE.

- Un événement du type DEFAILLANCE ou INDISPONIBILITE peut ou non être associé à une variable PANNE.

- * Si l'événement est associé à une variable PANNE, c'est-à-dire si l'identificateur d'une variable PANNE est le même que l'identificateur de l'événement, la règle décrivant l'occurrence de la défaillance ou de l'indisponibilité pourra s'écrire de façon plus concise. Implicitement, cette règle est conditionnée par le fait que la panne associée est à FAUX ; de plus, le déclenchement de cet événement provoque la mise à VRAI de la variable PANNE associée ;
- * En revanche, si l'événement n'est pas associé à une variable PANNE, l'utilisation des mots clés DEFAILLANCE ou INDISPONIBILITE permettra uniquement de fournir une information qualitative sur l'événement.

- De même, un événement du type REPARATION peut ou non être associé à une *ou plusieurs* variables PANNE.

- * Si l'événement est associé à une ou plusieurs variables PANNE, c'est-à-dire si on décrit explicitement que l'événement REPAIRE une ou plusieurs variables PANNE, la règle décrivant l'occurrence de la réparation pourra s'écrire de façon plus concise. Implicitement, cette règle est conditionnée par le fait qu'au moins une des pannes associées est à VRAI ; de plus, le déclenchement de cet événement provoque la mise à FAUX de *toutes* les variables PANNE associées ;
- * En revanche, si l'événement n'est associé à aucune variable PANNE, l'utilisation du mot clé REPARATION permettra uniquement de fournir une information qualitative sur l'événement.

Lors de l'instanciation en Figaro 0 toutes les conditions et actions implicites citées ci-dessus sont explicitées, car la PANNE est transformée en un simple ATTRIBUT.

6.3.3.1. Syntaxe d'une règle d'occurrence

```

OCCURRENCE [nom_de_règle]
[ GROUPE      liste_groupes ]
[ SI          condition]

IL_PEUT_SE_PRODUIRE
TRANSITION   nom_transition
              [ LIBELLE      "description" ]
              LOI            loi_et_paramètres
              [ PROVOQUE     actions ]

| DEFAILLANCE nom_de_defaillance | nom_de_panne
              [ LIBELLE      "description" ]
              LOI            loi_et_paramètres
              [ PROVOQUE     actions ]

| REPARATION  nom_reparation
              [ LIBELLE      "description" ]
              LOI            loi_et_paramètres
              [ REPAIRE      liste_pannes ]
              [ PROVOQUE     actions ]

| INDISPONIBILITE nom_indisponibilité | nom_de_panne
              [ LIBELLE      "description" ]
              LOI            loi_et_paramètres
              [ PROVOQUE     actions ]

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```
[ OU_BIEN      TRANSITION ... | DEFAILLANCE ... |
  REPARATION ... | INDISPONIBILITE ... ]
[ OU_BIEN      ... ]
```

Remarque : La syntaxe des règles d'occurrence des objets est semblable à celle des règles d'occurrence des types, à une exception près : la facette REPAIRE n'est pas autorisée dans un objet.

Les règles d'occurrence définies pour un type sont introduites par le mot-clé OCCURRENCE.

La syntaxe d'une règle d'occurrence est la suivante :

Nom de règle (facultatif)

Permet d'attribuer un nom à une règle. C'est utile pour "commenter" les règles et pour les repérer dans les outils de recherche de bogues.

Exemple *def_fonc* est le nom de la règle :

```
TYPE pompe ;
ATTRIBUT
  etat DOMAINE 'marche' 'arret' ;
OCCURRENCE
  def_fonc
  SI etat = 'marche'
  IL_PEUT_SE_PRODUIRE
    ..... ;
```

Facette GROUPE (facultative)

Permet de spécifier l'appartenance de la règle à un ou plusieurs groupes (cf. Notions avancées, § 7).

Facette SI (facultative)

Décrit la condition nécessaire pour que la règle d'occurrence soit déclenchable. A cette condition explicite s'ajoutent les conditions implicites expliquées plus haut.

Facette IL_PEUT_SE_PRODUIRE (obligatoire)

Permet de décrire quels sont les événements qui se déclenchent lorsque la condition de la règle est vérifiée pour l'état du système. Chaque événement est identifié par son nom et qualifié par le mot-clé décrivant la nature de l'événement (TRANSITION, DEFAILLANCE, REPARATION, INDISPONIBILITE).

Exemple :

```
TYPE pompe ;
ATTRIBUT
  etat DOMAINE 'marche' 'arret' ;
OCCURRENCE
  SI etat = 'marche'
  IL_PEUT_SE_PRODUIRE
    DEFAILLANCE def_fonc
    ..... ;
```

Dans le cas d'une règle instantanée contenant plusieurs événements :

- Le premier est introduit par la facette IL_PEUT_SE_PRODUIRE
- Les événements suivants sont introduits par la facette OU_BIEN .

Exemple :

```
TYPE pompe ;
```


EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

PANNE defaillance_soll ;
ATTRIBUT
    etat DOMAINE 'marche' 'arret' 'sollicite' ;
OCCURRENCE
    SI etat = 'sollicite'
    IL PEUT SE PRODUIRE
        DEFAILLANCE defaillance_soll
            PROVOQUE etat <-- 'arret'
            LOI INS (gamma)
    OU BIEN
        TRANSITION demarrage
            PROVOQUE etat <-- 'marche';

```

Chaque événement de la règle peut être décrit précisément par les facettes suivantes :

Facette LIBELLE (facultative)

Permet de mettre un texte en clair (entre guillemets) descriptif de l'événement. Cette facette libellé est facultative (mais recommandée pour obtenir des résultats d'étude lisibles).

Facette REPAIRE (facultative)

Elle n'est présente que lorsque le type d'événement est REPARATION (cf. Paragraphe Réparations). Elle permet de déclarer la liste des pannes supprimées à la suite d'une réparation. La facette doit être suivie des noms des variables PANNE, séparés par des virgules.

Facette LOI (obligatoire)

Permet de préciser la loi de probabilité associée à l'événement. La loi sera déclarée avec une syntaxe de la forme :

nom_de_loi (expression_numérique, expression_numerique...)

Cette facette est obligatoire *sauf* pour le dernier événement associé à une règle instantanée (dernière facette OU_BIEN de la règle). Si elle n'est pas renseignée, Figaro rajoutera automatiquement un paramètre de loi instantanée de valeur égale à 1- somme des paramètres de loi des autres événements contenus dans la règle.

Les noms de lois les plus utilisés sont EXP ou EXPONENTIELLE (loi exponentielle), INS ou INSTANTANEE (loi instantanée), T_C ou TEMPS_CONSTANT (distribution de Dirac). Chacune de ces lois a un seul paramètre, dont la dimension est l'inverse d'un temps pour EXP (taux de transition), un temps pour T_C. Le paramètre de la loi INS est une probabilité, donc il est sans dimension.

Il existe tout un catalogue d'autres lois : on le trouve en annexe de la note décrivant la syntaxe de Figaro [5]. Toutes ces lois, à l'exception de la loi INS (pour laquelle cela n'a pas de sens), de la loi EXP (sans mémoire par définition) ont une variante dite « à mémoire ». Le nom d'une loi à mémoire est le nom de la loi de base suivie du suffixe _M. L'utilité de ces lois à mémoire a été décrite dans le § 3.4.

L'expression numérique permettant d'attribuer une valeur à un paramètre de la loi associée à l'événement peut être principalement :

- Une expression constante faisant intervenir des champs PARAMETRE_LOI et des champs CONSTANTE (c'est le cas le plus fréquent) :

Exemple :

```

TYPE pompe ;
ATTRIBUT
    etat DOMAINE 'marche' 'arret' ;
PARAMETRE_LOI
    lambda_pompe;
OCCURRENCE

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

SI etat = 'marche'
IL PEUT SE PRODUIRE
  DEFAILLANCE defaillance_fonc
  LOI EXP (lambda_pompe)
  ..... ;

```

- Une expression faisant intervenir des attributs *variables*. Ce cas de figure ne pose pas de problème pour les lois instantanées (on prend la valeur du paramètre au moment où on a besoin de faire le choix entre les alternatives), ni pour les lois exponentielles, sans mémoire. Pour toutes les autres lois, la signification d'un changement de paramètre qui se produit pendant que l'événement est valide (les conditions de son occurrence sont satisfaites) peut être sujette à discussion. C'est particulièrement visible pour la loi T_C, pour laquelle deux interprétations « extrêmes » sont possibles : soit on ignore tout changement de la valeur du paramètre tant que l'événement est valide, soit on recalcule une nouvelle date d'occurrence à chaque changement du paramètre. Suivant les cas, on peut avoir besoin de l'une ou de l'autre interprétation, c'est pourquoi on donne le choix à l'utilisateur dans les outils de calcul. Pour les autres lois, une seule option est proposée : celle qui consiste à recalculer la date d'occurrence à chaque changement de paramètre. Nous rappelons que le § 3.4 explique plus en profondeur les options que l'on aurait pu prendre pour la sémantique de ce type de transitions.

En résumé, il ne faut utiliser des paramètres variables pour les lois qu'avec de grandes précautions !

Facette PROVOQUE (facultative)

Indique les conséquences immédiates de l'événement, c'est-à-dire les **actions** à effectuer.

Nous allons maintenant donner quelques exemples, en mettant en exergue les catégories d'événements et leurs liens avec les pannes.

6.3.3.2. Exemple : transitions normales

Les *transitions normales* sont des événements décrivant le comportement normal du système étudié.

Exemple :

```

TYPE alternance_jour_nuit ;
PARAMETRE_LOI
  jn PAR_DEFAULT 12 ;
  nj PAR_DEFAULT 12;
ATTRIBUT
  valeur DOMAINE 'jour' 'nuit' ;
OCCURRENCE
  SI valeur = 'jour'
  IL PEUT SE PRODUIRE
  TRANSITION pjn
    LIBELLE "passage de jour à nuit"
    LOI TEMPS_CONSTANT ( jn )
    PROVOQUE valeur <-- 'nuit' ;

  SI valeur = 'nuit'
  IL PEUT SE PRODUIRE
  TRANSITION pnj
    LIBELLE "passage de nuit à jour"
    LOI TEMPS_CONSTANT ( nj )
    PROVOQUE valeur <-- 'jour' ;

```

Ce type permet de définir un objet qui va traduire l'alternance entre le jour et la nuit (utile par exemple si certains événements ne peuvent intervenir que la nuit dans le fonctionnement d'un système). Les identificateurs pjn et pnj sont des informations purement qualitatives qui pourront être exploitées dans les sorties d'un calcul pour désigner ces événements.

6.3.3.3. Défaillances

On définit deux types de défaillances : les défaillances à la sollicitation et les défaillances en fonctionnement.

Les sollicitations contiennent la description de ce qui va se produire de manière instantanée (loi instantanée)

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

quand on va demander à un objet de réaliser l'une des fonctions pour laquelle il a été conçu. Cet objet pourra alors réagir de deux manières :

- soit il refuse d'adopter le comportement qu'on lui demande : on dit qu'il y a une *défaillance à la sollicitation*,
- soit il réalise ce qu'on attend de lui : la sollicitation est réussie et va être décrite éventuellement par une transition normale.

Les *défaillances en fonctionnement* sont des défaillances qui surviennent pendant le fonctionnement du système, sans qu'on explicite leur cause (loi exponentielle).

- Lorsque l'événement défaillance n'est pas associé à une variable PANNE, le mot-clé DEFAILLANCE est une information qualitative qui ne sera pas exploitée par les traitements.

- Lorsque l'événement défaillance est associé à une variable PANNE (en précisant le nom d'une variable panne comme nom d'événement de défaillance), le processus suivant s'applique :

1. Par définition, une défaillance ne peut se produire que si le composant n'est pas déjà défaillant.

Ainsi il n'est pas nécessaire d'intégrer à la condition d'occurrence de l'événement, la condition '<variable de panne> = FAUX'. Celle-ci est implicite et sera prise automatiquement en compte par les traitements.

2. De même, l'occurrence d'un événement défaillance provoque toujours la mise à VRAI de la variable PANNE associée.

Il n'est donc pas nécessaire de décrire explicitement cette action dans les règles.

Exemple de défaillance à la sollicitation :

Prenons l'exemple d'un composant de type filtre. Lorsque ce composant est sollicité, deux alternatives peuvent se produire :

- * Le filtre peut être bouché, ce qui provoque la panne du composant. La probabilité de cet événement est une loi instantanée de paramètre g_bo.
- * Le filtre n'est pas bouché et la sollicitation réussit. La probabilité de cet événement est une loi instantanée de paramètre (1-g_bo).

Pour modéliser ce comportement, on écrit la règle d'occurrence suivante :

```

TYPE filtre SORTE_DE composant_thermohydraulique ;
ATTRIBUT sollicite DOMAINE BOOLEEN PAR_DEFAULT VRAI;
PANNE bouchage ;
PARAMETRE_LOI g_bo ;
OCCURRENCE
  SI sollicite
    IL_PEUT_SE_PRODUIRE
    DEFAILLANCE bouchage
      LIBELLE "bouchage à la sollicitation"
      LOI_INSTANTANEE ( g_bo )
      PROVOQUE sollicite <-- FAUX
    OU_BIEN
    TRANSITION non_bo
      PROVOQUE sollicite <-- FAUX;

```

Cette règle est comprise par Figaro comme si on avait écrit (en gras : les éléments qui étaient implicites dans la première version de la règle) :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

TYPE filtre SORTE_DE composant_thermohydraulique ;
ATTRIBUT sollicite DOMAINE BOOLEEN PAR_DEFAULT VRAI;
PANNE bouchage ;
PARAMETRE_LOI g_bo ;
OCCURRENCE
  SI sollicite ET NON bouchage
  IL PEUT_SE_PRODUIRE
  DEFAILLANCE bouchage
    LIBELLE "bouchage à la sollicitation"
    LOI INSTANTANEE ( g_bo )
    PROVOQUE sollicite <-- FAUX,
              bouchage <-- VRAI
  OU BIEN
  TRANSITION non_bo
    LOI INSTANTANEE (1-g_bo)
    PROVOQUE sollicite <-- FAUX ;

```

Exemple de défaillance en fonctionnement :

Prenons l'exemple d'un composant de type disjoncteur sur lequel peut survenir l'événement 'ouverture intempestive'.

```

TYPE disjoncteur SORTE_DE composant_electrique ;
ATTRIBUT position DOMAINE 'ouvert' 'ferme'
              PAR_DEFAULT 'ferme' ;
PANNE oi ;
OCCURRENCE
ouverture intempestive
SI position = 'ferme'
IL PEUT_SE_PRODUIRE
DEFAILLANCE oi
  LIBELLE "ouverture intempestive"
  LOI EXPONENTIELLE ( l_oi ) ;

```

Cette règle est comprise par Figaro comme si on avait écrit :

```

TYPE disjoncteur SORTE_DE composant_electrique ;
ATTRIBUT position DOMAINE 'ouvert' 'ferme'
              PAR_DEFAULT 'ferme' ;
PANNE oi ;
OCCURRENCE
ouverture intempestive
SI position = 'ferme' ET NON oi
IL PEUT_SE_PRODUIRE
DEFAILLANCE oi
  LIBELLE "ouverture intempestive"
  LOI EXPONENTIELLE ( l_oi )
  PROVOQUE oi <-- VRAI ;

```

6.3.3.4. Indisponibilités

Les *indisponibilités* représentent des événements qui se sont produits dans le passé du système et qui se révèlent à la première sollicitation des composants. Les indisponibilités sont donc un type particulier de défaillance à la sollicitation et elles sont toujours associées à une **loi instantanée**.

Les indisponibilités seront traitées exactement comme des défaillances à la sollicitation (cf. Paragraphe précédent). Autrement dit, si dans un système un composant est amené à être sollicité plusieurs fois, et si certaines défaillances ne peuvent se produire qu'à la **première sollicitation**, le concepteur de la base de connaissances devra modéliser différemment les conditions d'occurrence des événements indisponibilités et les conditions d'occurrence des événements défaillances à la sollicitation.

6.3.3.5. Réparations

Les *réparations* représentent les changements d'état dus à la maintenance du système. Ce sont des transitions qui ont pour effet de supprimer les pannes d'un objet. Pour cela on leur associe une facette indiquant les pannes supprimées par la réparation.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Exemple :

```

TYPE disjoncteur SORTE_DE composant_electrique ;
ATTRIBUT position DOMAINE 'ouvert' 'ferme'
                PAR_DEFAULT 'ouvert' ;

PANNE oi ;
PARAMETRE_LOI m_oi ;
OCCURRENCE
IL_PEUT_SE_PRODUIRE
REPARATION rep_oi
    LIBELLE "reparation de l'ouverture intempestive"
    REPARE oi
    LOI EXPONENTIELLE ( m_oi ) ;

```

Le mot-clé **REPARE** est suivi de la liste des pannes supprimées par la réparation en question.

Si l'événement **REPARATION** est associé à un ensemble de variables **PANNE** (par la facette **REPARE**), le processus suivant s'applique :

1. Par définition, une panne ne peut être réparée que si elle a eu lieu.

Ainsi il n'est pas nécessaire d'intégrer à la condition d'occurrence de l'événement, la condition '<variable de panne> = VRAI'. Celle-ci est implicite et sera prise automatiquement en compte par les traitements.

Lorsque plusieurs pannes ont été associées à une même réparation, cette dernière ne peut avoir lieu que si au moins une des pannes en question s'est produite.

2. De même, l'occurrence d'un événement réparation provoque toujours la mise à FAUX de la variable **PANNE** associée.

Il n'est donc pas nécessaire de décrire explicitement cette action dans les règles.

*Remarque : Les événements de type **REPARATION** sont traités de façon particulière par le générateur d'arbres de défaillances. Si une feuille de l'arbre correspond à une **PANNE** associée à un événement **REPARATION**, alors l'événement de base de la feuille sera déclaré comme un événement de base de type "GLM" et le paramètre loi de l'événement **REPARATION** sera associé à cet événement de base. Toutefois il est plus « propre » (c'est-à-dire que cela rend le modèle plus lisible) d'associer explicitement un modèle de panne GLM à la panne dans un groupe de règles dédié à la génération d'arbres de défaillances et de n'utiliser les règles d'occurrence que pour la simulation en chaînage avant (cf. § 7.4 sur les groupes de règles).*

6.3.4. Les règles d'interaction

Syntaxe d'une règle d'interaction :

```

INTERACTION [nom_de_regle]
    [ GROUPE      liste_groupes ]
    [ ETAPE      nom_etape ]
    [ SOIT variable UN nom_d_ensemble
      [ DE_TYPE type ]
      [ VERIFIANT condition ] ]
    [ SI          condition ]
    ALORS actions
    [ SINON      actions ]

```

Remarque : La syntaxe des règles d'interaction des objets est semblable à celle des règles d'interaction des types.

Les règles d'interaction permettent de propager les conséquences d'un événement à tout le système. Autrement dit, les règles d'interaction permettent d'exprimer les **relations déterministes** existant entre un objet et les autres objets du système. Ces relations peuvent s'exprimer principalement de deux façons :

- Soit en modifiant l'état d'un objet en fonction de l'état des autres objets.

```

TYPE pompe

TYPE pompe_secours SORTE_DE pompe;
ATTRIBUT

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

demande_demarrage DOMAINE BOOLEEN ;
INTERFACE secourue GENRE pompe
INTERACTION
SI IL_EXISTE x UNE secourue TELLE_QUE PANNE(x)
ALORS demande_demarrage ;

```

- Soit en modifiant l'état des autres objets en fonction de l'état d'un objet.

```

TYPE pompe
INTERFACE pompe_secours GENRE pompe ;
INTERACTION
SI PANNE
ALORS POUR_TOUT x UNE pompe_secours FAIRE demande_demarrage(x) ;

```

Les règles d'interaction définies pour un type sont introduites par le mot-clé **INTERACTION**.

Pour décrire une règle d'interaction, on dispose des facettes suivantes :

Nom de règle (facultatif)

Permet d'attribuer un nom à une règle. C'est utile pour "commenter" les règles et pour les repérer dans les outils de recherche de bogues.

Facette GROUPE (facultative)

Permet de spécifier l'appartenance de la règle à un ou plusieurs groupes (cf. § 7.4).

Facette ETAPE (facultative)

Permet de spécifier l'appartenance de la règle à **une** étape (cf. § 7.5).

Facette SI (facultative)

Décrit la condition nécessaire pour que la règle d'interaction soit déclenchable. Si aucune condition n'est associée à la règle (la règle doit se déclencher quel que soit l'état du système), la facette SI n'est pas mentionnée.

Facette ALORS (obligatoire)

Permet de spécifier les actions qui sont déclenchées lorsque la condition de la règle est VRAIE.

Facette SINON (facultative)

Permet de déclarer les actions qui sont déclenchées lorsque la condition de la règle est FAUSSE.

Exemple :

```

TYPE composant ;
ATTRIBUT
    alimente DOMAINE BOOLEEN PAR_DEFAULT FAUX ;
INTERFACE
    amont GENRE composant ;
INTERACTION
    SI MARCHE ET IL_EXISTE x UN amont TEL_QUE alimente (x)
    ALORS alimente <-- VRAI
    SINON alimente <-- FAUX ;

```

Facette SOIT (facultative)

Comme nous l'avons vu au chapitre 2, lorsqu'on applique les règles de la base de connaissances à la base de faits, une règle définie dans un type sera instanciée pour chaque objet de ce type dans le modèle à l'ordre 0. Les traitements examinent si la règle est applicable et l'appliquent éventuellement.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

La facette SOIT permet de définir une règle d'interaction (**condition et actions**) pour un **ensemble** d'objets défini par une variable liée (ensemble d'objets en interface ou ensemble d'objets d'un type particulier). Dans ce cas, la règle sera examinée ("instanciée") pour tous les couples d'objets définis par :

(Objet O1 du type possédant la règle, Objet O2 appartenant à l'ensemble défini par la facette SOIT)

Si l'ensemble défini par la facette SOIT est vide, la règle ne sera pas instanciée.

Prenons l'exemple d'une pompe qui a plusieurs pompes secours en interface. On veut exprimer le fait que, si la pompe tombe en panne, toutes les pompes secours qui sont à l'arrêt doivent être sollicitées. Autrement dit, on veut que pour chaque couple (pompe, pompe secours) du système la règle soit examinée. La facette SOIT permet d'écrire **une seule règle** pour modéliser ce processus :

```

TYPE pompe ;
INTERFACE
    pompe_secours GENRE pompe ;
ATTRIBUT
    etat DOMAINE 'marche' 'arret' 'sollicite' ;
PANNE
    defaillant ;
INTERACTION
    SOIT x UNE pompe_secours
    SI defaillant ET etat(x) = 'arret'
    ALORS etat(x) <-- 'sollicite' ;

```

Si un système contient une pompe P1 normale secourue par les pompes P2 et P3, cette règle sera instanciée deux fois :

- pour le couple (P1, P2)

```

SI defaillant(P1) ET etat(P2) = 'arret'
ALORS etat(P2) <-- 'sollicite' ;

```

- pour le couple (P1, P3)

```

SI defaillant(P1) ET etat(P3) = 'arret'
ALORS etat(P3) <-- 'sollicite' ;

```

La portée de la variable liée définie s'étend sur l'ensemble de la règle d'interaction. Ainsi, une variable définie par un SOIT pourra être utilisée dans les autres facettes de la règle (facettes SI, ALORS, SINON) si elle est décrite avant celles-ci.

L'ensemble d'objets défini par la facette SOIT est soit un nom de champ interface (multivaluée) soit une expression « OBJET DE_TYPE nom_type ». Comme pour les opérateurs complexes, il est possible de préciser des contraintes optionnelles (DE_TYPE et VERIFIANT) pour ne traiter qu'un sous-ensemble d'objets.

Exemple :

```

SOIT x UN <ensemble d'objets>
    VERIFIANT < conditions de restriction de l'ensemble >
    DE_TYPE < nom de type >

SOIT x UN amont
    DE_TYPE vanne
    VERIFIANT debit DE x > 2500

SI ...

```

Il est possible de définir plusieurs variables dans une même facette SOIT et d'imposer certaines contraintes entre ces variables.

```

SOIT x UN <ensemble d'objets>
    ET y UN <ensemble d'objets>

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

VERIFIANT <conditions de restrictions de l'ensemble>

```

SOIT x UN montant
ET y UN montant
VERIFIANT x <> y
SI ...

```

Une autre syntaxe, équivalente, est possible : l'utilisation de plusieurs facettes **SOIT** séparées par des virgules. La règle qui précède peut aussi s'écrire :

```

SOIT x UN montant,
SOIT y UN montant
VERIFIANT x <> y
SI ...

```

Dans ce cas-là, Figaro instancie $\text{card}(\text{montant}) * \text{card}(\text{montant})$ fois la règle d'interaction, où $\text{card}(\text{montant})$ représente le nombre d'objets de l'ensemble *montant*.

*Remarque : Etant donné que la portée des variables déclarées dans le **SOIT** s'étend sur l'ensemble de la règle, si d'autres variables de même nom sont utilisées dans d'autres facettes par des opérateurs complexes (**QQSOIT**, **IL_EXISTE**, **IL_EXISTE AU MOINS**, **SOMME**, **PRODUIT**, **POUR_TOUT**), nous rappelons que ce sont les variables de plus petite portée (donc celle des opérateurs complexes) qui seront d'abord prises en compte. L'exemple suivant montre un cas où une surcharge de variable liée est effectuée. Même si c'est clair pour les outils Figaro, d'un point de vue lisibilité par des humains, il vaut mieux prendre des noms de variables différents.*

Exemple :

```

TYPE pompe ;
INTERFACE
    montant GENRE vanne ;
INTERACTION
    SOIT x UN montant
    SI IL_EXISTE x UN montant VERIFIANT debit DE x > 2500
        (* il s'agit du x de IL_EXISTE *)
    ALORS debit DE x <-- 3000 ;
        (* il s'agit du x du SOIT *)

```

Pour finir, voici un exemple typique d'utilisation de **SOIT**, combiné avec l'identificateur d'objet **MOI_MEME**. Imaginons que dans une base de connaissances, on ait les types composant et equipe_reparation. Pour éviter des erreurs de saisie, on demande à l'utilisateur de remplir uniquement l'interface cpt (liste des composants gérés) pour chaque équipe de réparation. Donc, dans le type composant, on n'a pas directement accès par une interface à l'équipe de réparation qui gère le composant. Malgré tout on veut exprimer que lors d'une panne, le composant va « consommer » un réparateur dans l'équipe qui le gère. Voici comment on peut l'écrire en Figaro

```

TYPE composant ;
ATTRIBUT etat DOMAINE 'marche' 'attente_rep' 'en_reparation'
    PAR_DEFAULT 'marche';

INTERACTION
    SOIT x UN OBJET DE_TYPE equipe_rep VERIFIANT MOI_MEME INCLUS_DANS cpt(x)
    SI etat = 'attente_rep' ET nb_rep_dispo(x) > 0
    ALORS etat <-- 'en_reparation',
        nb_rep_dispo(x) <-- nb_rep_dispo(x) - 1;

TYPE equipe_rep;
ATTRIBUT nb_rep_dispo DOMAINE ENTIER PAR_DEFAULT 1 ;
INTERFACE cpt GENRE composant ;

```

On aurait aussi pu écrire :

```

TYPE composant ;

```


EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

ATTRIBUT etat DOMAINE 'marche' 'attente_rep' 'en_reparation'
PAR_DEFAULT 'marche';

TYPE equipe_rep;
ATTRIBUT nb_rep_dispo DOMAINE ENTIER PAR_DEFAULT 1 ;
INTERFACE cpt GENRE composant ;
INTERACTION
  SOIT x UN cpt
  SI etat(x) = 'attente_rep' ET nb_rep_dispo >0
  ALORS etat(x) <-- 'en_reparation',
  nb_rep_dispo <-- nb_rep_dispo - 1 ;

```

Cette deuxième version est plus simple, mais d'un autre côté, il paraît plus logique d'avoir dans le type composant la règle qui déclenche le début de sa réparation, plutôt que dans le type `equipe_rep`.

7. Notions avancées

Nous allons étudier dans cette section les particularités de Figaro qui sont d'approche un peu plus complexe et qui sont moins usitées lors de l'écriture d'une première base de connaissances.

7.1. La pré-compilation

Les outils Figaro, en préalable à tout traitement sur une base de connaissances, lui appliquent l'opérateur de pré-compilation standard des compilateurs qui reconnaît des mots-clés commençant par "#". Les paragraphes suivants expliquent les différents usages de la précompilation dans le contexte Figaro :

- Inclusion de fichiers par `#include`
- Définition de macros par `#define`
- Définition de variantes de compilation (`#define`, `#ifdef`, `#ifndef`, `#else`, `#endif`, `#undef`)

7.1.1. Inclusion de fichiers

La directive :

```
#include "nomfichier"
```

recherche le fichier *nomfichier* dans le répertoire du fichier Figaro principal (*nomfichier* peut être un chemin relatif par rapport à ce répertoire) et insère le contenu de *nomfichier* à la place de la directive. Le découpage en plusieurs fichiers permet par exemple de séparer la définition de données de fiabilité par défaut (par définition de sous-types ne contenant que ces données) du reste de la base de connaissances, plus stable.

7.1.2. Définition de macros

Il est possible de définir des « macros » introduites par le mot-clé `#define`. Cela permet de gagner beaucoup de temps dans l'écriture de bases de connaissances contenant des structures répétitives telles que la déclaration de nombreuses défaillances. Une macro peut faire appel à d'autres macros, comme dans l'exemple suivant où l'on commence par définir la concaténation de deux arguments, avant de l'utiliser dans la déclaration d'une panne.

Exemple :

```

(* macro de concaténation de deux chaines de caractères *)
#define colle2(debut,fin) debut/**/fin

(* macro définissant une déf. en fct, utilisant la macro colle2 *)
#define DEFAILLANCE_FONC(nomdef,comment,param,valeur) \
  CONSTANCE param DOMAINE REEL EDITION NON VISIBLE ROLE CONCEPTION \
  PAR_DEFAULT valeur; \
  ATTRIBUT colle2(__inhibate__,nomdef) DOMAINE BOOLEEN EDITION VISIBLE, \
  MODIFIABLE PAR_DEFAULT FAUX; \

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

PANNE nomdef LIBELLE comment EDITION VISIBLE, MODIFIABLE \
    PAR_DEFAULT FAUX \
DONNEES_FIABILISTES \
    GROUPE Groupe_add \
    SI ((visu DE GLOBAL_OBJ = 'inconnu') ET (NON \
        colle2(__inhibate__,nomdef))) \
    MODELE_PAR_DEFAULT MODELE_GLM \
    MODELE_GLM GAMMA 0 LAMBDA param MU 0 MODELE_FIGE;

```

Les caractères “\” en fin de ligne servent à dire que la macro se prolonge à la ligne suivante. Ces caractères doivent être suivis **immédiatement** du caractère fin de ligne.

Exemple d'appel à la macro précédente :

```
DEFAILLANCE_FONC(DF,"perte du fluide dans %OBJET",l_df,1e-4)
```

Préalablement à tout traitement à partir de la base de connaissances, le préprocesseur sera appelé et « dépliera » la ligne d'appel à la macro en la remplaçant par le texte de la macro, lui-même modifié par le remplacement de la chaîne

- *nomdef* par DF,
- *comment* par "perte du fluide dans %OBJET"
- *param* par l_df
- *valeur* par 1e-4

Ce processus est récursif puisqu'une macro peut contenir l'appel à une autre macro.

Outre le fait de fournir des raccourcis pour des structures répétitives, les macros peuvent servir à simuler la définition de fonctions simples.

Exemple

```
#define surface(rayon)\
3.1416*rayon**2
```

7.1.3. Définition de variantes d'une base de connaissances

Enfin, grâce aux constantes de précompilation, il est possible de définir des variantes d'une base de connaissances. Dans l'exemple suivant, on choisit de rendre l'attribut `calcul` visible ou non selon la valeur de la constante de précompilation `__DEBUG__`.

Exemple :

```

#ifdef __DEBUG__
    #define __VISUALISATION__ VISIBLE
#else
    #define __VISUALISATION__ NON_VISIBLE
#endif

TYPE t ;
ATTRIBUT calcul EDITION __VISUALISATION__;

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Normalement, c'est au moment de charger la base de connaissances dans KB3 qu'on doit choisir la valeur à donner aux constantes de précompilation car cela va modifier le comportement des modèles saisis avec cette base de connaissances. KB3 lit ces valeurs dans le fichier de paramétrage de l'interface graphique, d'extension .bdc. Mais on peut aussi définir et détruire les constantes de précompilation directement dans le fichier de la base de connaissances par des instructions du type :

```
#define __DEBUG__
#undef __DEBUG__
```

7.2. Eléments globaux

Pour modéliser une classe de systèmes, on a parfois besoin de définir des règles ou des variables attachées à la base de connaissances elle-même, donc au système complet que saisira l'utilisateur, et non à un composant particulier de ce système.

Pour permettre la définition de tels éléments, le langage Figaro permet de déclarer des objets particuliers, les OBJET_SYSTÈME. Ces objets existent dans tout système décrit avec cette base de connaissances et leurs variables sont accessibles depuis tout autre objet du modèle (ils ne contiennent aucune déclaration d'interface). Les OBJET_SYSTÈME sont les seuls que l'on a le droit de déclarer dans une base de connaissances.

Dans l'exemple suivant, la variable nb_def(Compteur_defaillances) est directement accessible depuis n'importe quel type de la base de connaissances.

```
TYPE compteur_def ;
ATTRIBUT
  nb_def DOMAINE ENTIER PAR_DEFAULT 0;
INTERACTION
  ALORS nb_def <-- SOMME POUR_TOUT x UN OBJET DE_TYPE composant
                  DES_TERMES (1 * PANNE(x));

OBJET_SYSTÈME Compteur_defaillances EST_UN compteur_def ;
```

7.3. Héritage et surcharge

Les règles de gestion des mécanismes de surcharge et d'héritage mises en œuvre dans Figaro sont complètement explicitées en référence /2/. Dans ce manuel, nous nous contenterons d'exposer les principes de ces mécanismes et les interdits associés.

Le mécanisme de surcharge est mis en œuvre avec les deux principes suivants :

- La surcharge (redéfinition) d'une caractéristique (ou d'une partie seulement) s'effectue sur son nom.
- Le "dernier qui parle a raison" : un type peut redéfinir localement les caractéristiques du ou des types dont il hérite. La surcharge peut également s'effectuer entre types hérités. Dans ce cas, c'est l'ordre d'apparition des types pères dans la facette SORTE_DE qui détermine l'ordre de prise en compte des caractéristiques et de leurs facettes.

Voici les principaux interdits associés à ce mécanisme :

- Il est interdit de modifier dans un type la nature (attribut, constante, paramètre loi, effet, panne) d'un champ défini dans un père.

```
TYPE pompe ;
ATTRIBUT
  a DOMAINE BOOLEEN ;

TYPE pompe_secours
  SORTE_DE pompe ;
CONSTANTE
  a DOMAINE BOOLEEN ; (*INTERDIT*)
```

Le champ a défini initialement comme attribut est redéfini comme constante.

- Il est interdit de surcharger le domaine d'un champ :

```
TYPE pompe ;
ATTRIBUT
  a DOMAINE BOOLEEN ;
```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

TYPE pompe_secours
  SORTE_DE pompe ;
ATTRIBUT
  a DOMAINE REEL ; (*INTERDIT*)

```

Le champ a défini initialement comme booléen est redéfini comme réel.

Il est en revanche autorisé de surcharger un domaine énuméré (de l'étendre, de le modifier, de le restreindre, etc.). La surcharge d'un domaine énuméré comporte les restrictions suivantes :

1. Le domaine doit rester un domaine énuméré.
2. La surcharge d'un domaine énuméré est autorisée à condition qu'au moins un élément du nouveau domaine soit commun aux deux domaines.

- La surcharge d'une interface est autorisée uniquement si le type avec lequel on surcharge est un sous-type du type surchargé.

```

TYPE composant_hydraulique ;
INTERFACE
  secours GENRE composant_hydraulique ;

TYPE pompe SORTE_DE composant_hydraulique;
INTERFACE
  secours GENRE pompe;

```

- Il est interdit de surcharger une règle d'interaction par une règle d'occurrence et réciproquement.

- Il est interdit de surcharger les caractéristiques d'un type dans un objet particulier instance de ce type.

7.4. Les Groupes de règles

En principe, lors de l'exploitation d'un modèle Figaro, toutes les règles d'interaction et d'occurrence appartenant à des objets du système étudié sont analysées. En fait, ce principe de base peut être raffiné par l'introduction de la notion de **groupes de règles**.

Dans une base de connaissances, il est possible de définir des **groupes** particuliers de règles à partir de l'ensemble des règles du modèle. Cette possibilité permet, lors de l'exploitation d'un modèle Figaro, d'effectuer une analyse sélective du modèle en restreignant l'ensemble des règles à analyser à un ou plusieurs groupes de règles.

Prenons par exemple le cas d'une base de connaissances destinée à être utilisée pour générer automatiquement des arbres de défaillances sur des systèmes thermohydrauliques. Dans cette base, les redondances passives entre composants ne sont pas modélisées car, pour permettre l'étude du système par arbre de défaillances, on fait l'approximation que toutes les redondances du système sont actives.

On veut ensuite utiliser cette même base de connaissances pour construire un modèle du type "graphe d'états" permettant de valider cette approximation. Pour ce faire, on va enrichir la base de connaissances initiale des règles modélisant les redondances actives entre composants et créer deux groupes de règles, le groupe "passif" et le groupe "actif".

- Toutes les règles modélisant le système sans tenir compte des redondances passives seront attribuées au groupe "actif".
- Toutes les règles modélisant les reconfigurations de type normal/secours seront attribuées au groupe "passif".

Lorsque la base de connaissances sera utilisée pour générer un modèle du type graphe d'états, on analysera uniquement les règles du groupe "passif".

Les groupes de règles de la base de connaissances sont déclarés en début de base de connaissances dans la section NOM_DES_GROUPES (**l'information recensant les identifiants des groupes utilisés dans la base est une information relative à l'ensemble de la base et non à un type particulier**).

```

NOMS_DES_GROUPES

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```
passif ;
actif ;
```

Il est alors possible de spécifier pour une règle d'occurrence ou pour une règle d'interaction que la règle appartient à un ou plusieurs groupes (énumération de groupes) en utilisant la facette GROUPE .

```
INTERACTION
regle_prop
GROUPE passif actif
SI IL_EXISTE a UN amont TEL_QUE relie DE a
ALORS relie ;
```

L'attribution d'une règle à un groupe particulier n'est pas obligatoire. Pour l'exploitation d'un modèle Figaro, les conventions suivantes ont été adoptées :

- Si aucun nom de groupe n'est spécifié, l'ensemble des règles du modèle est analysé.
- L'utilisateur peut demander l'exploitation d'un modèle Figaro sur l'union des groupes d'une liste particulière.

Ex : analyse des groupes *passif* UNION *actif*

- Les règles n'appartenant à aucun groupe pourront être exploitées spécifiquement en demandant l'analyse du groupe identifié par SANS_NOM.

Ex : analyse des groupes SANS_NOM UNION *actif*

Il est très important de noter que l'organisation des règles d'une base de connaissances en groupes de règles est une étape importante de la conception de base de connaissances. Plus particulièrement, le rôle de chaque groupe de règles (y compris du groupe SANS_NOM) devra être explicité pour les utilisateurs de la base.

Il faut noter que le découpage du modèle Figaro en groupes n'est possible que pour les règles. La définition des champs valués du modèle devra donc être adaptée, quels que soient les groupes de règles analysés. Il arrive que certains champs valués ne soient pas du tout utilisés par certains traitements, étant donné les règles sélectionnées, mais ce n'est pas gênant car cela n'affecte pas les performances du traitement.

Lorsque des groupes ont été définis dans la base de connaissances, ils sont exploitables dans les bases de faits associées à cette base de connaissances. La redéfinition d'un GROUPE dans une base de faits est interdite et provoque un message d'erreur de duplication de nom de groupes. En effet, toutes les données définies dans la base de connaissances sont accessibles dans la base de faits. Il n'est donc pas nécessaire de les redéfinir.

7.5. Les étapes

Comme nous l'avons vu au chapitre 2, l'ordre dans lequel sont appliquées les règles d'interaction d'un modèle Figaro n'est absolument pas défini. Autrement dit, les bases de règles doivent être écrites de manière à ce que, quel que soit l'ordre d'application, le résultat obtenu soit le même.

Cependant, il peut arriver que l'on ait besoin d'introduire un ordre dans l'application des règles d'interaction. C'est pourquoi le langage Figaro permet de **regrouper les règles d'interaction en blocs de règles** appelés ETAPES et **d'ordonner ces étapes** afin que les règles soient examinées dans un ordre précis.

Prenons l'exemple d'une classe de systèmes thermohydrauliques pour laquelle on veut modéliser la propagation du fluide. Cette classe de systèmes contient des objets de type *composant* qui peuvent être ou non des sources de fluide. Pour modéliser la propagation du fluide, il faut modéliser deux règles :

- les objets *source* sont toujours alimentés,
- les objets de type *composant* sont alimentés s'il existe un composant alimenté en interface *amont*.

Pour minimiser le nombre de règles à appliquer, il est intéressant d'appliquer la première de ces deux règles en premier. C'est pourquoi on introduit deux étapes dans la base de connaissances : une étape *initialisation* et une étape *propagation*.

```
ORDRE_DES_ETAPES
initialisation ;
propagation ;
```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

TYPE composant ;
INTERFACE
    montant GENRE composant ;
EFFET
    alimente DOMAINE BOOLEEN ;
CONSTANTE
    source DOMAINE BOOLEEN ;

INTERACTION
    init
    ETAPE initialisation
    SI source = VRAI
    ALORS alimente ;

    prop
    ETAPE propagation
    SI IL_EXISTE x UN montant TEL_QUE alimente(x)
    ALORS alimente ;

```

7.5.1. Déclaration des étapes

Une étape peut avoir un ou plusieurs identifiants déclarés sous forme d'une liste d'identifiants (par exemple les identifiants "init" et "initialisation", séparés par une virgule, identifient la même étape).

Il est aussi possible d'associer une condition à une étape. Une condition associée à une étape Figaro est une expression booléenne complexe ou non. Cette condition sera évaluée lorsque les règles associées à l'étape vont être évaluées :

- Si la condition n'est pas vérifiée, aucune règle de l'étape n'est évaluée (ce qui correspond à une optimisation sensible car l'évaluation des conditions de ces règles n'est pas effectuée).
- Si la condition est vérifiée, les conditions des règles de l'étape sont évaluées. Il s'agit du processus normal de déclenchement des règles.

Remarque : Une étape sans condition est considérée comme une étape dont la condition renvoie toujours VRAI. Les règles de l'étape sont donc toujours évaluées.

Cette spécificité a été ajoutée pour optimiser le déclenchement des règles dans les traitements Figaro. La condition de l'étape correspond, en quelque sorte, à une factorisation de la partie commune des conditions des règles composant cette étape.

Les noms des étapes et leur ordre d'exécution sont déclarés au début de la base de connaissances dans la section **ORDRE_DES_ETAPES** (l'information recensant les identifiants et l'ordre des étapes utilisées dans la base est une information relative à l'ensemble de la base et non à un type particulier).

Les étapes sont ordonnées en fonction de leur ordre de définition dans la base de connaissances. Lorsque le mot clé **ORDRE_DES_ETAPES** apparaît plusieurs fois dans la base de connaissances, on considère que les étapes sont définies dans l'ordre de lecture de la base de connaissances.

Exemple 1 : Déclaration d'étapes sans condition.

```

ORDRE_DES_ETAPES
    initialisation, init;
    propagation;

```

Dans cet exemple, le numéro d'ordre de l'étape initialisation est 1 et le numéro d'ordre de l'étape propagation est 2.

Exemple 2 : Déclaration d'étapes avec condition.

```

ORDRE_DES_ETAPES
    initialisation CONDITION NON initialisation_effectuee(parametres_globaux);
    propagation ;

TYPE parametres ;
ATTRIBUT initialisation_effectuee DOMAINE BOOLEEN PAR_DEFAULT FAUX ;

OBJET_SYSTEME parametres_globaux EST_UN parametres ;

```

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Exemple 3

ORDRE_DES_ETAPES etape1 ; etape2 ;	est équivalent à	ORDRE_DES_ETAPES etape1 ; etape2 ; etape3 ;
ORDRE_DES_ETAPES etape3 ;		

La définition de nouvelles étapes est autorisée dans la base de faits. Dans ce cas, les étapes déclarées dans la base de faits sont concaténées à la suite des étapes de la base de connaissances. Leur ordre est donc toujours supérieur à celui des étapes de la base de connaissances.

La duplication dans la base de faits d'une étape déjà définie dans la base de connaissances est interdite et provoque un message d'erreur de duplication de nom d'étape. En effet, toutes les données définies dans la base de connaissances sont accessibles dans la base de faits. Il n'est donc pas nécessaire de les redéfinir.

7.5.2. Regroupement des règles par étapes

Lorsque des étapes sont déclarées, chaque règle du modèle doit **obligatoirement** être associée à **au moins une étape**. Cette association se fait avec la facette ETAPE des règles d'interaction.

Exemple :

```

INTERACTION
  init
  ETAPE initialisation
  SI source = VRAI
  ALORS alimente <-- VRAI ;

```

Toutefois, pour des raisons de commodité, un automatisme des outils Figaro considère que les règles pour lesquelles l'étape d'appartenance n'est pas précisée sont dans l'étape appelée `etape_par_defaut`.

L'étape `etape_par_defaut` peut être placée où l'on veut dans l'ordre des étapes, il suffit de la déclarer comme une étape définie explicitement.

Une règle d'interaction peut être associée à plus d'une étape : cela signifie simplement qu'à chaque étape déclarée dans la facette ETAPE de cette règle, elle est prise en compte par le mécanisme d'inférence.

7.5.3. Application des règles d'interaction en chaînage avant

L'introduction d'étapes dans un modèle Figaro modifie le traitement d'application des règles d'interaction en chaînage avant. Dans ce cas, les règles sont traitées étape par étape dans l'ordre spécifié par **ORDRE_DES_ETAPES**, et en sautant les étapes dont la condition n'est pas vérifiée. Sauter l'étape k revient à considérer, dans la définition formelle de la sémantique de Figaro 0 de la section 3.2, que la fonction l_k est la fonction identité.

Il est important de noter que l'organisation d'une base de règles d'interaction en étapes permet de structurer la base de règles et d'optimiser les temps d'exploitation d'un modèle.

7.5.4. Application des règles d'interaction en chaînage arrière

A ce jour, les étapes ne sont pas exploitées par le traitement d'exploitation d'un modèle Figaro en chaînage arrière.

7.6. Les équations

Certains problèmes, tels que le calcul des courants et tensions dans un réseau électrique, sont difficiles à exprimer dans un formalisme en règles. Dans le meilleur des cas, on arrive à une résolution itérative par de nombreuses applications des règles, qui finissent par faire converger les attributs vers les valeurs que l'on veut calculer.

Il est, dans un tel cas, beaucoup plus élégant de poser tout simplement le système d'équations que doivent satisfaire les valeurs d'un certain nombre d'attributs, déclarés (le temps de cette résolution) comme inconnues du

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

système d'équations.

7.6.1. Systèmes d'équations

Lorsqu'une base de connaissances nécessite l'écriture d'un ou plusieurs systèmes d'équations, chaque système doit être identifié par un nom. Les noms des systèmes d'équations utilisés dans la base sont déclarés au début de la base de connaissances dans la section NOMS_DES_SYSTEMES (l'information recensant les identifiants des systèmes utilisés dans la base est une information relative à l'ensemble de la base et non à un type particulier).

```
NOMS_DES_SYSTEMES
    courant; tension;
```

7.6.2. Equations

Syntaxe:

```
EQUATION      nom_equation
[ GENRE      LINEAIRE ]
[ SYSTEME_EQUATIONS liste_systemes_equations ]
[ SI          expression ]
[ FORMULE     [ expression = expression ] ; ]
```

Les équations d'un système sont décrites dans les types de la base de connaissances (une équation appartient à un type).

Une équation ne peut appartenir qu'à un unique système d'équations.

Les équations seront définies par le type de champ **EQUATION**.

Détail des facettes :

Nom d'équation (obligatoire)

Permet d'attribuer un nom à l'équation. C'est utile pour "commenter" les équations et pour les repérer dans les outils de recherche de bogues.

Facette SYSTEME_EQUATIONS (obligatoire)

Permet de spécifier les systèmes auxquels appartient l'équation.

Facette GENRE (obligatoire)

Doit être suivi d'un mot clé qui permettra d'orienter vers une méthode de résolution adaptée au problème. Pour l'instant, seul est prévu le genre LINEAIRE, pour lequel des algorithmes efficaces existent.

Facette SI (facultatif)

SI est une facette optionnelle permettant d'exprimer une condition qui doit être satisfaite par l'état du modèle pour que la formule soit prise en compte dans le système d'équations. Par exemple, les équations électriques sont indéterminées pour des composants qui ne sont reliés à aucune source de tension. Il faut donc conditionner la prise en compte des équations relatives à un composant par le fait qu'il est relié à une source, ce qui se fait facilement par des règles d'interaction classiques.

Facette FORMULE (obligatoire)

Cette facette doit être suivie de l'équation elle-même, dans laquelle les inconnues sont précédées d'un point d'interrogation '?'. Ces inconnues sont obligatoirement des attributs déclarés. La valeur de ces attributs sera calculée au moment de la résolution mathématique de la formule. Ainsi, la valeur des attributs sera modifiée. L'équation doit être écrite sous la forme de l'égalité entre un membre gauche dans lequel apparaissent les variables, au sein d'une expression numérique, et un membre droit, qui doit être une constante.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Afin de simplifier les traitements, on impose que les équations de type linéaire soient écrites sous une forme "standard", dans laquelle le membre de gauche se présente sous la forme : $a_1 \cdot x_1 + a_2 \cdot x_2 + \dots + a_n \cdot x_n$, où les a_i sont des expressions numériques ne faisant intervenir aucune variable de l'équation.

Exemple :

```

TYPE    sommet;
ATTRIBUT
    a      DOMAINE REEL;
    b      DOMAINE REEL;
    c      DOMAINE REEL;
    x      DOMAINE REEL;
    y      DOMAINE REEL;
    isole  DOMAINE BOOLEEN;
EQUATION
    equil
    GENRE LINEAIRE
    SYSTEME_EQUATIONS courant
    SI NON isole(s1) (* La formule n'a de sens que dans ce cas *)
    FORMULE a(s1)*x(s1) + b(s1)*y(s1) = c(s1);
              (* x et y sont les inconnues de la formule : elles vont
                 être substituées lors de la résolution de la formule.
                 x et y sont des attributs *)

```

Rappel : la résolution d'un système d'équations est déclenchée au moment voulu, par une règle d'interaction.

Exemple : déclenchement de la résolution pour le système "courant".

```

INTERACTION
    r1
    ETAPE resoudre
    ALORS RESOUDRE_SYSTEME(courant);

```

8. Documentation des modèles Figaro

Outre les commentaires classiques, entre "(" et ")", on peut ajouter à un modèle des commentaires formalisés, annoncés par des mots clés.

Pour une base de connaissances, on peut, en dehors des types, écrire une description, annoncée par le mot clé `DESCRIPTION_BDC`. Il doit y avoir *au plus une* description par base de connaissances. C'est une chaîne de caractères quelconques, qui peut éventuellement être écrite sur plusieurs lignes.

Pour un type, on peut, à l'intérieur du type, écrire une description annoncée par le mot clé `DESCRIPTION`. Il doit y avoir *au plus une* description par type. C'est une chaîne de caractères quelconques, qui peut éventuellement être écrite sur plusieurs lignes.

Enfin, tous les champs déclarés en `INTERFACE`, `CONSTANTE`, `ATTRIBUT`, `PANNE`, `EFFET` peuvent être munis d'une facette `LIBELLE` (unique). Les règles d'héritage et de surcharge s'appliquent à ces champs, y compris pour leurs libellés.

Lors d'une instanciation à l'ordre 0, les descriptions de la base de connaissances et des types disparaissent, alors que les libellés des champs cités ci-dessus sont conservés. A l'intérieur de ces libellés, la chaîne de caractères `%OBJET` est remplacée par le nom de l'objet auquel le libellé appartient.

Exemple de base de connaissances utilisant tous les types de documentation formalisée possibles :

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

DESCRIPTION_BDC "Base de connaissances avec commentaires" ;

TYPE pompe ;

DESCRIPTION

"Pompe générique, à spécialiser dans les sous-types"; (* une description peut être écrite sur plusieurs lignes *)

CONSTANTE debit **DOMAINE REEL**

PAR_DEFAULT 10

LIBELLE "Débit nominal de la pompe %OBJET";

PANNE fuite **LIBELLE** "Fuite interne ou externe" ;

TYPE pompe_electrique **SORTE_DE** pompe;

DESCRIPTION "Pompe mue par l'électricité";

9. Cohérence des modèles

Compte tenu de la complexité de certaines bases de connaissances de KB3 (la plus complexe, celle de l'outil TOPASE dédié aux études de fiabilité des postes THT [8], comporte 27000 lignes de langage Figaro), il semble de plus en plus nécessaire de s'assurer que celles-ci réalisent bien ce que l'auteur avait prévu ou du moins de concevoir un outil qui pourrait **avertir le concepteur que sa base, indépendamment des objectifs qu'elle poursuit, possède des propriétés indésirables, que nous appellerons « incohérences »**. Ces propriétés indésirables peuvent être de plusieurs ordres : contradictions, caractère non borné de valeurs numériques, définitions bouclées (A défini à partir de B et réciproquement) que rien ne permet de résoudre, divisions par zéro, underflow ou overflow dans des calculs sur les nombres réels, etc.

L'objectif de ce chapitre est de décrire les principes de méthodes permettant soit d'assurer **par construction** la cohérence des bases de connaissances, soit de détecter l'existence d'incohérences. La démonstration de la cohérence, quant à elle, est impossible dans le cas général, c'est à dire pour une base de connaissances dont la construction n'a pas été menée suivant les méthodes que nous préconisons. Cela n'a rien d'étonnant, dans la mesure où le langage Figaro combine toutes les possibilités de la logique des propositions, du calcul sur les nombres réels, et des automates.

A défaut de pouvoir prouver la cohérence au niveau d'une base de connaissances, il est quelquefois possible de traiter les modèles de systèmes au cas par cas. **Cela permet d'envisager l'utilisation de bases de connaissances imparfaites**, en vérifiant pour chaque système étudié que les modèles construits ne tombent pas dans les failles de la base. Un exemple typique est celui des bases de connaissances qui n'assurent la cohérence des modèles qu'à condition que la topologie des systèmes étudiés ne soit pas bouclée.

Ce chapitre est organisé comme suit : nous commençons par explorer les relations entre cohérence à l'ordre 1 et cohérence à l'ordre 0. Les relations établies nous permettent de limiter le travail de démonstration à l'ordre 0. Puis en nous appuyant sur la définition formelle de la sémantique du langage Figaro d'ordre 0 nous faisons une typologie des incohérences qu'un modèle en Figaro 0 est susceptible de contenir. Nous donnons ensuite un ensemble de règles de construction d'une base de connaissances permettant d'éviter les incohérences identifiées. Pour la plus délicate de ces règles : l'organisation des "étapes" du modèle, nous présentons les principes d'un outil qui aide l'utilisateur à faire ce travail. Pour finir, nous nous intéressons au cas des démonstrations de cohérence faisables uniquement à l'ordre 0, c'est à dire au cas par cas sur les modèles de systèmes saisis par l'utilisateur.

9.1. Relation entre incohérence à l'ordre 1 et à l'ordre 0

Démontrer (pour autant que cela soit possible) l'absence d'incohérences dans une base de connaissances revient à démontrer que :

Quel que soit le système (respectant les contraintes décrites dans la base de connaissances) saisi par l'utilisateur, le modèle en langage Figaro d'ordre 0 généré à partir de ce système et la base de connaissances ne contiendra pas d'incohérence.

Les incohérences peuvent apparaître soit dans la phase d'instanciation (transformation du modèle d'ordre 1 en modèle d'ordre 0), soit dans la phase d'exploitation du modèle (quand on l'utilise pour simuler le système modélisé). Les problèmes éventuels d'instanciation peuvent être détectés de manière **exhaustive** par les contrôles syntaxiques que nous décrivons ci-après, effectués **sur la base de connaissances**.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

9.1.1. Contrôles syntaxiques

Une base de connaissances est une collection de **types** d'objets décrits en langage Figaro **d'ordre 1** (c'est à dire permettant de décrire des connaissances génériques, grâce notamment à des opérateurs adaptés à la manipulation d'**ensembles**), que l'utilisateur de la base pourra assembler d'une infinité de manières différentes au sein de modèles de systèmes.

Cette liberté est malgré tout contrainte par les indications données dans les facettes GENRE et CARDINAL des interfaces déclarées dans la base de connaissances (ex : en amont d'un composant électrique, on ne peut avoir que des composants électriques, dont le nombre est compris entre 0 et N). Grâce à cette information, la vérification syntaxique de la base de connaissances réalisée par KB3 permet d'assurer qu'aucun problème du type tentative d'accès à une variable inexistante (ex : la pression d'un composant électrique) ne peut être révélé lors de l'instanciation d'un modèle. Cette vérification syntaxique est très complexe car elle doit tenir compte de toutes les règles d'héritage entre types, et son utilité est quotidiennement vérifiée par les concepteurs de bases de connaissances.

Avant toute exploitation d'un modèle saisi par l'utilisateur, KB3 vérifie que toutes les contraintes qui ont été définies par le concepteur de base de connaissances sont respectées.

Cette vérification peut déclencher des messages d'erreur du type : "pas assez (trop) d'objets dans l'interface X de l'objet Y".

Le respect des types d'objets mis en interface d'un objet donné est assuré par l'interface de saisie, qui ne permet pas d'erreur dans ce domaine.

Ces contrôles assurent que l'opération d'instanciation à l'ordre 0 dans le cadre d'un système particulier, préalable à tout traitement fiable, se passera sans problème.

Mais bien sûr, cela ne suffit pas : cette vérification ne protège en aucune manière contre les types d'incohérences décrits dans le préambule de ce chapitre.

9.1.2. Cohérence du comportement

La définition précise de ce que l'on entend par incohérence de comportement nécessite au préalable une définition formelle de la sémantique du modèle spécifiant ce comportement.

C'est en nous appuyant sur la définition de la sémantique du langage Figaro 0 (cf. chapitre 3) que nous montrerons où les incohérences peuvent se manifester. Il sera alors possible d'énoncer des règles de construction d'une base de connaissances qui assureront la cohérence de tout modèle en langage Figaro d'ordre 0 produit à partir de cette base.

9.2. Typologie des incohérences possibles

A partir de la formalisation du fonctionnement de "l'automate Figaro 0" (nous réutiliserons ici les notations du chapitre 3), il est possible de définir ce que l'on entend par incohérence d'un modèle. Nous définissons également des propriétés souhaitables.

Le langage Figaro d'ordre 0 peut être considéré comme une sorte de synthèse des concepts existants dans les langages d'intelligence artificielle en règles de production, et dans les réseaux de Petri stochastiques. Il est donc naturel de s'inspirer des travaux faits dans ces deux domaines pour mettre au point des méthodes de contrôle de la cohérence en langage Figaro.

Pour les réseaux de Petri, les propriétés qu'il est éventuellement possible d'établir à partir d'un réseau sont : la possibilité de revenir à l'état initial depuis tout état, le caractère borné des marquages, la vivacité des transitions (une transition donnée T est vivante si à partir de tout état atteignable à partir de l'état initial, il est toujours possible de trouver une séquence de franchissements de transitions incluant le franchissement de T), l'absence d'état absorbant, l'existence d'invariants (combinaisons linéaires des marquages de places qui sont constantes, quelles que soient les évolutions du réseau) [10],[11].

Ces propriétés des réseaux de Petri sont très faciles à transposer dans le contexte Figaro, à l'exception de l'existence d'invariants. Elles reviennent aux propriétés suivantes :

- P1 : caractère fini de l'espace des états,
- P2 : vivacité des transitions (avec la même définition que pour les RdP),
- P3 : non existence d'états absorbants,
- P4 : possibilité de revenir à l'état initial à partir de n'importe quel état.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

On peut d'ores et déjà remarquer que l'absence de valeurs numériques dans V suffit à assurer que l'ensemble des états réellement atteignables par le modèle, qui est inclus dans \mathcal{E} , est **fini** (le fait d'avoir des variables **déduites** numériques ne remet pas en cause cette propriété).

Les deux dernières propriétés sont évidemment à rechercher uniquement lorsqu'on cherche à modéliser un système réparable, par exemple en vue de calculer sa disponibilité asymptotique.

En intelligence artificielle, on retient généralement trois critères pour valider une base de règles, quel que soit son domaine :

- la cohérence au sens de la logique : on ne peut avoir une chose et son contraire simultanément,
- la complétude : il existe une solution au problème quelles que soient les données initiales,
- la pertinence : la base est conforme à la réalité physique (ce point n'est pas traité ici).

La transposition de la notion de cohérence et de complétude à un modèle Figaro 0 demande un peu d'interprétation. En fait, dans le fonctionnement de l'automate, on peut avoir les comportements indésirables suivants :

- **Inc1** : Impossibilité de calculer $I(V_0, Y_0)$, ou bien $I(X)$, X étant l'état obtenu par application d'un ou plusieurs groupes de transitions à partir d'un état précédent. Il y a aussi le cas où le calcul est possible, mais où son résultat dépend de l'ordre des règles. Ces problèmes de complétude et de cohérence d'une base de règles ont été étudiés en 1985 à EDF. Les règles de construction sûre de bases de connaissances données dans cet article sont déduites des théorèmes démontrés dans [5].

- **Inc2** : Incohérence liée à des transitions applicables **au même instant** à partir d'un état *instantané* X donné. La signification "physique" de cette situation est le fait que plusieurs actions de durée nulle (mais au résultat aléatoire) sont lancées en parallèle et doivent donc produire leurs effets exactement en même temps. Typiquement, il peut s'agir de demandes de démarrages simultanées sur plusieurs composants. Grâce à la notion de groupes de transitions, il est inutile d'explorer tous les ordres possibles de déclenchement des transitions, mais cela suppose en général que les groupes de transitions soient indépendants les uns des autres. Indépendance signifie ici deux choses :

- l'application d'une transition ne remet pas en cause la condition de déclenchement d'une **autre** transition (en revanche, il est normal qu'elle rende fausse sa propre condition),
- quel que soit l'ordre d'application des transitions, pour une combinaison donnée de transitions prises chacune dans un groupe, on a le même résultat pour $t_1 \circ t_2 \circ \dots \circ t_k(X)$.

L'une ou l'autre de ces conditions peut éventuellement ne pas être respectée pour des raisons bien particulières dans certains modèles, mais cela doit être un choix délibéré de l'utilisateur. C'est pourquoi tout outil exploitant le langage Figaro doit signaler à l'utilisateur le non-respect d'une de ces conditions pour l'alerter sur une éventuelle faute d'étourderie.

La notion de groupe de transitions est d'un grand secours dans la modélisation parce qu'elle permet de traiter d'une manière très élégante le genre de situation suivant : dans un système ayant n composants indépendants qui doivent démarrer au même instant, il est inutile d'explorer les $n!$ séquences qui mènent au même résultat (c.-à-d. une combinaison donnée de succès et de pannes parmi les 2^n possibilités existantes). Au contraire, l'automate Figaro 0 peut produire directement ces 2^n sorties résultats de l'état instantané initial.

- **Inc3** : A ces incohérences locales peut s'ajouter un comportement indésirable plus global qui est l'enchaînement infini d'une succession de transitions instantanées, possible car cette succession de transitions ramène à un état à partir duquel elle peut à nouveau se déclencher. Il n'existe pas pour l'instant (à notre connaissance) de méthode permettant de prévoir ce type de comportement.

Il est sans doute utile de compléter les remarques ci-dessus par l'observation du fait qu'il serait **absurde** de tenter de démontrer des propriétés de cohérence amalgamant des règles d'interaction et d'occurrence. En effet, pour un système réparable, il est bien évidemment tout à fait souhaitable qu'il existe des règles qui mettent une panne à VRAI, et que dès que cela a été fait, d'autres règles permettent de mettre la même panne à FAUX. Simplement, il ne faut pas que ces opérations puissent toutes se faire en un temps nul, ce qui amènerait une situation du type décrit à l'alinéa précédent.

Nous allons maintenant donner un certain nombre de méthodes d'écriture des bases de connaissances qui permettent d'éviter **par construction** les incohérences Inc1 à Inc3, et/ou d'assurer les propriétés P1 à P4 évoquées plus haut.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

9.3. Graphe de dépendances entre variables

Certaines méthodes que nous allons énoncer s'appuient sur le concept simple de graphe de dépendances entre variables. Cette notion étant importante et ayant des applications intéressantes en dehors même du contrôle de cohérence, nous en faisons une section à part entière de cet article.

Dans un modèle en Figaro 0, on dit que la variable v_1 "influence directement la variable v_2 " dans l'une des deux situations suivantes :

- il existe une instruction d'affectation (\leftarrow) où v_2 apparaît en partie gauche et v_1 en partie droite ($v_2 \leftarrow v_1$ par ex.),
- il existe une règle d'interaction ou d'occurrence dans laquelle v_1 apparaît en prémisse, et v_2 dans la partie gauche de la conclusion.

La meilleure façon de représenter un ensemble de relations d'influence entre des variables est d'utiliser un graphe orienté. En effet, dans ce graphe, l'existence d'un chemin entre deux nœuds associés à des variables représente une influence indirecte.

Lorsqu'on considère les variables d'un point de vue probabiliste, ces variables s'identifient à des processus stochastiques à temps continu ; on peut alors parler de dépendance stochastique entre ces processus. On peut démontrer que l'indépendance stochastique entre deux variables-processus est assurée dès lors qu'il n'existe aucun chemin entre ces deux variables dans le graphe que nous venons de définir.

Cette propriété est extrêmement utile, car elle permet de décomposer le modèle global en sous-modèles pouvant être résolus indépendamment. D'un point de vue qualitatif (pour faire du model-checking, par exemple), elle permet de remplacer la construction d'un graphe d'états global du modèle par une série de graphes indépendants beaucoup plus petits, et d'un point de vue quantitatif elle permet de simplifier le calcul de la probabilité d'être dans un état donné pour une variable d'état, en ne considérant que le sous-modèle contenant l'ensemble des variables qui influencent la variable considérée.

Ainsi donc, dans le cas où l'on souhaite calculer les probabilités d'être dans un état donné pour une variable d'état ou d'être dans différents états pour un **groupe** de variables, on peut commencer par **restreindre le modèle à l'ensemble des variables qui influencent ce groupe de variables**. Le sous-graphe qui ne contient que les variables qui influencent directement ou indirectement le groupe de variables considéré permet de définir le sous-modèle minimum à résoudre pour atteindre cet objectif.

En pratique, l'extraction d'un tel sous-modèle revient à éliminer dans le modèle global toute expression contenant une référence à une variable devant disparaître. Ainsi, les règles :

SI v_1 OU v_2 ALORS v_3 SINON v_4 ;

SI (v_4 ET v_5) OU NON v_3 ALORS v_6 ;

deviendront : SI NON (v_1 OU v_2) ALORS v_4 ; dans le cas où on ne s'intéresse qu'à v_4 .

Par ailleurs, un autre type de sous-système est intéressant à distinguer : les **composantes fortement connexes du graphe** (cfc), car elles constituent des sous-modèles indivisibles pour la résolution. En effet, toutes les variables d'une cfc sont interdépendantes.

On construit alors un "super graphe" dont les nœuds correspondent chacun à une cfc. Ce super graphe est forcément sans circuit (s'il en contenait un, il existerait une cfc plus grande que celles qui ont été identifiées). Ultérieurement une technique de résolution partant des sources du super-graphe et progressant vers la cfc intéressante pourra être utilisée.

9.4. Méthodes sûres d'écriture de bases de connaissances

9.4.1. Avoir un graphe de dépendances pyramidal

Une méthode très sûre pour construire une base de connaissances est de faire en sorte de donner une structure pyramidale au graphe de dépendances entre variables, en ayant un "supergraphe" dont les cfc racines contiennent la totalité des variables essentielles. L'idéal est d'avoir le plus petit nombre possible de variables dans chaque cfc racine.

L'exemple de base de connaissances permettant de décrire un réseau de télécommunications donné ci-après est typique.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

```

(* Bdc pour modéliser des réseaux de télécommunication *)
TYPE
    composant ;
PANNE
    indisponibilite ;
PARAMETRE_LOI
    gamma_indispo PAR_DEFAULT 0.00005 ;
EFFET
    relie LIBELLE "%OBJET connecté à une source" ;
ATTRIBUT indispo_testee DOMAINE BOOLEEN
    PAR_DEFAULT FAUX ;
OCCURRENCE
    SI NON indispo_testee
    IL PEUT SE PRODUIRE
        DEFAILLANCE indisponibilite
        LOI INS ( gamma_indispo )
        PROVOQUE indispo_testee <-- VRAI
    OU BIEN TRANSITION non_indisponibilite
        PROVOQUE indispo_testee<-- VRAI;
(*-----*)
TYPE noeud
    SORTE_DE composant ;
CONSTANTE
    fonction DOMAINE 'source' 'but' 'intermediaire'
        PAR_DEFAULT 'intermediaire' ;
INTERACTION
    SI MARCHE ET fonction ='source' ALORS relie ;
(*-----*)
TYPE arete_uni_dir
    SORTE_DE composant ;
INTERFACE
    depart GENRE noeud CARDINAL 1 ;
    arrivee GENRE noeud CARDINAL 1 ;
INTERACTION
    SI MARCHE ET relie DE depart ALORS relie DE arrivee;
(*-----*)
TYPE arete_bi_dir
    SORTE_DE composant ;
INTERFACE
    extremite GENRE noeud CARDINAL 2 ;
INTERACTION
    SI MARCHE
        ET ( IL EXISTE x UNE extremite TELLE_QUE relie DE x )
        ET ( QQSOIT x UNE extremite ON_A MARCHE DE x )
        ALORS POUR_TOUT y UNE extremite FAIRE relie DE y

```

Voici un réseau simple à trois nœuds (pour simplifier, on suppose les arêtes sans défaillance) suivi du graphe des dépendances entre variables pour ce système, dans lequel les variables faisant partie de la même cfc sont incluses dans un rectangle :

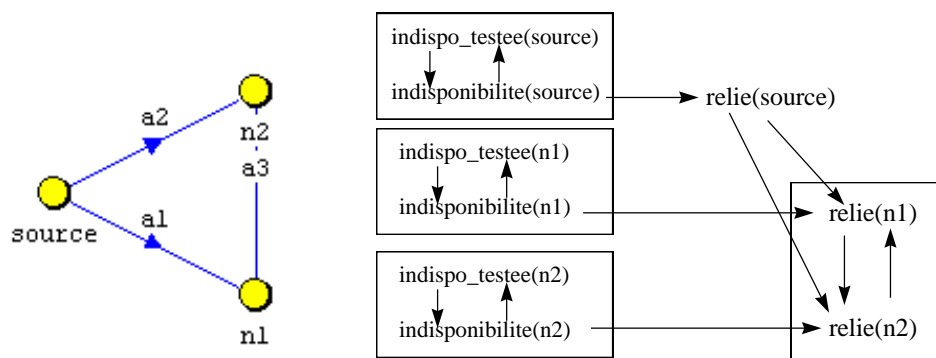


Figure 6. Un réseau et le graphe de dépendances correspondant

La majorité des bases développées à ce jour génèrent des graphes de dépendances pyramidaux. En particulier, c'est le cas des bases dont l'objectif est de produire des arbres de défaillances. La plupart des variables essentielles y sont les pannes, et les pannes d'objets différents sont indépendantes entre elles.

Une façon simple et naturelle d'avoir un graphe pyramidal est de ne créer de dépendances entre variables

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

essentielles que de manière interne aux types (donc aux objets après instanciation sur un système particulier) et de ne créer aucun lien de dépendance dans le sens variables déduites vers variables essentielles.

En donnant ce type d'architecture à une base de connaissances, on est capable de démontrer l'absence d'incohérences de types Inc2 et Inc3, ainsi que les propriétés P1 à P4 (sous réserve qu'on n'ait pas d'incohérence de type Inc1) par des raisonnements très locaux, portant sur quelques règles d'occurrence appartenant à un même type.

Ces démonstrations peuvent être faites au niveau de la base de connaissances, donc **pour tout système qui sera modélisé avec cette base**.

9.4.2. Raisonner en inférence monotone

L'objectif de la règle de construction que nous allons donner maintenant est d'éviter les incohérences de type Inc1.

Dans la référence [9] Hery et Laleuf ont démontré un théorème donnant une condition suffisante de convergence commutative d'une inférence conduite comme celle d'une étape des règles d'interaction en langage Figaro 0. Une conséquence immédiate de ce théorème est le **corollaire** suivant :

Si

- toutes les actions de règles sont des mises à VRAI de variables booléennes initialisées à FAUX avant l'inférence (en général, ce sont des EFFETs),
- toutes les conditions portant sur cet ensemble de variables testent si elles sont à VRAI,

Alors la convergence commutative de l'inférence est assurée.

Ce cadre très simple, mais employé dans la grande majorité des bases de connaissances dont l'objectif est de produire des arbres de défaillances est celui dit de "**l'inférence monotone**". Ce nom vient du fait que lorsqu'un booléen a été mis à VRAI, aucune règle ne permet de revenir sur cette conclusion. La deuxième condition du théorème est là pour assurer qu'aucun booléen ne sera mis à VRAI intempestivement, avant stabilisation de ceux dont il dépend.

L'énorme intérêt de ce théorème est qu'il est extrêmement facile de vérifier (même sans outil) ses hypothèses au niveau de la base de connaissances, ainsi qu'on peut le voir sur l'exemple de modélisation de réseaux de télécommunications donné ci-dessus. Il est donc possible de prouver **au niveau de la base de connaissances que quel que soit l'assemblage de types réalisé par un utilisateur de cette base, aucune incohérence de type Inc1 ne pourra être créée**. Cette propriété très forte a été exploitée dans toutes les bases de connaissances opérationnelles utilisées à EDF pour les études de sûreté des centrales nucléaires.

Grâce à l'inférence monotone, il est possible de modéliser **sans risque d'incohérence et d'une manière très concise et naturelle les propagations de flux dans des systèmes bouclés**, ainsi qu'on peut le voir dans l'exemple ci-dessus du réseau de télécommunication. Or, dans la grande majorité des formalismes classiques (arbres de défaillances, logique booléenne, réseaux de Petri, formalismes de Model Checking...), ce problème de modélisation conduit soit à des impossibilités, soit à des expressions très lourdes et peu lisibles, alors qu'on le rencontre constamment dans l'étude des systèmes réels.

9.4.3. Bien exploiter les étapes

Pour éviter les incohérences de type Inc1 dans des situations où l'inférence monotone est un cadre trop restrictif, on peut utiliser les étapes.

Le théorème sur la convergence commutative de [9], plus général que l'application restreinte que nous en avons faite au paragraphe précédent, montre l'importance d'avoir au niveau d'une inférence un maximum de variables d'état dites "non réceptrices", autrement dit qui ne sont modifiées par aucune action des règles. Un moyen efficace pour diminuer le nombre de variables réceptrices est bien sûr de diminuer le nombre de règles à prendre en considération. C'est précisément ce que permet le découpage en étapes. Une fois que l'inférence correspondant à une étape donnée a été terminée, les variables qui étaient réceptrices pour cette étape peuvent (éventuellement) devenir non réceptrices pour les étapes suivantes.

Voici un exemple très simple d'application des étapes. Supposons que l'on ait besoin d'écrire une règle du type "SI NON effet1 ALORS effet2". A cause de la négation dans la condition, on sort du cadre de l'inférence monotone, **sauf** si l'on met les règles agissant sur effet1 dans une étape précédant l'étape dans laquelle on met la règle ci-dessus.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

9.5. Séquencement automatique des règles d'interaction

Lors de l'écriture des règles d'interaction, il peut s'avérer difficile d'établir de manière systématique une organisation en étapes de ces règles qui assure le bon fonctionnement de l'ensemble du modèle. En général, le concepteur de base de connaissances peut établir facilement des successions locales de règles ou de groupes de règles, mais il est plus délicat de gérer l'ensemble des règles et encore plus d'ajouter une nouvelle règle dans une base de connaissances existante.

Pour diminuer ces difficultés, un outil de séquencement automatique a été mis au point. Cet outil suppose qu'une règle ne doit être utilisée que lorsque ses prémisses sont stabilisées. Dans ce cadre, l'ordre d'exécution des règles peut être directement déduit du graphe des dépendances entre règles, un concept analogue à celui de graphe de dépendances entre variables, que nous introduisons ci-après.

9.5.1. Définition des interdépendances entre règles

On définit qu'une règle R2 dépend d'une règle R1 s'il existe une variable V, modifiée par R1, qui intervient dans les prémisses ou dans les termes de calcul de R2.

Dans le cas d'un modèle Figaro 0, cette définition n'offre aucune difficulté, mais il est bien plus intéressant de pouvoir raisonner au niveau de la base de connaissances. La définition a donc été adaptée aux bases de connaissances en tenant compte des possibilités d'héritage entre types.

On considère donc comme distincts les éléments des différents types, même s'ils sont construits par simple héritage du type père : la variable V du type T est considérée comme différente de la variable V d'un des fils de T. On définit alors qu'une règle R2 dépend d'une règle R1 par l'intermédiaire de la variable V du type T dans le cas suivant :

- la variable V d'un type T1 père de T est modifiée par R1,
- la variable V d'un type T2 père de T intervient dans les prémisses ou dans les termes de calcul de R2.

Les types T, T1 et T2 peuvent être confondus.

9.5.2. Ordonnancement des règles

Une fois les interdépendances premières construites, il faut compléter les dépendances de manière à construire une relation d'ordre partiel ; c'est-à-dire une relation transitive et antisymétrique.

La transitivité est obtenue en ajoutant les dépendances nécessaires : si R3 dépend de R2 qui dépend de R1, on ajoute une dépendance entre R1 et R3.

Lorsque les relations d'interdépendance ajoutées assurent la transitivité de l'ensemble, il faut assurer l'antisymétrie : si R1 dépend de R2 et R2 dépend de R1 alors $R1 = R2$. Il est évident que l'on ne peut pas fusionner simplement les règles, l'antisymétrie est alors obtenue en regroupant les règles au sein d'étapes. La relation d'ordre n'est plus exprimée entre des règles mais entre des étapes : une étape E1 dépend d'une autre étape E2 si une des règles de E1 dépend d'une règle de E2. Le regroupement des règles est poursuivi jusqu'à ce que la relation entre étapes soit antisymétrique.

Le regroupement des règles violant la contrainte d'antisymétrie au sein d'une étape s'explique par le fait que ces règles dépendent les unes des autres de manière bouclée. Il est donc nécessaire de regrouper ces règles et de les exécuter ensemble jusqu'à convergence du résultat. Dans le cas extrême d'un ensemble de règles entièrement interdépendantes, l'algorithme conduit à la création d'une seule étape contenant toutes les règles.

Une fois la relation d'ordre partiel entre étapes établie, elle peut directement être utilisée pour ordonner l'exécution des étapes ; on commence par les étapes indépendantes puis on exécute les étapes ne dépendant que de celles déjà prises en compte et ainsi de suite jusqu'à épuisement des étapes.

9.5.3. Utilisation de l'outil de séquencement des règles

L'outil de séquencement peut être utilisé soit lors de la création d'une nouvelle base de connaissances, soit lors de la modification d'une base existante.

Pour une nouvelle base de connaissances, l'algorithme définit les groupes de règles interdépendantes et permet ainsi de contrôler la pertinence de ces groupes vis à vis du modèle physique.

L'algorithme peut aussi être utilisé pour compléter une définition partielle de l'ordre des règles. Le concepteur exprime certaines successions obligatoires comme par exemple le fait que le groupe de règles G1 doive être effectué avant le groupe G2 et l'algorithme complète l'ordre en signalant si les successions prédéfinies ne peuvent être respectées (cas où le groupe G1 dépend du groupe G2).

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

Enfin, il est possible, lorsqu'une répartition en étapes a déjà été effectuée par le concepteur, de subdiviser chaque étape de manière à obtenir l'ordre d'exécution optimal des règles, afin d'accélérer les inférences.

Lors de la modification d'une base de connaissances, le principe consiste à ajouter les nouvelles règles hors des étapes définies dans la base de connaissances et à confier à l'algorithme le soin de placer ces règles correctement dans de nouvelles étapes ou dans les étapes existantes. L'algorithme prévient lorsque les nouvelles règles introduisent des dépendances bouclées entre les étapes existantes.

9.6. Détection d'incohérences sur un modèle Figaro 0

Malgré tous les éléments que nous avons donnés jusqu'ici, il y a bien sûr des cas où rien ne peut être démontré au niveau de la base de connaissances, car les règles de construction sûre sont trop contraignantes pour permettre de modéliser certains types de systèmes. Il reste alors possible de faire des vérifications sur un modèle Figaro d'ordre 0 particulier.

Soit par exemple la base de connaissances suivante, permettant de déterminer quels nœuds sont reliés à des sources dans un réseau de topologie donnée (NB : elle est complètement déterministe - l'automate peut prendre un seul état : l'état initial complet calculé à partir de l'état initial incomplet choisi par l'utilisateur) :

```

TYPE composant ;
  ATTRIBUT relie DOMAINE BOOLEEN PAR_DEFAULT FAUX;

TYPE noeud SORTE_DE composant ;
INTERFACE amont GENRE composant;
INTERACTION
  SI IL_EXISTE x UN amont TEL_QUE relie DE x
  ALORS relie
  SINON NON relie;

TYPE source SORTE_DE composant ;
INTERACTION
  ALORS relie ;

```

La variable d'état "relie" est un ATTRIBUT, et non un EFFET. Elle n'est donc pas réinitialisée avant chaque passage des règles d'interaction. Or, dans une topologie bouclée contenant deux nœuds n1 et n2, tels que n1 ait n2 dans son interface amont et réciproquement, les règles contenues dans les deux nœuds, après instanciation, se résument aux deux affectations $relie(n1) \leftarrow relie(n2)$ et $relie(n2) \leftarrow relie(n1)$. Donc, on voit que selon les valeurs initiales choisies par l'utilisateur (on suppose qu'il peut se tromper) pour $relie(n1)$ et $relie(n2)$, deux états sont stables par application des règles d'interaction : celui où n1 et n2 sont reliés, et celui où ils ne le sont pas. En fait, seul le deuxième état a un sens physique, étant donné l'absence de source dans le système.

Malgré ce défaut (qui serait corrigé tout simplement en déclarant "relie" comme un EFFET), cette base de connaissances est utilisable dans des topologies non bouclées. En effet, dans une topologie non bouclée, le graphe des dépendances entre les différents attributs "relie" du système est sans circuit. L'inférence va progressivement stabiliser les valeurs des différents attributs "relie", en partant des sources ou nœuds racines de la topologie du système, dont l'attribut "relie" est mis à VRAI pour les sources, et FAUX pour les nœuds racines, quel que soit l'état initial déclaré par l'utilisateur.

Plus généralement, il arrive souvent qu'en utilisant des règles en équivalence (SI ... ALORS ... SINON) on doive se contenter d'utiliser la base de connaissances uniquement sur des systèmes dont la topologie n'est pas bouclée.

Il est donc de loin préférable d'utiliser chaque fois que cela est possible l'inférence monotone, et ceci bien qu'elle conduise éventuellement à des bases de connaissances un peu moins lisibles, car les diverses conditions qui permettent de mettre une variable à VRAI peuvent être dispersées dans différentes règles.

Un autre exemple de détection d'incohérence possible seulement au niveau Figaro 0 et même en cours d'exécution du modèle est celui de la détection d'incohérences de type Inc2. Ce type de conflit est facile à illustrer par un réseau de Petri traduit en langage Figaro. L'examen du modèle permettrait de détecter ce que l'on appelle les conflits **structurels** du réseau, dus au fait qu'une place P est en amont de deux transitions instantanées t1 et t2 (cf. [11]). Mais il peut y en avoir beaucoup dans un modèle sans que ce soit un problème. C'est au moment de l'exécution que certains de ces conflits structurels se transforment éventuellement en conflits **effectifs** (cf. [11]). C'est le cas dans l'exemple pris ci-dessus, lorsque le tir de t1 consomme les jetons qui permettaient de valider t2

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

et/ou réciproquement. L'outil Figseq détecte ces conflits, avertit l'utilisateur de leur existence et affiche la séquence d'événements qui a amené le système de l'état initial à celui dans lequel le conflit est effectif.

9.7. Conclusion sur la cohérence des modèles

Nous avons montré que le langage de modélisation Figaro permet de répondre à de légitimes interrogations quant à la cohérence de modèles de systèmes de plus en plus complexes.

En particulier, grâce au respect de règles simples de conception des bases de connaissances exploitées avec KB3, il est possible d'assurer **que tous les modèles qui seront construits par les utilisateurs de ces bases seront cohérents** (y compris ceux dont la topologie est bouclée).

Ces principes ont été mis en œuvre en particulier dans les bases de connaissances utilisées à EDF **pour les études probabilistes de sûreté des centrales nucléaires**. Plus généralement, **la grande majorité** des études de fiabilité ou disponibilité qui ont été réalisées à l'aide de l'outil KB3 l'ont été avec des bases respectant ces principes.

Lorsqu'il est impossible de les appliquer intégralement, comme c'est le cas pour la base très complexe (27000 lignes de langage Figaro d'ordre 1) TOPASE, il est au moins possible de s'appuyer sur des outils d'analyse des dépendances entre règles qui aident à organiser celles-ci en sous-ensembles plus faciles à maîtriser.

Enfin, certains contrôles peuvent être effectués sur les modèles de systèmes particuliers, à défaut d'être faisables au niveau d'une base de connaissances.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

10. Références

- [1] Un moyen d'unifier diverses modélisations pour les études probabilistes : le langage Figaro
Bouissou M., Bouhadana H., Bannelier M.
Note HT- 53/90/42A
- [2] Conception d'un ensemble d'outils pour la représentation, l'analyse et l'exploitation de la connaissance en sûreté de fonctionnement : l'Atelier Figaro.
Bouhadana H.
Thèse de doctorat - Juin 92
- [3] Gestion de la complexité dans les études quantitatives de sûreté de fonctionnement de systèmes
Bouissou M.
Lavoisier, éditions TEC&DOC, Octobre 2008
- [4] Définition du langage Figaro0
Bouissou M., Dia M., Ladeuil V.
Note HT - 53/93/040
- [5] Syntaxe du langage de modélisation stochastique Figaro
Bouissou M., Buffoni L., Houdebine J.C.
Juin 2016
- [6] ATELIER Figaro – spécifications du sous-système Figaro1
Humbert S., Ladeuil V., Leroy F.
Note HT - 53/95/016A
- [7] Réflexions sur la génération d'arbres de défaillance dans l'atelier Figaro
Bannelier M., Villatte N.
Note HT - 53/93/22A
- [8] M. Bulot, I. Renault, Reliability studies for high voltage substations using a knowledge base: TOPASE project concepts and applications. Note interne EDF 96NR00101, ISSN 1161-0581, 1996.
- [9] J.F. Hery, J.C. Laleuf, Cohérence d'une base de connaissances : la convergence commutative en langage L.R.C. Note interne EDF HT 14/22/85, février 1985.
- [10] R. David, H. Alla, Du Grafcet aux Réseaux de Petri. 2ème édition, Hermes, 1992.
- [11] J. L. Peterson, Petri Net Theory and the Modelling of Systems. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
- [12] Dutuit Y., Signoret J.P., Thomas P., Prise en compte des transitions dynamiques au sein des réseaux de Petri stochastiques. Congrès Lambda-mu 20, St Malo, 2016.

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

11. ANNEXE 1 : Evolutions du langage Figaro

Dans cette annexe, nous présentons toutes les évolutions qui ont été apportées au langage depuis 1995, année de sortie de la version officielle précédente du manuel de référence.

Ces évolutions sont dues surtout à l'élargissement de la palette des traitements disponibles, c'est pourquoi les deux parties qui suivent sont relatives à ce qui a été ajouté pour la simulation de Monte Carlo et ce qui a été ajouté pour la génération des arbres de défaillances.

11.1. Ajouts pour la simulation de Monte Carlo

Lors d'une simulation de Monte Carlo, il est possible d'accéder pour chaque événement simulé à la date (de l'histoire simulée) à laquelle l'événement se produit. C'est ce qui a permis d'introduire le mot-clé `DATE_COURANTE` qui se comporte comme une variable globale que l'on peut consulter dans les règles. Pour la même raison il a été possible d'introduire les fonctions `TEMPS_DE_SEJOUR()` et `INTEGRALE()` dont les arguments sont des expressions booléennes.

Par ailleurs, puisque dans chaque histoire chaque variable aléatoire est échantillonnée, on a pu ajouter la fonction `RAND` (sans argument) qui produit un nombre aléatoire entre 0 et 1 ainsi que des fonctions donnant des nombres aléatoires tirés suivant différentes lois définies par leurs paramètres (leur nom commence par `RAND_`). La principale utilisation de ces lois est la définition des temps aléatoires avant défaillances ou réparations, mais il est aussi possible de les utiliser dans les règles.

Enfin, la fonction `DEJA_REALISE()` dont l'argument est une expression booléenne permet de faire un calcul de fiabilité sans être obligé de rendre les états de panne absorbants en les déclarant comme états cibles dans le paramétrage de l'outil YAMS.

11.2. Ajouts pour la génération d'arbres de défaillances

Les modèles de fiabilité associés aux pannes ont été ajoutés de façon à permettre de spécifier simplement dans une base de connaissances un ou plusieurs modèles (avec un modèle par défaut) que l'on retrouvera dans les événements de base des arbres de défaillances générés.

Avant cet ajout, le générateur d'arbres devait en quelque sorte reconnaître à partir des règles d'occurrence si une panne donnée était à la sollicitation ou en fonctionnement, si elle était réparable ou non... Et il paraissait bien difficile d'aller au-delà de ces quelques caractéristiques simples. Maintenant, le modélisateur dispose de tout un catalogue de modèles qu'il peut associer aux pannes.

Ce catalogue (cf. §6.2.4.4) a été constitué en faisant une compilation des modèles disponibles dans plusieurs outils de traitement des arbres de défaillances, et pourra être étendu si de nouvelles possibilités apparaissent dans ces outils.

11.1. Remplacement du pseudo-type GLOBAL par les objets système

L'existence du pseudo-type `GLOBAL` était une singularité par rapport à la philosophie orientée objet de Figaro. Elle répondait à une nécessité : celle de pouvoir accéder à des variables « globales » depuis tous les objets. Maintenant ce pseudo-type a disparu, mais en revanche on peut déclarer dans une base de connaissances des `OBJET_SYSTEME` (il peut y en avoir plusieurs). Ces objets existent dans tout système décrit avec cette base de connaissances et leurs variables sont accessibles depuis tout autre objet du modèle.

12. ANNEXE 2 : modèles de fiabilité associables à une PANNE

Ces modèles de fiabilité sont destinés à être affectés aux événements de base (feuilles) d'un arbre de défaillances généré à partir d'un modèle Figaro. Ils représentent des petits processus stochastiques indépendants. La feuille de l'arbre de défaillances est considérée comme ayant la valeur VRAI à un instant donné lorsque le processus correspondant est dans un état de panne à cet instant.

Chaque modèle est donné avec ses paramètres (ce sont tous des réels), et le cas échéant un petit graphe d'états représentant le comportement qu'il modélise. L'état initial est en jaune et les états de panne sont en rouge.

MODELE_GLM

Ce modèle représente un composant réparable qui peut tomber en panne à $t=0$ ou en fonctionnement. C'est de loin le plus utilisé.

Paramètres : **GAMMA** (taux de défaillance à la sollicitation), **LAMBDA** (taux de défaillance en fonctionnement), **MU** (taux de réparation).

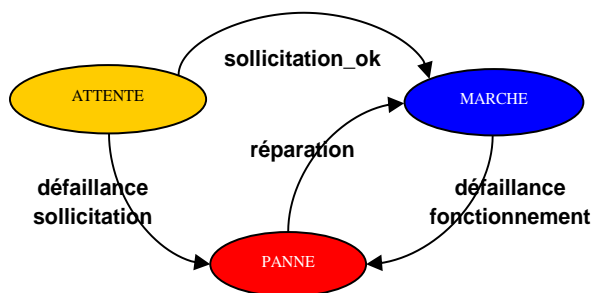


Figure 7. Graphe d'états du modèle MODELE_GLM

MODELE_GLTM

Ce modèle représente un composant non réparable qui peut tomber en panne à $t=0$ ou en fonctionnement, sur une durée limitée à **TM**.

Paramètres : **GAMMA** (taux de défaillance à la sollicitation), **LAMBDA** (taux de défaillance en fonctionnement), **TM** (temps de mission).

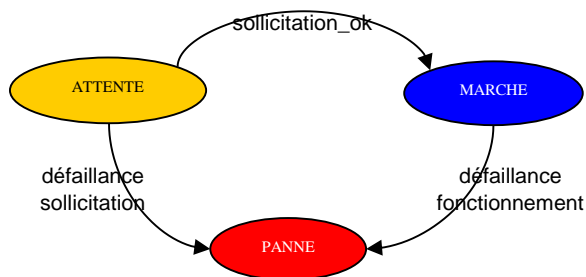


Figure 8. Graphe d'états du modèle MODELE_GLTM

MODELE_G

Ce modèle représente un composant non réparable qui peut tomber en panne à $t=0$. Suivant qu'il a démarré ou pas il reste indéfiniment en marche (respectivement : en panne).

Paramètre : **GAMMA** (taux de défaillance à la sollicitation).

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

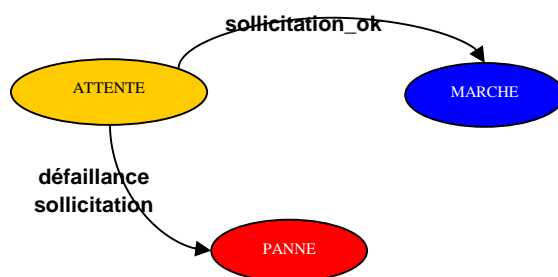


Figure 9. Graphe d'états du modèle MODELE_G

MODELE_FIGE

Ce modèle n'en est pas vraiment un : il sert à spécifier que la panne a une valeur constante, donc garde tout le temps sa valeur initiale (qui peut être choisie par l'utilisateur de la base de connaissances).

Paramètres : il n'y en a aucun.

MODELE_F

Ce modèle, d'utilisation très spécifique, représente un composant associé à une **fréquence** de défaillance constante. Ce type de modèle est utilisé dans les EPS (études probabilistes de sûreté des centrales nucléaires) pour représenter les incidents initiateurs.

Paramètre : FREQUENCE (mathématiquement, c'est la même chose qu'un taux de défaillance).

MODELE_WB

Ce modèle représente un composant non réparable dont la fiabilité est donnée par une loi de **Weibull**.

Paramètres : ALPHA (paramètre d'échelle), BETA (paramètre de forme), T0 (décalage de l'origine).

La fiabilité du composant est donnée, en fonction de ces trois paramètres, par la formule :

$$R(t) = \exp \left[- \left(\frac{t - T0}{ALPHA} \right)^{BETA} \right]$$

MODELE_TA

Ce modèle représente un composant réparable qui peut tomber en panne à t=0 ou en fonctionnement. La réparation ne démarre pas dès que le composant entre dans l'état de panne : il faut attendre qu'un **test aléatoire**, qui a un taux d'occurrence constant, détecte la défaillance. L'avantage de modéliser le test comme aléatoire est de permettre au modèle d'être markovien.

Paramètres : GAMMA (taux de défaillance à la sollicitation), LAMBDA (taux de défaillance en fonctionnement), MU (taux de réparation), LAMBDA_TEST (taux d'occurrence des tests).

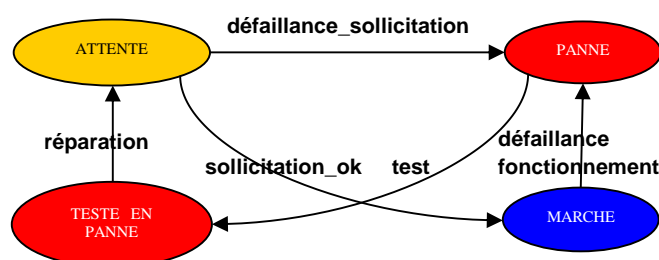


Figure 10. Graphe d'états du modèle MODELE_TA

MODELE_TPE

Ce modèle représente un composant réparable qui peut tomber en panne à $t=0$ ou en fonctionnement. La réparation ne démarre pas dès que le composant entre dans l'état de panne : il faut attendre qu'un **test périodique**, qui a lieu à intervalles de temps fixés, détecte la défaillance. Ce modèle, très complet (tellement complet qu'il est quasi-impossible de rassembler les valeurs de tous ses paramètres !) prend en compte le fait que le test n'est pas infaillible, qu'il peut même mettre en panne le composant qui était en bon état etc.

Paramètres : GAMMA (taux de défaillance à la sollicitation), LAMBDA (taux de défaillance en fonctionnement), MU (taux de réparation), T_INIT_TEST (instant du premier test), T_INTER_TEST (temps séparant deux débuts de tests), T_TEST (durée d'un test), GAMMA_NON_DETECTION (probabilité qu'un test ne détecte pas la panne du composant), GAMMA_TEST (probabilité qu'un test mette en panne le composant qui fonctionnait bien), GAMMA_RECONFIG (probabilité de mauvaise reconfiguration à la fin d'une réparation).

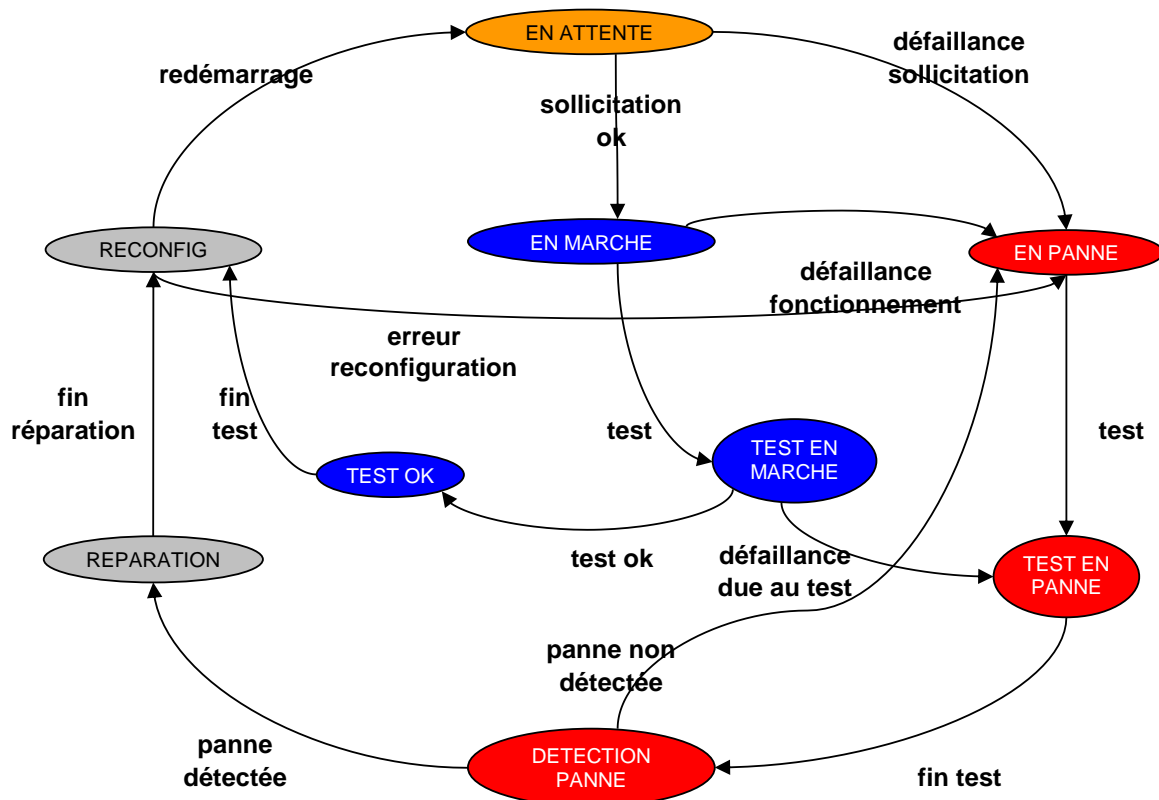


Figure 11. Graphe d'états du modèle MODELE_TPE

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

13. ANNEXE 3 : INDEX

?

? · 92

A

Action · 71
ALORS · 82
ATTRIBUT · 66
AU_MOINS ... PARMI · 53

B

Base de connaissances · 12
BOOLEEN · 64

C

CARDINAL · 16, 62
CARDINAL (opérateur) · 54
Chaînage arrière · 28
Chaînage avant · 23
Chaîne de caractères · 34
Champs · 17
Commentaire · 34
Condition de règle · 70
CONSTANTES · 63

D

DE · 42, 43
DE_TYPE · 39, 40
DEFAILLANCE · 74, 78
DES_TERMES · 55, 56
DOMAINE · 44, 64

E

EDITION · 61
EFFET · 66, 69
Ensemble d'objets · 37
ENTIER · 64
Énumération · 34, 58, 89
EQUATION · 91
ET · 46
ETAPE · 82
Etat d'un modèle FIGARO · 19
Etat initial d'un modèle FIGARO · 59, 62, 64, 67
Événement · 19, 74, 75
EXP, EXPONENTIELLE · 77
Expressions · 37

F

FAIRE · 72
FAUX · 35
FORMULE · 92

G

GENRE · 62, 92
GLOBAL · 43, 87
GROUPE · 76, 82, 89

H

Héritage · 13, 58, 87

I

Identificateurs · 35
IL_EXISTE · 47
IL_EXISTE AU_MOINS · 49
IL_PEUT_SE_PRODUIRE · 76
INCLUS_DANS · 49, 51
INDISPONIBILITE · 74, 80
INFINI · 63
INS, INSTANTANEE · 77
INTERACTION · 82
INTERFACE · 15, 62

J

JUSQUA · 63

L

LIBELLE · 64, 65, 67, 77
LINEAIRE · 92
Liste · 34, 61, 71, 90
LOI · 77

M

MARCHE · 52
Modèle FIGARO · 13
MODIFIABLE · 61
Mots-clés · 35

EDF R&D	Manuel de référence du langage de modélisation probabiliste Figaro	Version E
---------	--	-----------

N

NOM_DES_GROUPES · 88
 NOM_DES_SYSTEMES · 92
 NON · 46
 NON MODIFIABLE · 61
 NON OBLIGATOIRE · 62
 NON VISIBLE · 61

O

Objet · 11
 OBLIGATOIRE · 61
 OCCURRENCE · 76
 ON_A · 48
 ORDRE_DES_ETAPES · 90
 OU · 46
 OU_BIEN · 74, 76

P

PANNE (opérateur) · 53
 PANNE (variable d'état) · 66, 67, 74
 PAR_DEFAULT · 60
 PARAMETRE_LOI · 64
 POUR_TOUT · 55, 56, 72
 PRODUIT · 55
 PROVOQUE · 78

Q

QQSOIT · 48

R

REEL · 64

Règle · 19, 70
 Règle d'interaction · 81
 Règle d'occurrence · 73
 REINITIALISATION · 65, 67
 REPARATION · 74, 80
 REPARE · 77, 81
 RESOUDRE_SYSTEME · 73

S

SI · 76, 82
 SINON · 82
 SOIT · 82
 SOMME · 54
 SORTE_DE · 58
Surcharge · 14, 87
 Système d'équations · 73
 SYSTEME_EQUATIONS · 92

T

T_C, TEMPS_CONSTANT · 77
 TEL_QUE · 47
 TRANSITION · 74, 78
 TYPE · 11, 58

U

UN · 47, 48

V

variable liée · 42
 Variable liée · 37
VERIFIANT · 41
 VISIBLE · 61
 VRAI · 35