REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE USTO-MB

Faculté des sciences
Département d'Informatique

Mémoire présenté par

Melle DJEBBAR Nacera

Pour l'obtention

DU DIPLOME DE MAGISTER

Spécialité : Informatique
Option : Systèmes, Réseaux, et Bases De Données

Modélisation orientée langage pour la sûreté de fonctionnement des systèmes

Soutenu le : 03/07/2011

Devant les membres du jury

Mme	**BELBACHIR Hafida**	**Professeur**	**USTO**	**Présidente**
Mme	**NOUREDDINE Myriam**	**Maître de conférences**	**USTO**	**Rapporteur**
Mme	**MEKKI Rachida**	**Maître de conférences**	**USTO**	**Examinatrice**
Mr	**BELKADI Khaled**	**Maître de conférences**	**USTO**	**Examinateur**

REMERCIEMENTS

- J’exprime mes profonds remerciements à mon encadreur, Mme Myriam Noureddine pour l’aide précieuse qu’elle m’a apportée. Son œil critique m’a été très profitable pour améliorer la qualité de ce mémoire.

- Je tiens à remercier également Mme H. Belbachir, Responsable de la Post Graduation, de m’avoir fait l’honneur de présider mon jury.

- Je souhaite exprimer ma reconnaissance aux membres du jury, Mme R. Mekki, et Mr K. Belkadi qui ont bien voulu accepter de juger mon travail.

- Je remercie aussi ma famille, mes collègues, mes camarades, et tous mes amis pour leur aide, soutien et encouragements.

- Merci également à tous ceux qui m’ont apporté leur aide pour parvenir au terme de ce travail.

SOMMAIRE

II LANGAGES POUR LA SURETE DE FONCTIONNEMENT DES SYSTEMES

III DEMARCHE DE MODELISATION ORIENTEE LANGAGES

IV MISE EN ŒUVRE EXPERIMENTALE

LISTE DES TABLEAUX

CHAPITRE I

CHAPITRE III

CHAPITRE IV

LISTE DES FIGURES

CHAPITRE I

CHAPITRE II

CHAPITRE III

CHAPITRE IV

INTRODUCTION GENERALE

Les activités industrielles engendrent divers accidents et événements catastrophiques qui ont des conséquences lourdes sur les vies humaines et l'environnement.

Afin de maîtriser les risques au maximum, les industriels ont eu recours à une discipline appelée Sûreté de Fonctionnement (SdF). La construction manuelle des modèles classiques utilisés en SdF (arbres de défaillances, graphes de Markov, réseaux de Petri, etc.) pour l'analyse et l'évaluation des systèmes demande à la fois beaucoup de temps et d'expertise, comporte des risques d'erreurs et ne permet pas la capitalisation des connaissances. Il a donc fallu définir un langage de plus haut niveau capable d'être traduit de façon automatique vers ces modèles classiques et actuellement, il existe plusieurs langages pour la sûreté de fonctionnement.

Ce mémoire de magister propose une démarche pour la modélisation orientée langage des systèmes, dans un contexte de sûreté de fonctionnement et dans le but de permettre par la suite des analyses qualitatives et quantitatives des systèmes.

Ce document est structuré en quatre chapitres, hormis l'introduction générale et la conclusion générale :

- Dans le premier chapitre, nous présentons la sûreté de fonctionnement des systèmes, à travers les notions de bases ainsi que les méthodes support générant des modèles.
- Dans le deuxième chapitre, nous abordons les langages de sûreté de fonctionnement, suivant des concepts généraux puis en étudiant de manière détaillée trois langages considérés comme des incontournables par les industries à risques.
- Dans le troisième chapitre, tout d'abord nous comparons ces trois langages suivant plusieurs niveaux. Cette comparaison nous a permis ensuite de définir une démarche pour la modélisation orientée langage pour les études de sûreté de fonctionnement.
- Dans le quatrième et dernier chapitre, nous validons notre démarche par une mise en œuvre expérimentale sur un système critique.

A la fin de ce mémoire nous donnons les résultats de notre étude, et proposons quelques perspectives pour les travaux futurs.

I. SURETE DE FONCTIONNEMENT DES SYSTEMES

I.1 Introduction

Dans le domaine de l'analyse des systèmes, deux communautés cohabitent : la vérification formelle «VF» et la sûreté de fonctionnement «SdF»; la nuance est plutôt subtile mais elle permet de distinguer deux approches : le fonctionnel et le dysfonctionnel. Dans le domaine de la SdF, le principal objectif est de maîtriser le risque inhérent à des systèmes complexes (de par leurs dimensions ou leurs hétérogénéités technologiques) tandis que la vérification formelle est orientée vers les aspects fonctionnels du système, ce système à une fonction à remplir, cette fonction étant généralement très complexe, il est nécessaire de vérifier si elle est bien réalisée ou réalisable par le système (Point, 2000).

Bien que les techniques d'analyse diffèrent, les études sur les systèmes dont les conséquences de dysfonctionnements ne sont pas négligeables et peuvent être économiques, écologiques, ou humaines (systèmes critiques) ont la même finalité : avoir un certain niveau de confiance en le système. En effet, il ne suffit pas de montrer qu'un système est correct d'un point de vue fonctionnel mais il est nécessaire d'envisager les raisons et les conséquences de ses dysfonctionnements (Point, 2000).

Dans le cadre de ce mémoire, nous nous plaçons dans le domaine de la sureté de fonctionnement, et nous présentons dans les sections suivantes ses concepts de base, ses méthodes et modèles les plus utilisés.

I.2 Concepts de base de la sureté de fonctionnement

I.2.1 Système et composants

A- Système

Un *système* est défini comme un ensemble d'éléments qui interagissent entre eux. Tout système se caractérise par une ou plusieurs fonctions qu'il doit accomplir (Megdiche, 2004).

Les éléments définissant un système sont (Megdiche, 2004) :

- Les fonctions assurées par le système.
- L'architecture du système, qui comprend les divers composants mis en jeu et leurs connections.
- Le système d'exploitation, qui représente les comportements du système durant son fonctionnement.

B- Composant

La définition des *composants* d'un système peut se faire de multiples façons. En effet, un composant peut être lui même décomposé en sous composants. Il est ainsi nécessaire de fixer la finesse de l'étude afin de déterminer le choix des composants (Megdiche, 2004).

I.2.2 Sûreté de fonctionnement

La *sûreté de fonctionnement « SdF »* vise la maîtrise des risques (Peytavin, 1998; Point, 2000; Megdiche, 2004 ; Messaoudi, 2010); elle fournit les données nécessaires pour indiquer le degré de confiance que l'utilisateur peut avoir sur le système et pour cela elle utilise des concepts et des outils pour faire une analyse des systèmes.

Le *risque* possède deux dimensions : l'événement non souhaité et la mesure associée à ses conséquences, la maîtrise de la première est la plus naturelle car elle vise à diminuer

(en général) la fréquence d'occurrence de la défaillance du système. La seconde dimension est apparue lorsque, avec le retour d'expérience, les industriels ont pris conscience qu'il était inutile de concevoir des systèmes trop sécurisés et, par voie de conséquence, trop onéreux par rapport aux conséquences du risque lui-même (Point, 2000).

I.2.3 Défaillances

Dans le cadre de la SdF, un système connaît une défaillance lorsqu'il n'est plus en mesure de remplir sa (ou ses) fonction (s) et une défaillance est la cessation de l'aptitude du système (ou d'un dispositif) à accomplir une fonction requise (Villemeur, 1988; Zwingelstein, 2009).

Les défaillances d'un même composant peuvent avoir des conséquences différentes. Il est ainsi important de distinguer, pour un composant, les défaillances (modes de défaillances) qui ont des conséquences différentes et qui, donc, entrainent des comportements différents du système (Megdiche, 2004).

I.2.4 Paramètres et lois de sûreté de fonctionnement

Les principaux paramètres de sûreté de fonctionnement (Megdiche, 2004) sont illustrés dans le tableau suivant :

Paramètre	Désignation	Définition
La fiabilité	R(t)	probabilité qu'une entité fonctionne entre l'instant 0 et t
La disponibilité	A(t)	probabilité qu'une entité fonctionne à l'instant t
La maintenabilité	M(t)	probabilité qu'une entité soit réparée entre l'instant 0 et t
La sécurité	S(t)	probabilité qu'une entité n'ait aucune défaillance catastrophique entre l'instant 0 et t
Le taux de défaillance	$\Lambda(t)$	L a probabilité qu'une défaillance d'un composant apparaît entre t et t + dt, sachant que le composant a fonctionné entre 0 et t
Le taux de réparation	$\Psi(t)$	La probabilité qu'un composant soit réparé entre t et dt sachant qu'il était défaillant entre 0 et t
Le taux de défaillance a la sollicitation	$\gamma(t)$	la probabilité que le composant tombe en panne au moment ou il est sollicité

Les paramètres constants	λ	Approximation de $\Lambda(t)$
	μ	Approximation de $\Psi(t)$

Tableau.I.1. Paramètres de sûreté de fonctionnement

Parmi les lois les plus utilisées pour la quantification de la sureté de fonctionnement on cite la loi exponentielle :

- *Loi exponentielle* (Megdiche, 2004; Pagetti, 2010) : Utilisée fréquemment car les calculs sont simples. Pour la loi de probabilité exponentielle, le taux de défaillance est égal à λ et la moyenne de la loi, c'est-à-dire le temps moyen de fonctionnement, est égale à $1/\lambda$. Le paramètre λ de loi se confond avec le taux de défaillance Λ. On emploi souvent le terme λ pour qualifier le taux de défaillance Λ (et respectivement μ pour le taux de réparation Ψ). Les simplifications induites par cette loi font qu'elle est très souvent employée pour les calculs de sureté de fonctionnement.

I.3 Analyses préliminaires à la sûreté de fonctionnement

I.3.1 Analyse fonctionnelle

Avant de faire une étude de sûreté de fonctionnement il est indispensable de faire une analyse fonctionnelle afin de déterminer les caractéristiques du système (fonctions, structure, etc.).L'analyse des systèmes ne possède pas de méthode standard, les méthodes utilisées diffèrent selon la nature des systèmes (technologies déployées) et de l'objectif de l'étude.

Pour effectuer cette analyse il est nécessaire d'identifier les caractéristiques des systèmes (Alani, 2006) : les fonctions, la structure, les technologies déployées, les modes de fonctionnement, les conditions d'exploitation, l'environnement du système et l'inventaire des moyens de mesures.

I.3.2 Analyse des risques

Il s'agit à la base d'une analyse qui permet d'identifier des points critiques devant faire l'objet d'études plus détaillées. Les méthodes les plus utilisées dans cette analyse sont : l'Analyse Préliminaire des Risques (APR) et l'Analyse des Modes de Défaillances et de leurs Effets (AMDE).

A- Analyse Préliminaire des Risques

La méthode d'Analyse Préliminaire des Risques « APR » a deux objectifs (Villemeur, 1988) :

- Identifier les dangers d'une installation industrielle et ses causes.
- Evaluer la gravité des conséquences liées aux situations dangereuses et aux accidents potentiels.

On en déduit les actions correctrices éliminant ou maîtrisant les situations dangereuses, et les accidents potentiels. Elle permet, en autre, de définir des dangers majeurs, des entités à analyser en détail, en faisant appel à d'autres méthodes (comme par exemple, les arbres de défaillances) (Villemeur, 1988).

B- Analyse des Modes de Défaillances et de leurs effets

L'analyse des Modes de Défaillances et de leurs Effets AMDE ET l'AMDE(C) (complétée par une analyse de la criticité) sont des outils complets permettant de formaliser un grand nombre de connaissances sur les installations industrielles, en particulier par l'analyse exhaustive de tous les modes possibles de pannes reliés à leurs causes et leurs effets (**Tableau.I.2.**). La méthode AMDE(C) est une méthode qualitative et inductive visant à recenser les défaillances, puis à en estimer les risques (Noureddine, 2009)

L'AMDE et l'AMDE(C) constituent des analyses préliminaires qui doivent être complétées par l'utilisation d'autres méthodes pour l'identification des combinaisons de défaillances pertinentes. Une AMDE ne peut être généralement réalisée que pour un état bien défini du système ; sinon le travail risque de devenir très lourd et sans réel objet (Villemeur, 1988).

Composant	Mode de défaillance	Origine de la défaillance	Conséquences de la défaillance
1ier composant	1er mode de défaillance		
	…		
….			

Tableau.I.2. Analyse des modes de défaillances

I.4 Classification des méthodes de sûreté de fonctionnement

I.4.1 Analyse qualitative et quantitative

L'évaluation de la sûreté de fonctionnement fait appel à de nombreux modèles et aux méthodes d'analyse qualitative et quantitative (Belhadaoui *et al.*, 2007) associées à ces modèles.

A- Analyse qualitative

L'analyse qualitative permet d'identifier les relations de causes à effets entre fautes, erreurs et défaillances. (par exemple quelles fautes peuvent entraîner une défaillance).

B- Analyse quantitative

L'analyse quantitative évalue les paramètres probabilistes de la sûreté de fonctionnement.

I.4.2 Analyse statique et dynamique

L'analyse de sûreté de fonctionnement est caractérisée par le type d'analyse, et le type du système étudié (Point, 2000). L'analyse peut être une analyse une analyse statique, ou bien une analyse dynamique du système.

A- Analyse statique

Un composant, un sous-système ou le système lui-même possède uniquement deux états: marche et panne. Les modèles statiques décrivent des formules booléennes (modèles booléens), et l'étude est faite sur un état bien défini du système. Les méthodes les plus utilisées dans cette analyse sont : les Arbre de Défaillance (AdD).

B- Analyse dynamique

Cette analyse prend en compte la reconfiguration, les redondances, les modes de fonctionnement, les réparations, la maintenance et utilise des modèles comportementaux ou dynamiques. Parmi les méthodes les plus utilisées dans cette analyse on cite les Graphes de Markov (GM), et les arbres d'événements.

I.5 Méthodes pour la sûreté de fonctionnement

I.5.1 Arbre de défaillances

On construit et on utilise un arbre de défaillances dans le cadre d'une étude à priori d'un système. Ayant pour point de départ un événement redouté (dysfonctionnement ou accident), la démarche consiste à s'appuyer sur la connaissance des éléments constitutifs du système étudié pour identifier tous les scénarios conduisant à l'événement redouté. L'arbre de

défaillance est une représentation en deux dimensions (**Figure.I.1.**) des enchaînements qui peuvent conduire à l'événement redouté, le point de départ de la démarche. On peut ensuite utiliser cette représentation pour calculer la probabilité de l'événement redouté à partir des probabilités des événements élémentaires qui se combinent pour le provoquer (Mortureux, 2002).

Pour faciliter la construction d'un arbre défaillance, il est utile de réaliser préalablement une AMDE, l'algèbre booléenne peut aussi être utilisée pour trouver les coupes minimales, définies comme étant « la plus petite combinaison d'événements entraînant l'événement indésirable » (Villemeur, 1988).

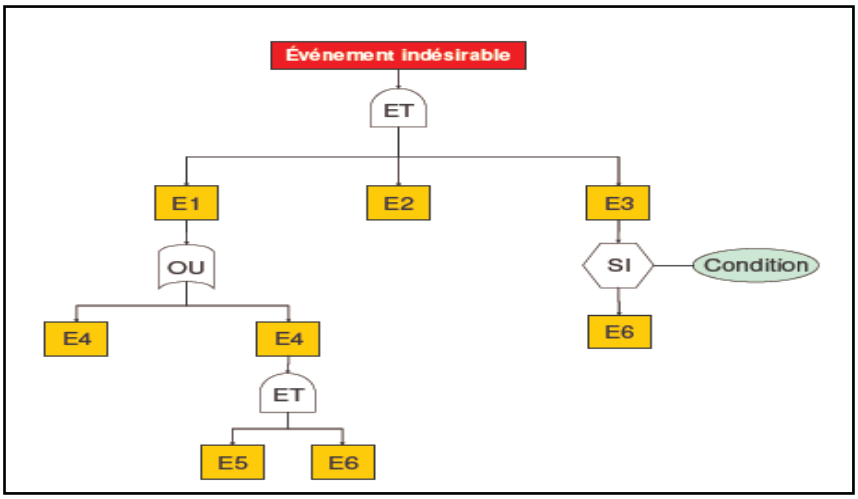
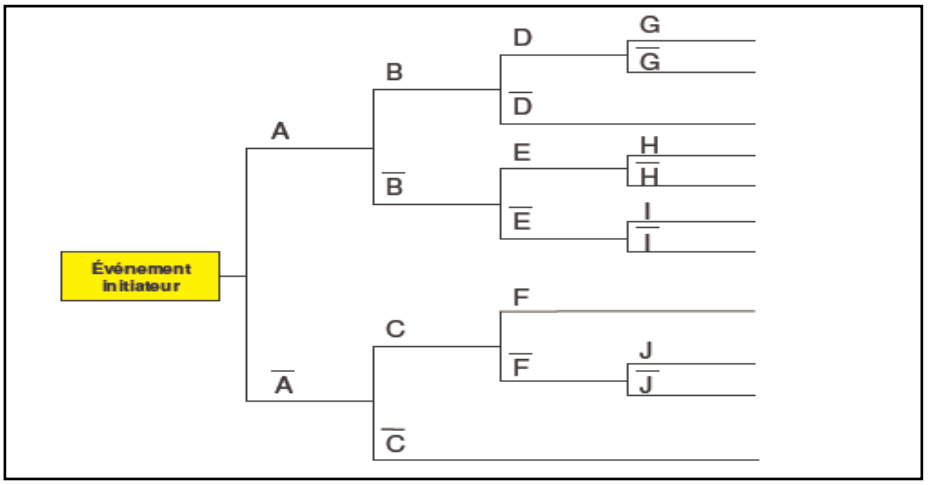


Figure.I.1. Structure d'un arbre de défaillances (Megdiche, 2004)

I.5.2 Arbre d'événements

Le raisonnement employé pour cette méthode est l'inverse de l'arbre de défaillance. On part d'un événement dit initiateur puis on envisage toutes suites possibles chronologiquement jusqu'a retour a l'état normal du système. Partant de l'événement initiateur, le comportement du système face à cette défaillance se décompose en plusieurs étapes. A chaque étape, on envisage le bon déroulement de l'étape ou le mauvais déroulement. Les événements initiateurs sont choisis comme étant toutes les défaillances premières susceptibles de se produire (Megdiche, 2004).

De par sa complexité proche de celle de l'analyse par arbre des défaillances, cette méthode s'applique préférentiellement sur des sous-systèmes bien déterminés. Elle apporte une aide précieuse pour traiter des systèmes comportant de nombreux dispositifs de sécurité et de leurs interactions. À l'instar de l'analyse par arbre des défaillances dont elle s'inspire, elle permet d'estimer les probabilités d'occurrence de séquences accidentelles. Cette méthode est particulièrement utilisée dans le domaine de l'analyse après accidents en vue d'expliquer les conséquences observées résultant d'une défaillance du système [site 1].



FigureI.2. Structure d'un arbre de conséquences (Megdiche, 2004)

I.5.3 Autres méthodes.

Il existe d'autres méthodes de sûreté de fonctionnement, et nous présentons quelques unes :

A- Diagramme de succès

La méthode du diagramme de succès ou diagramme de fiabilité modélise les conditions de bon fonctionnement du système ou de réalisation d'une de ses fonctions. Elle consiste à construire un diagramme composé de blocs, chacun d'eux représentant une entité (composant, sous-système) reliés par des lignes orientées indiquant les dépendances des entités entre elles. Un chemin à sucés est un ensemble de blocs qui assurent la fonction du système (Pagetti, 2010; Point, 2000; Zwingelstein, 2009).

B- Graphe de Markov

Un processus stochastique est un ensemble de variables aléatoires $(X_t)_{t\geq 0}$ à valeurs dans l'ensemble des observations. Un processus est markovien si la probabilité de passage de l'étape présente à la suivante ne dépend pas du passé (Pagetti, 2010).

Le principe des graphes de Markov est de représenter tous les états du système (états des composants et mode de fonctionnement du système) et toutes les transitions possibles entre ces états. La transition d'un état i vers un état j est la probabilité que le système, étant dans l'état i à l'instant t, passe à l'état j à l'instant t+dt (Megdiche, 2004).

C- Réseau de Petri

Les réseaux de Petri sont un bon outil pour modéliser le comportement dysfonctionnel d'un système : on appréhende plus aisément les différentes pannes et l'impact sur le système (Pagetti, 2010).

Un réseau de Petri «RdP» est un graphe orienté avec deux types de noeuds : les places (états ou conditions) représentées par des cercles et les transitions (ou événements) symbolisées par des barres. Ces noeuds sont connectés entre eux par des arcs orientés à des places aux transitions et des transitions aux places exclusivement. La circulation de jetons (marqueurs indivisibles), symbolisant la présence à un instant donné d'une information ou d'une initialisation particulière aux places ou ils résident, permet la modélisation dynamique du comportement système (aussi bien désiré que redouté) au sein du réseau. Un réseau de Petri stochastique est un réseau de Petri étendu tel qu'on associe à chaque transition une durée de franchissement aléatoire ou déterministe (nulle ou non). Si la durée déterministe est 0, on parle de transitions immédiates (Pagetti, 2010).

Si on représente le graphe des états associés, au système de transition entre marquage on retombe sur la chaîne de Markov représentant le système. L'exploitation quantitative d'un réseau de Petri se fait généralement, soit en traitant le modèle markovien issu du graphe des

marquages accessibles du réseau de Petri, soit en animant par simulation de Monte-Carlo directement le modèle réseau de Petri (Pagetti, 2010).

D- Simulation de Monte-Carlo

Le principe de la méthode par simulation de Monte Carlo «SMC» est de simuler la vie du système au cours de sa période de fonctionnement. Plusieurs échantillons sont nécessaires pour déterminer de façon statistique les indices de sûreté de fonctionnement du système. Les résultats données par cette méthode ne sont pas les solutions numériques exactes mais des intervalles de confiance les contenant avec une probabilité donnée (Megdiche, 2004). Cette méthode est fréquemment rencontrée lorsque (Point, 2000) :

- Le processus stochastique est markovien mais l'ensemble des états est trop important pour être stocké ou pour appliquer les méthodes analytiques usuelles.
- Le processus est non markovien ce qui interdit l'utilisation des méthodes analytiques.

La simulation de Monte Carlo est une méthode à la fois très générale et insensible au nombre d'états. Cependant, les résultats qu'elle produit peuvent être imprécis et demander des temps de calcul prohibitifs pour des systèmes très fiables. Cette limitation de la simulation explique pourquoi les calculs analytiques sur les graphes de Markov, qui sont d'autant plus efficaces que les systèmes sont plus fiables, ont un grand intérêt (Bouissou, 2006).

E- Boolean Logic Driven Markov Process

Les arbres de défaillances sont le type de modèle statique de loin le plus utilisé par les ingénieurs chargés de faire des études de sûreté de fonctionnement de systèmes (Bouissou, 2006). Grâce a un nouveau formalisme appelé : «BDMP » (Boolean Logic Driven Markov Process) (Bouissou, 2005; Bouissou, 2006 ; Nedjari *et al.,* 2009), il est possible de spécifier, à l'aide de modèles graphiques ressemblant fortement aux arbres de défaillances, des graphes de Markov de très grande taille.

I.6 Conclusion

Dans ce chapitre nous avons présenté les principes de bases de la sûreté de fonctionnement, ses méthodes et modèles les plus utilisés. Etant donnée la diversité des méthodes et modèles utilisés pour une même étude, la difficulté de construire ces modèles pour des systèmes complexes, le risque d'erreur engendré par la construction manuelle de ces modèles, et l'impossibilité de leurs réutilisation et mise à jour, il devient indispensable d'utiliser un modèle générique capable d'être traduit vers ces modèles, et pouvant être réutilisé et modifié facilement, d'où l'intérêt des langages de sûreté de fonctionnement que nous allons présenter dans le chapitre suivant.

II LANGAGES POUR LA SURETE DE FONCTIONNEMENT DES SYSTEMES

II.1 Introduction

Un langage de modélisation pour la SdF des systèmes est un langage (Bouissou *et al.*, 2006) compositionnel, partagé par ceux qui font les analyses de sûreté de fonctionnement et les concepteurs de systèmes, qui permet des études qualitatives et quantitatives de même type que celles que l'on fait avec les méthodes classiques, qui permet une bonne structuration des connaissances de façon à permettre la capitalisation et la réutilisation des modèles, qui possède une syntaxe et une sémantique claire, et qui est facile à documenter et à associer à des visualisations graphiques variées.

Dans ce contexte, on cite les langages FIGARO, AltaRica, et AADL; ces langages, qui sont les plus utilisés actuellement, sont à la une des recherches et des développements, c'est pour quoi dans ce chapitre nous présentons l'historique, les principales caractéristiques, les concepts de bases, et les outils de ces langages ainsi qu'une brève présentation d'autres langages de SdF.

II.2 Langage FIGARO

II.2.1 Historique

Le langage « FIGARO » (Torrente *et al.*, 2008) a été le premier créé spécialement pour faciliter la construction de modèles de SdF. Il a été développé en 1989-1990 à EDF-R&D (entreprise d'Electricité De France-Recherche et Développement) et validé depuis par des centaines d'études de systèmes complexes. C'est ce langage qui a permis à EDF de capitaliser dans des bases de connaissances à la fois pérennes et évolutives, la majorité des connaissances sur la SdF des systèmes hydrauliques, électriques, de contrôle-commande, de télécommunications étudiées au fil des ans.

II.2.2 Principales caractéristiques

Les caractéristiques du langage Figaro sont (Bouissou *et al.*,2006; Torrente *et al.,* 2008) :

1. FIGARO fait partie des langages dit «Hybride», c'est-à-dire qu'il prend certains de ses caractéristiques des langages orientés objet et modélise le comportement d'un objet par des règles de production d'ordre 1.
2. Pour le langage FIGARO il existe clairement deux classes d'utilisateurs : ceux qui développent des bases de connaissances et ceux qui les utilisent.
3. Le langage « FIGARO d'ordre 1 » sert à écrire des modèles très génériques dans des bases de connaissances.
4. Le langage « FIGARO d'ordre 0 » est un sous langage très simplifié du langage FIGARO. Il ne permet d'écrire que des modèles spécifiques à un système donné. Le rôle premier du langage FIGARO0 est de servir d'interface entre l'outil de construction graphique de modèles et les outils de traitement.
5. Sur la base d'un langage de modélisation FIGARO, différents compilateurs et traducteurs permettent de déduire automatiquement les données qui sont nécessaires aux outils de traitements des modèles classiques : arbres de défaillance, chaînes de Markov, réseaux de Petri, etc.
6. Le langage FIGARO a bénéficié d'une grande stabilité à la fois syntaxique et sémantique, ce qui a permis de réutiliser les types FIGARO de certaines bases de connaissances plus de dix ans après leur création, pratiquement sans adaptation, avec l'outil « KB3».
7. Le langage FIGARO permet d'écrire des modèles probabilistes quantifiables pouvant être simplifiés en modèles qualitatifs.
8. La sémantique du langage « FIGARO 0 » équivaut au fonctionnement d'un automate.

III.2.3 Concepts de base

A- FIGARO 1 (Villate, 2005; Torrente et *al.,* 2008)

Les *bases de connaissances* « BDC » son écrites en langage « FIGARO 1 », langage spécifique développé à EDF. Chaque base de connaissances est sous la responsabilité d'un administrateur qui écrit et fait évoluer la base avec l'aide d'experts. Pour une catégorie donnée de systèmes, une base de connaissances contient la description générique des *types de composants* que l'on peut rencontrer dans les systèmes de cette catégorie en décrivant leurs comportements. Dans une base de connaissances, un type X est décrit par :

- Son nom.
- Ses variables appartenant à différentes familles tel que : *attribut, constante, effet, panne,* auxquelles sont affectés des valeurs par défaut.
- Des *interfaces* dont le genre est un type Y de la BDC; elles vont permettre de mettre des objets instances du type Y en relation avec des objets instances du type X.
- Deux types de règles décrivant un comportement générique fonction des variables et des interfaces :
  - *Règles d'interaction* « RI » : modélisent la propagation des effets instantanés.
  - *Règles d'occurrences* « RO » : donnent la liste des événements qui peuvent arriver dans un état du système.

B- FIGARO 0 (Villatte, 2005)

Des *modèles comportementaux* « FIGARO 0 » sont créés, par assemblage d'objets (composants élémentaires) *instances de types* déclarés dans la base de connaissance liée à l'étude. Un *objet* est décrit par :

- Son nom.
- Ses variables, elles sont issues de la base de connaissances (héritées du type dont l'objet est une instance); ou ajoutées par l'utilisateur.
- Ses interfaces (héritées du type dont l'objet est une instance), elles contiennent d'autres objets du système mis en relation avec l'objet étudié. Les objets présents dans une

interface sont des instances du type spécifié dans le genre de l'interface. Le nombre d'objets dans une interface est borné par une cardinalité fixée par la base de connaissance.

II.2.4 Outils

Nous présentons dans cette section les outils support du langage FIGARO :

A- VisualFigaro

« VisualFigaro » est un environnement qui permet de développer ou de modifier des bases de connaissances et intègre un ensemble d'outils d'éditions de modèles FIGARO, de vérification syntaxique et de paramétrage de l'IHM (Interface Homme Machine) de KB3 (outil FIGARO présenté dans la section suivante). En s'appuyant sur cet environnement, la méthode de développement d'une nouvelle base de connaissances passe par les étapes suivantes (Torrente *et al.*, 2008) :

1. Définition de la structuration statique de l'ensemble des types FIGARO (leurs noms, leurs relations d'héritage, leurs interfaces).
2. Définition d'un paramétrage minimal de l'IHM de KB3 (choix des icônes, de caractéristiques de liens, instructions de remplissage des interfaces).
3. Itérations sur :
   - L'écriture des règles décrivant le comportement des types FIGARO.
   - Les tests sur des petits exemples d'assemblages de composants.
4. Ajout de variantes graphiques (changements de couleurs, affichage de textes prévus pour la simulation interactive).
5. Ajout de contraintes sur la saisie pour rendre l'outil créé robuste.

B- KB3

L'utilisateur de KB3 utilise à la fois une base de connaissances adaptée à son application et le logiciel KB3 lui-même qui est constitué d'une interface pour la description graphique du système étudié et des buts de l'étude. Cette interface permet la construction de modèles de systèmes industriels complexes. A partir de cette description graphique dans l'IHM de KB3 et des connaissances décrites dans la base, le logiciel, KB3 génère un modèle dit FIGARO à l'ordre 0 (Torrente *et al.*, 2008). Ce modèle unique est utilisé (Torrente *et al.*, 2008) :

- Soit par le générateur d'arbres de défaillances pour les modèles statiques.
- Soit par le générateur de séquences « FIGSEQ » pour les modèles dynamiques markoviens.
- Soit par l'outil de simulation de Monte Carlo « YAMS » pour tous les modèles (qu'ils soient markoviens ou non).

La Plate-forme d'outils KB3 permet aussi aux fiabilistes de construire graphiquement tous leurs modèles habituels : arbres de défaillances, bien sûr, mais aussi diagrammes de fiabilité, graphes de Markov, réseaux de Petri, réseaux de fiabilité et de nombreuses extensions de ces modèles standard, dont en particulier le formalisme BDMP [site 2]. La figure suivante illustre le fonctionnement de l'atelier KB3.

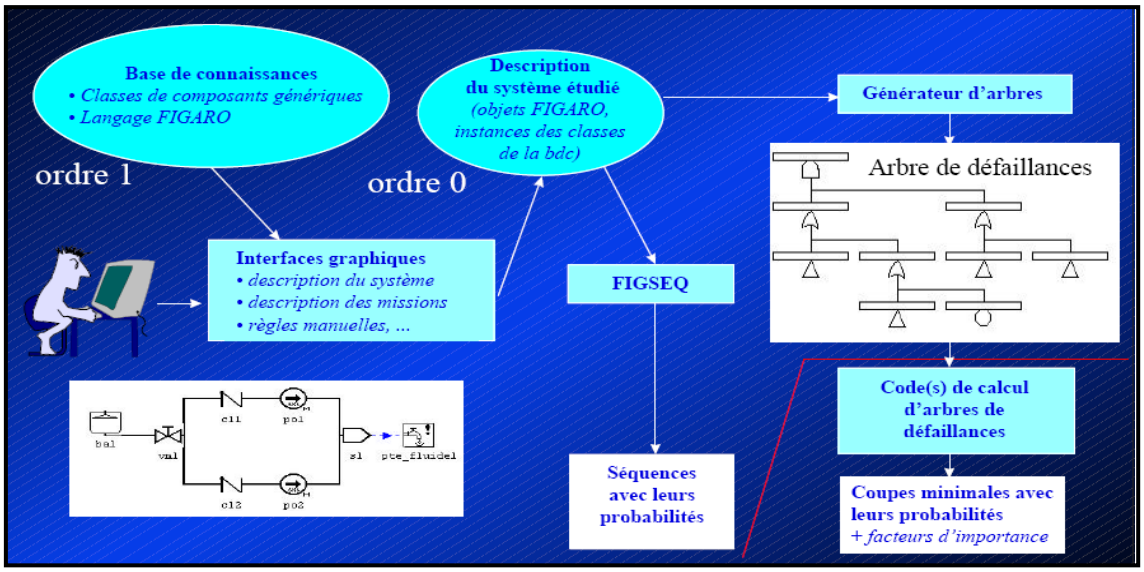


Figure.II.1. Atelier KB3 (Bouissou, 2000)

C- Autres Outils

A partir du modèle textuel dit en FIGARO 0, transparent pour l'utilisateur, il est possible de faire différents types de traitements avec les outils de calcul, développés par des tiers (ARALIA, RISK SPECTRUM pour les arbres de défaillances). Ces outils permettent de faire des calculs de fiabilité, disponibilité, productivité, coûts, etc. Ils donnent également des indications qualitatives précieuses pour améliorer les performances des systèmes : coupes minimales ou séquences prépondérantes, facteurs d'importance [site 2].

II.3 Langage AltaRica

II.3.1 Historique (Bouissou *et al.*, 2006; Bernard, 2009)

De novembre 1996 à octobre 1997, une première phase visant à étudier les possibilités de connexion de différents outils a permis de définir un langage de description de systèmes complexes appelé : « AltaRica ». Des expérimentations sur différents cas d'études ont permis de valider ce langage. Cette première phase comprenait la compagnie d'ingénierie IXI , la société pétrolière Elf Aquitaine ainsi que deux laboratoires de Bordeaux : le LaBRI et le Laboratoire d'Analyse des Dysfonctionnements de Système LADS .

De novembre 1997 à octobre 1999, une seconde phase a permis de formaliser la sémantique du langage AltaRica, de définir une méthodologie de modélisation et de réaliser des prototypes d'outil restreignant AltaRica aux domaines finis. Dans cette version, les modèles AltaRica sont constitués d'automates de mode (ou à contraintes) hiérarchisés interfacés quelconques.

L'environnement d'édition et de calcul « Cecilia OCAS » a été ensuite développé par Dassault Aviation au début des années 2000. Cet environnement a été notamment utilisé pour modéliser les commandes de vol électrique du Falcon 7X et générer des arbres de défaillances qui ont contribué au dossier de certification de l'avion. Il s'est avéré que pour développer

leurs modèles, les ingénieurs ont utilisé un fragment d'AltaRica appelé « AltaRica DataFlow ».

Dans ce fragment, les flux sont orientés et ceci permet de tester plus aisément la complétude de chaque description. D'autre part des outils d'évaluation plus optimisés ont été développés pour ce fragment du langage et appliqués non seulement chez Dassault mais aussi validés dans différents projets de recherches.

Parallèlement, différentes extensions d'AltaRica ont été proposées ou font l'objet de travaux de recherches.

II.3.2 Principales caractéristiques

Les caractéristiques du langage AltaRica sont (Bouissou et al., 2006) :

1. Au départ, les développeurs d'AltaRica ont cherché uniquement à décrire des comportements qualitatifs, ne faisant intervenir aucune loi de probabilité. Ils ont avant tout recherché les concepts minimaux permettant une projection aisée et rigoureuse des modèles AltaRica vers les modèles traditionnels de fiabilité (arbre de défaillances, réseaux de Petri stochastique, automate à états finis, etc). Ainsi, ils ont choisi de représenter la dynamique d'un composant par un automate de modes. Dans un automate de modes, des transitions d'automate qui modélisent les enchaînements des modes de fonctionnement et défaillance, cohabitent avec des formules booléennes qui contraignent les valeurs que peuvent prendre les paramètres du système dans chaque mode.

2. Les modèles AltaRica peuvent être réalisés à différentes fins. Ils sont créés à partir de composants génériques élémentaires, que l'on instancie en nombre suffisant et que l'on interconnecte pour créer des modules plus complexes jusqu'à l'obtention du modèle du système complet.

3. AltaRica ayant été défini à partir d'un ensemble minimum de concepts, le langage peut être aisément appris. Aussi, les utilisateurs peuvent aussi bien réutiliser des modèles que les adapter ou en créer de nouveaux, par exemple si le système comprend des composants technologiques nouveaux, avec des modes de défaillances propres.

4. Le langage AltaRica est orienté vers la création de modèles qualitatifs pouvant être enrichis par des informations de nature probabiliste.

5. La sémantique du langage AltaRica équivaut au fonctionnement d'un automate.

6. Un modèle AltaRica est un automate qui génère un graphe des états atteignables à partir d'états initiaux.

II.3.3 Concepts de base (Bouissou et *al.*, 2006; Point, 2000)

A- Composant AltaRica

- Un composant est une machine abstraite dont l'état est modifié par certains phénomènes appelé événements, et peut être équipé de flux (appelés parfois observations) qui l'informent partiellement sur l'état de son environnement. En sens inverse, ces mêmes flux lui permettent de transmettre une information vers l'extérieur. L'ensemble composé des événements et des flux d'un composant est appelé l'*interface* du composant.
  Un composant interagit avec son environnement par le biais de son interface.
- Les *états* d'un composant sont décrits de manière implicite par un ensemble de variables (dites d'état). Un état du composant est alors une évaluation de ces variables.
- La *configuration* est l'association d'un état et d'une évaluation des variables de flux.
- Le modèle AltaRica suppose qu'il existe une relation de dépendance entre l'état du composant et la valeur de ses flux. Cette relation est représentée par une contrainte sur les variables du composant; cette contrainte est appelée l'*assertion* du composant et elle est supposée vérifiée par toutes les configurations.
- D'un point de vue syntaxique, un *événement* d'un composant est simplement un nom (une étiquette dont l'interprétation est laissée à la charge du modélisateur).
  Les événements modifient l'état d'un composant; ces changements d'états sont appelés *transitions*.
- Le modèle AltaRica intègre la notion de *priorité.* Le concept de priorité est un mécanisme bien connu pour l'ordonnancement d'activités dans les systèmes concurrents, ce mécanisme consiste à définir une relation d'ordre sur les actions du système.

B- Noeud AltaRica

Un *noeud* AltaRica est un modèle de sous-système qui intègre et contrôle d'autres sous-systèmes. D'un point de vue interne, un noeud peut être considéré comme l'union d'un contrôleur et d'un ensemble de composants qui interagissent sous la tutelle du contrôleur. Ce dernier a pour rôle de contraindre les interactions des sous-composants du nœud. Un nœud AltaRica peut :

- Observer et contraindre les flux de ses sous-nœuds.
- Contraindre plusieurs événements des sous-nœuds à se produire en même temps : on parle de *synchronisation d'événements*. En l'absence de cette rubrique, les sous-noeuds sont dits indépendants (asynchrones).

C- Bibliothèques AltaRica

Les composants AltaRica peuvent être groupés dans des bibliothèques, pour être repris ou étendus par les utilisateurs.

II.3.4 Outils

A- ARC

Historiquement, le LaBRI (Laboratoire Bordelais de Recherche en Informatique) a développé, d'une part, des outils regroupés sous le nom d' « Altatools » et, d'autre part, un vérificateur de modèles « Mec » dont l'évolution « MecV » accepte directement les modèles AltaRica LaBRI. Depuis 2005, le LaBRI conçoit l'outil AltaRica Checker « ARC » (Bernard, 2009), basé à la fois sur les Altatools et sur MecV, afin de proposer un unique logiciel réunissant tous les services des outils historiques.

Un modèle en AltaRica LaBRI est seulement composé d'une description textuelle. Tout éditeur de texte permet donc d'écrire un modèle en AltaRica LaBRI. L'outil ARC fournit (Bernard, 2009) :

- *Un contrôleur syntaxique* qui vérifie que le code respecte les règles de syntaxe. Cette opération est effectuée automatiquement lors du chargement d'un modèle en AltaRica LaBRI dans l'outil ARC.
- *Un contrôleur de cohérence* qui permet de s'assurer que toutes les informations nécessaires à la description d'un noeud (domaines, sous-nœuds, etc.) sont spécifiées et que le comportement décrit respecte les exigences du langage. Cette opération est effectuée automatiquement lors du chargement d'un modèle en AltaRica LaBRI dans les outils MecV et ARC.
- Un *simulateur interactif* qui permet, à partir d'une situation initiale décrite dans le code AltaRica ou définie par l'utilisateur et mémorisée dans un outil, de sélectionner manuellement les actions que doit effectuer le système et d'observer son évolution. Cet outil permet de tester le comportement du modèle sur un ensemble fini d'exécutions d'événements afin de s'assurer que le comportement modélisé est fidèle à celui du système réel. Dans l'outil ARC, le simulateur présente respectivement sous forme de tableaux : la configuration courante, les événements possibles et le détail de la transition de l'événement sélectionné. Observer l'évolution du système revient à observer les changements de valeur des différentes variables.

B- Autres outils

Divers outils traitent les modèles AltaRica et permettent l'édition, la génération et l'analyse d'arbres de défaillance, de diagrammes de décision, la génération de séquences d'événements, la simulation graphique, le model checking, l'analyse stochastique, la compilation en modèles de plus bas niveau (Réseau de Pétri), etc… (Mareschal, 2004).On peut citer (Mareschal, 2004; Bernard, 2009), [site 3] :

- *Dassault Aviation a développé l'Outil de Conception et d'Analyse Système (OCAS) :* interface de conception graphique, simulateur graphique, générateurs de modèles (arbres de défaillance, modèles stochastiques), générateur de séquences, …
- *Arboost Technologies a développé les outils Combava (anciennement Toolbox) :* génération d'arbre de défaillance, simulateur graphique, analyseur Markov, simulateur stochastique, et générateur de séquences.

- *EADS APSYS a développé l'atelier SIMFIA :* génération automatique des tableaux d'AMDEC, obtention des diagrammes de fiabilité, génération automatique d'arbre de défaillance, et simulateur graphique.
- *Les outils Externes :* ARALIA (analyse des arbres de défaillance), SMV (model checking), Moca-RP (Réseau de Pétri), UPPAAL (vérification de système temps réel).

II.4 Langage AADL

II.4.1 Historique

Les « ADL's » (Architecture Design Langage) sont une famille de langages élaborés dans les années 90 dans le domaine de l'ingénierie du logiciel. Ils ont en commun une capacité à décrire l'architecture des systèmes mais diffèrent largement quant à leurs motivations et leurs formalismes d'écriture. Ils sont en général graphiques et textuels et possèdent des outils associés (Mareschal, 2004). Les ADL's peuvent être classé comme suit (Vergaud, 2005) :

- ADL formels : pour formaliser la description du fonctionnement d'un système ils ne son pas destinés à s'intégrer dans une démarche de génération automatique. Exemple : Wright, Rapide.
- ADL restreints : pour décrire l'assemblage de composants logiciels ou la cohérence de l'application. Exemple : ArchJava, Fractal.
- ADL concrets : pour décrire l'architecture afin de la générer automatiquement. Eexemple : UML, AADL.

AADL « Architecture Analysis and Design Language » est un langage de description d'architecture, standardisé par la SAE (Society of Automotive Engineers). AADL était destiné à ses débuts aux systèmes avioniques et spatiaux et a fait l'objet d'un intérêt croissant dans l'industrie des systèmes critiques (comme Honeywell, Rockwell Collins, l'Agence Spatiale Européenne, Astrium, Airbus), ce qui explique son ancien nom (Avionics Architecture Description Language). Il s'est avéré par la suite que le langage était très riche et qu'il offre des capacités d'expression qui dépassent largement le domaine de l'avionique et sont

extensibles à d'autres systèmes logiciels et matériels. La première version du standard AADL 1.0 a été publiée en Octobre 2004 (Zalila, 2008; Rugina, 2007).

Une annexe au standard AADL a été définie « *AADL Error Model Annex* » pour compléter les capacités de description du langage de base. Cette annexe représente un sous-langage qui sert à décrire les caractéristiques du système modélisé en AADL liées à la sûreté de fonctionnement (Rugina, 2007).

Parmi les contributions concernant la sureté de fonctionnement, on cite celle de (Joshi *et al.* 2007 ; Rugina, 2007). Elle présente un outil interne de Honeywell qui permet la génération d'arbres de fautes à partir de modèles AADL enrichis avec des éléments de l'annexe des modèles d'erreur (*AADL Error Model Annex*).

Dans ce contexte, (Rugina, 2007) propose une approche itérative pour la modélisation de la sûreté de fonctionnement, basée sur l'utilisation d'AADL.

En 2008, l'équipe de X.Dumas (Dumas *et al.,* 2008), propose des techniques de génération automatique de modèles de sûreté de fonctionnement à partir de spécifications exprimées sous forme de modèle AADL. L'algorithme générique proposé a été implanté par un code de transformation de modèles AADL en modèles AltaRica.

II.4.2 Principales caractéristiques

Les caractéristiques de AADL sont (Dumas *et al.,* 2008; Rugina, 2006; Rugina, 2007) :

1. AADL permet de faciliter la conception et l'analyse de systèmes complexes, critiques, temps réel dans des domaines comme l'avionique, l'automobile et le spatial.
2. AADL fournit une notation textuelle et graphique standardisée pour décrire des architectures matérielles et logicielles et pour effectuer différentes analyses du comportement et des performances du système modélisé.
3. Le succès de AADL dans l'industrie est justifié par sa sémantique précise, son support avancé à la fois pour la modélisation d'architectures reconfigurables et pour la conduite d'analyses.
4. Le langage a été conçu pour être extensible afin de permettre des analyses qui ne sont pas réalisables avec le langage de base.

6. Une spécification d'architecture AADL contenant des modèles d'erreur fournit une vue orientée vers la sûreté de fonctionnement et peut être utilisée pour une variété de méthodes d'analyse. Des modèles classiques de sûreté de fonctionnement, comme les arbres de fautes, les chaînes de Markov et les réseaux de Petri stochastiques généralisés (GSPN) peuvent être générés.

7. Les modèles d'erreurs représentent des automates décrivant des comportements en présence de fautes.

II.4.3 Concepts de base

A- Composants (Rugina, 2007; Zalila, 2008)

- Une description AADL est faite de composants. Un composant AADL, selon le standard AADL, représente une entité matérielle ou logicielle qui appartient au système en cours de modélisation.
- Un composant possède un *type*, qui décrit son interface. Ce type joue le rôle d'une spécification pour le composant. Cette spécification est utilisée par les autres composants du système pour interagir avec le composant spécifié. Le type d'un composant AADL est constitué de trois parties : les *éléments d'interface*, les *flots* et les *propriétés*.
- Une ou plusieurs *implémentations* peuvent être associées au même type, correspondant à différentes structures du composant en termes de sous composants, connexions (entre les ports des sous composants), appels de sous-programmes et modes opérationnels.
- Contrairement aux autres composants, les composants systèmes, ne représentent pas une entité concrète. Ils sont utilisés pour créer des blocs qui aident à structurer l'application.
- L'*interface* d'un composant fournit des éléments d'interface (les ports de communication, les paramètres, etc.). Les composants communiquent les uns avec les autres en connectant leurs éléments d'interface respectifs. Un élément d'interface modélise une caractéristique qui est visible par les autres composants. En AADL, ces éléments sont des entités nommées permettant à un composant d'échanger les données et les signaux avec l'extérieur.

B- Annexes et Propriétés (Rugina, 2007; Zalila, 2008)

Pour enrichir les descriptions AADL avec les caractéristiques non architecturales, le langage offre deux mécanismes : les annexes et les propriétés.

- Les *annexes* dans les modèles AADL permettent d'incorporer dans le modèle d'une application, des éléments écrits dans un langage différent de AADL (Zalila, 2008).Le « AADL Error Model Annex » offre des primitives permettant de décrire le comportement du système en présence de fautes (fautes, modes de défaillance, propagations d'erreurs, hypothèses de maintenance si on s'intéresse à la disponibilité).
- AADL introduit la notion de *propriété*. Les propriétés sont des caractéristiques associées à différentes entités (composants, connections, éléments d'interface, etc.).Il s'agit d'attributs qui permettent de spécifier des caractéristiques ou des contraintes s'appliquant aux éléments de l'architecture.

C- Paquetage et ensembles de propriétés (Zalila, 2008)

Pour mieux organiser les déclarations dans un modèle AADL, les notions de paquetage et d'ensemble de propriétés on été introduites.

- Les paquetages (package) servent à organiser les déclarations des composants dans des bibliothèques de composants réutilisables.
- Les ensembles de propriétés (property set) sont les équivalents des paquetages pour les déclarations de propriétés en AADL. Ils permettent de rassembler des définitions de types, de constantes et de propriétés utilisées dans les modèles.

D- Modes Opérationnels (Zalila, 2008)

Les *modes opérationnels* représentent les états des composants AADL (toutes familles confondues). Ils permettent de contrôler les valeurs des propriétés, l'activation des connexions et même des sous-composants.

E- Composant hiérarchique (Hugues, 2009)

Pour modéliser une architecture, les composants AADL doivent être déclarés comme sous-composants (*instances*) d'autres composants AADL. Au niveau le plus haut, un composant *system* contient toutes les instances des autres composants. La plupart des composants AADL peuvent contenir des sous-composants permettant ainsi une description hiérarchique du système.

F- AADL architectural et AADL de sûreté de fonctionnement (Rugina, 2007)

AADL fournit un support pour définir des configurations architecturales et le passage entre les modes opérationnels. La dynamique des modes opérationnels influence les mesures de sûreté de fonctionnement comme la disponibilité (§ I.2.4). Par conséquent, ils doivent être pris en compte dans le modèle de sûreté de fonctionnement.

- Le modèle défini par l'ensemble des composants du système et des modes opérationnels est appelé : *modèle AADL architectural*.
- Un *modèle de sûreté de fonctionnement* est un modèle contenant des informations telles que modes de défaillance, politiques de réparation et propagations d'erreur. Un modèle AADL de sûreté de fonctionnement est un modèle architectural annoté avec des modèles d'erreur qui contiennent ces informations.

II.4.4 Outils

Différents outils proposés ces dernières années exploitent un aspect de l'architecture AADL, ces outils permettent :

- D'éditer des modèles AADL (outils Stood, ADELE, OSATE, TOPCASED,etc).
- De générer le logiciel du système à partir d'un modèle AADL (Stood, Ocarina,etc).
- De conduire diverses analyses (OSATE, Versa/- Furness, ADAPT, Cheddar,etc).

A- Ocarina

L'outil Ocarina (Vergaud, 2006) est une application autonome utilisable en ligne de commande permettant de produire du code exécutable, un réseau de Petri, de passer d'une représentation AADL à une autre ou simplement de vérifier la validité d'une description AADL. Tous les modules ainsi que le coeur d'Ocarina (**Figure.II.2.**) peuvent également être utilisés comme des bibliothèques logicielles, intégrées dans une application tierce afin de permettre à des outils existants de manipuler une description AADL.

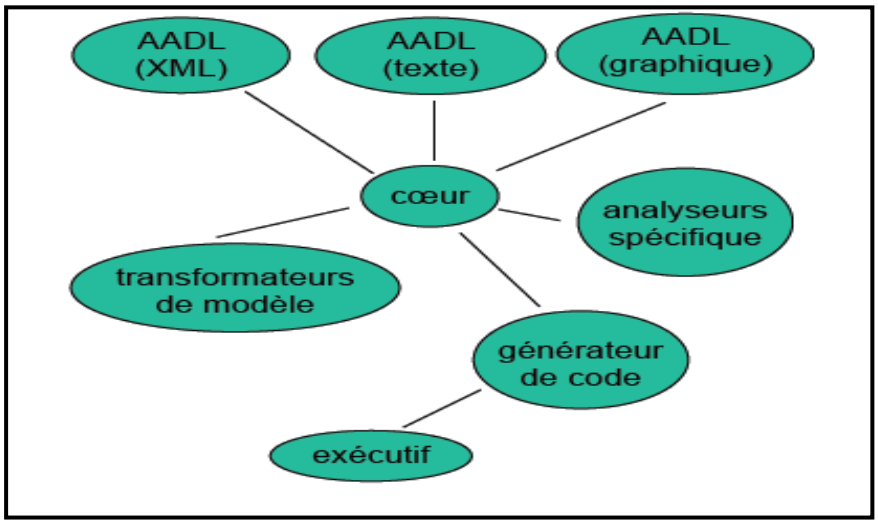


Figure.II.2. Atelier Ocarina (Vergaud, 2005)

B- Autres outils (Idiri, 2009)

- *Osate* est un outil de modélisation textuelle d'AADL en environnement Eclipse, développé par le SEI (Software Engineering Institute). OSATE permet de faire la vérification syntaxique et sémantique des modèles AADL. Il est indispensable pour commencer à faire de l'AADL (Idiri, 2009).
- *Topcased* est un outil de modélisation graphique d'AADL en environnement Eclipse, développé par le projet Topcased piloté par Airbus. Cet outil est fondé sur OSATE et le complète.

- *Stood* est une suite d'outils de modélisation des systèmes temps réel basée sur la méthode HOOD. Il contient un outil graphique (qui supporte la syntaxe du standard AADL 1.0, UML) et un générateur de code Ada et C.
- *Cheddar* est un outil de simulation permettant de calculer différents critères de performance (contraintes temporelles, dimensionnement de ressources). L'outil permet, entre autres, de tester le respect des contraintes temporelles d'un jeu de tâches modélisant une application/un système temps réel.

II.5 Autres langages

II.5.1 SyRelAn

L'environnement « SyRelAn » (Bernard, 2009), est un logiciel fortement graphique permettant de réaliser un modèle sans nécessiter l'apprentissage d'un langage. D'un point de vue sûreté de fonctionnement, la seule défaillance prise en compte est la perte du composant, les cas de comportement erroné sont exclus. Des calculs de performance comparables à ceux réalisés par SyRelAn sont proposés par les outils industriels traitant d'autres modèles écrits en AltaRica data-flow. Les possibilités de modélisation offertes par l'environnement SyRelAn sont moins importantes que celles offertes par les langages et les outils AltaRica.

II.5.2 UML

UML (Unified Modeling Language) fait partie des langages de description d'architectures de systèmes. Parmi les travaux portant sur ce langage, le projet européen HIDE propose une méthode pour analyser et évaluer automatiquement la sûreté de fonctionnement à partir de modèles UML. Une transformation de modèle a été définie à partir de diagrammes UML structurels et comportementaux vers des RdPSG (Réseaux de Petri Stochastiques

Généralisés), pour l'évaluation de la sûreté de fonctionnement. D'autre part, d'autres travaux on permit l'obtention d'arbres de fautes à partir d'UML (Rugina, 2007).

II.5.3 EAST-ADL2

« EAST-ADL2 » (Bernard, 2009) est un langage dédié à la description d'architectures de systèmes embarqués pour les automobiles. L'approche proposée considère un méta-modèle en EAST-ADL2 composé d'un modèle d'erreur et d'un modèle fonctionnel d'architecture, réalisé selon AUTOSAR (Automotive Open System Architecture). Tout comme AltaRica, et Figaro l'approche EAST-ADL2 considère les défaillances de composants et la propagation de ces défaillances, cependant le comportement dysfonctionnel est séparé du comportement nominal.

II.6 Conclusion

Dans ce chapitre nous avons étudié les langages de sûreté de fonctionnement FIGARO, AltaRica, et AADL, et nous avons fait un tour d'horizon sur d'autres langages de SdF. Après cette étude nous avons procédé à la comparaison des langages FIGARO, AltaRica, et AADL et l'élaboration d'une démarche pour la génération et l'analyse des modèles classiques de SdF comme présenté dans le chapitre suivant.

III DEMARCHE POUR LA MODELISATION ORIENTEE LANGAGE

III.1 Introduction

Dans le chapitre I, nous avons présenté les concepts de base ainsi que les méthodes de SdF; ensuite dans le chapitre II, nous avons étudié les langages dédiés à la sûreté de fonctionnement des systèmes. Dans ce chapitre, nous présentons notre démarche, pour la modélisation orientée langage pour la SdF des systèmes. Nous proposons cette démarche après avoir établi une comparaison des langages FIGARO, AltaRica et AADL et mis en évidence les apports, les avantages, et les inconvénients de chaque langage.

III.2 Comparaison des Langages

Nous adoptons cinq niveaux de comparaison des trois langages FIGARO, AltaRica et AADL :

- La définition de chaque langage
- Les concepts de base de chaque langage
- La description de chaque langage
- La représentation du système et l'organisation des composants
- La traduction en d'autres langages.

Chaque niveau est explicité suivant plusieurs paramètres, et le résultat final de la comparaison des trois langages est donné sous forme de tableaux regroupant les paramètres de chaque niveau.

III.2.1 Définition des langages

Le niveau Définition des langages (**Tableau.III.1.**) regroupe les principaux objectifs (Bouissou *et al.*, 2006; Hugues *et al.*, 2009; Rugina, 2007; Vergaud, 2005), et les principales caractéristiques (Bernard, 2009; Bouissou *et al.*, 2006; Hugues *et al.*, 2009; Rugina, 2006; Rugina, 2007) des langages FIGARO, AltatRica et AADL.

	FIGARO	AltaRica	AADL
Principaux objectifs	langage pour la SdF des systèmes		Langage de description d'architecture
Principales caractéristiques	permet d'écrire des modèles probabilistes quantifiables pouvant être simplifiés en modèles qualitatifs	orienté vers la création de modèles qualitatifs pouvant être enrichis par des informations de nature probabiliste	associé à l'annexe de modèle d'erreur ; AADL permet l'analyse qualitative pouvant être enrichie par des informations de nature probabiliste
	la déclaration des types et leurs instanciations dans deux niveaux de langage	la déclaration des types et leurs instanciations dans un seul niveau de langage	la déclaration des types, leurs implémentations et leurs instanciations dans un seul niveau de langage
	possède deux niveaux : l'ordre 1 (déclaration d'objets) et l'ordre 0 (instanciation d'objets)	plusieurs variantes du langage: AltaRica LaBRI, AltaRica Data-Flow	Le langage AADL est associé à différentes annexes (annexe de traduction, annexe comportemental, etc.)

Tableau.III.1. Définition des langages FIGARO, AltaRica et AADL

III.2.2 Concepts de base

Les points essentiels et spécifiques de chaque langage FIGARO, AltaRica et AADL sont résumés dans le tableau (**Tableau.III.2.)**, décrivant les concepts de base (Bernard 2009; Bouissou *et al.*, 2006; Point, 2000; Rugina, 2007; Zalila, 2008).

	FIGARO	AltaRica	AADL
Concepts de base	un seul type d'objets	un seul type de composants	3 familles de composants : matériel, logiciel, hybride
	un type d'objet est définit par : ses variables (attribut, constante, effet, panne), les interfaces, les règles d'interactions et les règles d'occurrences	un composant est défini par son état interne, son interface, les assertions et les transitions	un type d'un composant AADL est constitué de trois parties : les éléments d'interface, les flots et les propriétés
	les règles d'occurrences définissent le changement d'état de l'objet	les transitions définissent le changement d'état du composant.	les transitions (définies dans l'implémentation des composants du modèle d'erreur définissent le changement d'état du composant.
	permet héritage, mais pas l'organisation hiérarchique entre les objets	Permet l'assemblage hiérarchique des composants sans conditions, mais pas d'héritage entre les composants	permet héritage, et l'organisation hiérarchique entre les composants.
	un objet (composant) interagit avec son environnement par le biais de son interface, le mode d'échange entre composants de AADL ressemble fortement a celui de AltaRica		
	les interfaces d'un objet permettent de spécifier les types d'objets qui sont en relation.	l'interface d'un composant est composée des événements et des flux (variables d'entrée et de sortie) d'un composant	Les composants communiquent les uns avec les autres en connectant leurs éléments d'interface respectifs les éléments d'interface regroupent les ports, les paramètres, les sous programmes, et les accès aux sous composants

Tableau.III.2. Concepts de base des langages FIGARO, AltaRica et AADL

III.2.3 Description des langages

La description des langages (**Tableau.III.3.**) définit un niveau syntaxique et sémantique (Bouissou *et al.*, 2006; Hugues *et al.*, 2009) ainsi que la robustesse c'est-à-dire la capacité à

détecter les incohérences d'une part, et d'autre part à aider l'utilisateur à débusquer diverses erreurs.

Pour la variante AltaRica Data-Flow, les flux sont orientés et ceci permet de tester plus aisément la complétude de chaque description; des outils d'analyse plus optimisés ont été développés pour ce fragment du langage (Bouissou *et al.*, 2006).

<table>
<tr><th></th><th>FIGARO</th><th>AltaRica</th><th>AADL</th></tr>
<tr><td rowspan="2">Syntaxe et mots-clés</td><td>la syntaxe est proche du langage naturel</td><td colspan="2">la syntaxe rappelle celle d'un langage de progr-ammation</td></tr>
<tr><td>grand nombre de concepts et mots</td><td>nombre réduit de concepts et de mots-clés</td><td>grand nombre de concepts et mots clefs,</td></tr>
<tr><td>Sémantique</td><td colspan="3">le comportement qualitatif du modèle équivaut au fonctionnement d'un automate qui génère les états atteignables à partir des états initiaux</td></tr>
<tr><td>Robustesse</td><td colspan="3">permet la détection des erreurs syntaxiques et les incohérences sémantiques, la vérification formelle, et les études de sûreté de fonctionnement du modèle avec différentes analyses et divers outils (à condition de se limiter à la variante AltaRica Data-Flow pour le langage AltaRica)</td></tr>
</table>

Tableau.III.3. Description des langages FIGARO, AltaRica et AADL

III.2.4 Représentation du système

Le niveau Représentation du système (**Tableau.III.4.**) décrit la puissance de modélisation des langages (Bouissou *et al.*, 2006), la réutilisation des composants au sein des langages (Bouissou *et al.*, 2006; Zalila, 2008), ainsi que l'existence d'une représentation graphique (Bouissou *et al.*, 2006; Loiret, 2008;Vergaud, 2005) des langages FIGARO, AltatRica et AADL.

	FIGARO	AltaRica	AADL
Puissance de modélisation	le système est étudié dans sa globalité	Le système est étudié avec plusieurs niveaux hiérarchiques	
	l'héritage permet de réduire le nombre de déclarations de caractéristiques dans les types, mais on a plus de types	pas d'héritage donc répétition des caractéristiques communes entre les types, mais moins de déclarations de types	l'héritage permet de réduire le nombre de déclarations de caractéristiques dans les types, mais on a plus de types
	les interactions complexes entre les objets sont faciles à modéliser (accès direct)	les interactions complexes entre les composants sont plus difficiles à modéliser il faut passer par la hiérarchie	
Réutilisation composants	les objets réutilisables se trouvent dans une base de connaissance écrite en Figaro1	les composants réutilisables se trouvent dans les bibliothèques de composants	
	la généralisation et l'héritage favorisent la réutilisation des objets Figaro	l'encapsulation favorise la réutilisation des composants	l'encapsulation et l'extension favorisent la réutilisation des composants
Représentation graphique	l'interface graphique permet de représenter le modèle physique du système à partir du modèle du langage		

Tableau III.4. Représentation du système et Organisation des composants

III.2.5 Traduction des langages

Le niveau de traduction (**Tableau.III.5.**) des langages concerne l'aptitude à la traduction des langages FIGARO, AltaRica, et AADL en d'autres langages (Bouissou *et al.*, 2006; Dumas *et al.*, 2008; Hugues *et al.*, 2009) et le type de résultats que l'on peut obtenir avec les trois langages (Bouissou *et al.*, 2006; Bozzano *et al.*, 2010; Dumas *et al*., 2008; Hladik *et al.*, 2010; Rugina, 2006; Rugina, 2007).

<table>
<tr><th></th><th>FIGARO</th><th>AltaRica</th><th>AADL</th></tr>
<tr><td rowspan="3">Aptitude à la traduction</td><td>la traduction en langage AltaRica suppose le respect de certaines restrictions par le modèle FIGARO de départ, .il n’est pas possible par exemple de traduire des modèles Figaros avec des boucles, et des modèles comportementaux probabilistes complexes en modèles Altarica</td><td>Il est possible de traduire automatiquement tout modèle AltaRica en modèle FIGARO</td><td>des travaux récents ont permis de transformer les modèles AADL en modèles AltaRica</td></tr>
<tr><td>pas de traduction en langage de programmation</td><td colspan="2">permet la traduction vers des langages de programmation tels que C ou Ada</td></tr>
<tr><td>difficile à traduire vers d’autres modèles de vérification</td><td colspan="2">peut être mis en relation avec des modèles de vérification tels que Esterel ou Lustre.</td></tr>
<tr><td rowspan="2">Résultats obtenus</td><td colspan="3">- avec les outils qui ont été spécifiquement développés pour interpréter les langages AltaRica (la version DataFlow), FIGARO, et AADL il est possible à partir de ces types de modèles, de faire les traitements suivants :
- génération d’AdD (sous réserve que le modèle satisfasse certaines conditions d'indépendance des événements associés aux pannes)
- simulation interactive, simulation Monte-Carlo
- génération de graphe de tous les états atteignables à partir de l’état initial
- quantification probabiliste du graphe si toutes ses transitions correspondent à des événements associés à des lois exponentielles ou instantanées
- recherches et éventuellement quantification de séquences menant a des états possédant certaines caractéristiques, etc.</td></tr>
<tr><td>difficile à traduire vers un modèle acceptable par un model-checker, mais cela reste envisageable.</td><td colspan="2">il est relativement aisé de faire du model-checking à partir de modèles AltaRica et AADL, soit directement avec des outils dédiés à ces langages, soit après traduction automatique.</td></tr>
</table>

Tableau.III.5. Traduction des langages FIGARO, AltaRica et AADL

III.3 Démarche de modélisation proposée

III.3.1 Motivations pour le choix des langages

La comparaison des trois langages nous a permis de faire les constatations principales suivantes:

1. Les langages FIGARO et AltaRica sont des langages de sûreté de fonctionnement conçus initialement pour généraliser les modèles classiques de SdF et faciliter l'analyse des systèmes complexes tels que les systèmes industriels, les systèmes à haute technologie, ainsi que les processus industriels à risques. Par contre, AADL est un langage extensible de modélisation, de conception et d'analyse d'architectures de systèmes embarqués temps-réel. Cependant, le langage AADL a une grande aptitude à l'extension et l'interprétation en d'autres langages, ce qui lui permet d'être traduit en modèles d'évaluation de SdF ou d'être interprété en AltaRica.

2. Des trois langages, AltaRica est le plus abstrait et le plus général dans la description des systèmes. Le petit nombre de concepts et de mots-clés définis en AltaRica fait que son apprentissage est beaucoup plus rapide que celui de FIGARO et AADL. Ceci dit, le langage FIGARO ressemble plus au langage naturel, et donne l'impression que le concepteur écrit un algorithme plutôt qu'un programme donc ceci a aussi son intérêt.

3. On ne peut pas dire que la décomposition hiérarchique d'AltaRica facilite la conception et l'analyse des systèmes, il peut être plus avantageux dans une étude d'analyser le système dans sa globalité avec le langage FIGARO, et effectuer par la suite une décomposition en sous systèmes et composants avec le langage AltaRica.

4. Du coté de la modélisation des interactions entre composants, chaque langage l'appréhende de façon différente mais chaque façon a ses avantages et ses inconvénients. Pour AltaRica c'est facile de modéliser la propagation des effets entre composants; la propagation est définie au niveau du nœud supérieur c'est lui qui joue le rôle de lien entre composants en connectant leurs variables de flux (assertions). Dans le modèle FIGARO, la propagation peut être réalisée au niveau des nœuds ou des liens entre les composants, il suffit de déclarer les règles d'interactions et les interfaces pour avoir accès à la donnée voulue. La modélisation des échanges de données est plus facile avec AltaRica, mais la modélisation des échanges

complexes est plus facile avec FIGARO vue qu'il y a un accès direct à la donné via l'interface au lieu de passer par la hiérarchie.

5. En ce qui concerne les modèles classiques de SdF, les deux langages FIGARO et AltaRica sont aptes à la traduction vers ces modèles et les deux langages permettent une étude qualitative et quantitative du système. Il est difficile de faire du model-cheking à partir de modèle FIGARO, aussi l'aptitude de traduction du langage en d'autres modèles, n'est pas aussi facile que celle d'AltaRica mais il existe des moyens et des contraintes pour que cela reste envisageable.

6. Le langage FIGARO est plus stable syntaxiquement que le langage AltaRica, qui est en permanente évolution, ce qui a permis d'aboutir à des outils performants et des bases de connaissances importantes dans divers domaines.

7. Il n'est pas possible de dire qu'un des deux langages est plus complexe que l'autre, ou bien plus intéressant : chaque langage a ses défauts et ses avantages. Aussi, il n'est pas intéressant d'unifier ces deux langages différents de par leurs sémantiques.

8. L'analyste de SdF a à sa disposition divers langages et outils dédiés à ces langages; il lui est possible de choisir un langage selon ses attentes. S'il a comme objectif des études probabilistes complexes dans ce cas il peut s'orienter vers le langage FIGARO, et si son objectif et une analyse qualitative pouvant être complétée par une analyse quantitative simple, il peut s'orienter vers un des deux modèles; ceci dit l'utilisateur doit être conscient des avantages et des limites de chaque langage.

III.3.2 Méta-modèle de la démarche

A l'issue de cette comparaison, nous retenons les deux langages FIGARO et AltaRica étant donné que le langage AADL n'offre aucune vision supplémentaire dans une analyse de SdF par rapport à ces deux langages.

Dans ce contexte, nous proposons dans cette section du chapitre une démarche qui permet de tirer profit des apports des deux langages retenus et des outils qui leurs sont dédiés. Nous proposons le méta-modèle suivant :

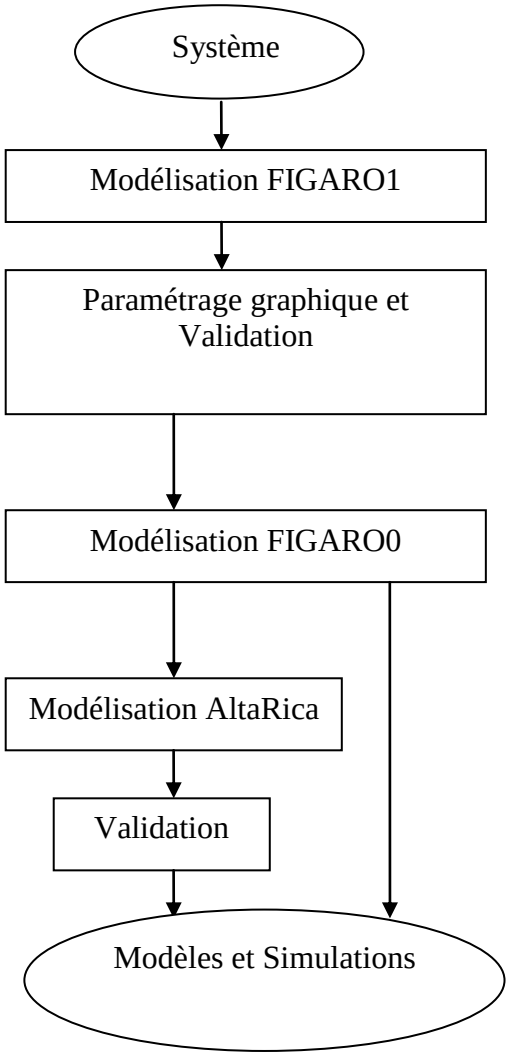


Figure.III.1. Démarche proposée

III.4 Description des modules de modélisation

III.4.1 Modélisation FIGARO1

La modélisation du système avec FIGARO1 (§ II.2.3) passe par les étapes suivantes :

- Détermination des types et de leurs interfaces
- Définition des règles

A- Détermination des types et de leurs interfaces

Les types sont très simplement définis comme représentation directe des composants présents dans le système, auxquels on ajoute d'autres types afin d'y déclarer les caractéristiques communes. La figure suivante représente la déclaration des types avec VisualFigaro (§II.2.4.1) :

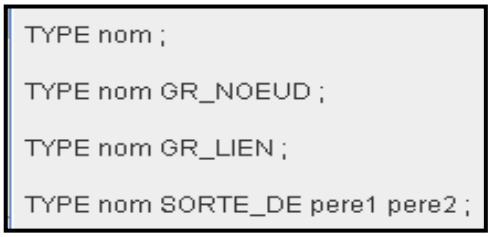


Figure.III.2. Déclaration d'un type

Un type est décrit par des variables telles que : attribut, constante, effet et panne.

Exemple :

```
TYPE noeud ;
 CONSTANTE
    fonction DOMAINE 'source' 'but' 'intermediaire'
    PAR_DEFAUT 'intermediaire' ;
    lambda DOMAINE REEL PAR_DEFAUT 1e-5 ROLE CONCEPTION ;
    mu DOMAINE REEL PAR_DEFAUT 0.1 ROLE CONCEPTION ;
 EFFET
    relie ;
    PANNE def ;
```

Les interfaces servent à représenter les connexions entre les composants et permettent ainsi de les faire interagir (Torrente *et al.,* 2008). La figure suivante représente la déclaration des interfaces avec VisualFigaro :

```
INTERFACE nom GENRE nom_de_type ;
INTERFACE nom GENRE nom_de_type CARDINAL 1 ;
```

Figure.III.3. Déclaration d'une interface

Exemple :

```
TYPE arete_uni_dir SORTE_DE lien ;
INTERFACE
     depart GENRE noeud CARDINAL 1;
     arrivee GENRE noeud CARDINAL 1 ;
```

B- Définition des règles

Les composants sont maintenant des boîtes vides, que l'on peut accrocher les unes aux autres, à condition de respecter certaines contraintes. Pour les faire interagir il faut écrire des règles en langage FIGARO1, sans oublier de déclarer les constantes et les variables qu'elles manipulent. Les règles d'interactions permettent de propager les conséquences des modifications qui ont été effectuées sur un composant à la suite de l'occurrence d'un événement (règles d'occurrences) à tous les autres composants entraînant ainsi l'évolution du système vers un nouvel état (Torrente *et al.,* 2008). L'édition des règles dans VisualFigaro est présentée dans les figures (**Figure.III.4** et **Figure.III.5)** :

```
OCCURRENCE nom_regle IL_PEUT_SE_PRODUIRE DEFAILLANCE nom_panne LOI EXP(Expression) ;
```

Figure.III.4. Edition d'une règle d'occurrence

```
INTERACTION SI Condition ALORS ListeActions ;
```

Figure.III.5. Edition d'une règle d'interaction

Remarques :

- Les règles peuvent être groupées dans un ensemble de règles spécifiques à un traitement (simulation interactive, génération d'arbre de défaillances, etc.), ou bien en un ensemble de règles spécifiques à une étape.
- Nous utilisons l'outil VisualFigaro (version 1.14 sous Windows) comme outil support au langage.

III.4.2 Modélisation FIGARO0

La génération du modèle FIGARO0 (§ II.2.3) (instantiation des types et remplissage des interfaces) se fait automatiquement par l'outil KB3 (§ II.2.4) à partir du shéma physique du système, du modèle FIGARO1, et de son parametrage graphique (§ III.5.1) afin de fournir une interface vers le modèle AltaRica, et vers les outils de génération de modèles classiques (comme par exemple, les arbres de défaillances), et les outils de traitements.

III.4.3 Modélisation AltaRica

La modélisation AltaRica (§ II.3) donne la description d'un nœud. Le code de tout nœud AltaRica (Bernard, 2009) est encapsulé par les mots-clés « node » et « edon ». Un nœud simple est composé de :

- *Variables :*
  - *Variables d'états :* champs « state » et « init » (pour préciser la valeur initiale de chaque variable).
  - *Variables de flux :* champ « flow ».
- *Evénements :* champ « event ».
- *Règles :*
  - *Règles de transitions :* champ « trans ».
  - *Règles d'assertions :* champ « assert ».
- *Informations externes : champ « extern » (déclaration d'informations externes à la sémantique d'AltaRica mais utile pour certains outils complémentaires).*

Le langage étant hiérarchique, un nœud peut contenir des sous-noeuds. Il est alors possible de déclarer, outre les informations citées ci-dessus :

- *Les sous-noeuds contenus :* champ « sub ».
- *Les synchronisations d'événements :* champ « syn ».

Nous proposons le tableau suivant qui illustre le passage du modèle FIGARO0 vers le modèle AltaRica.

FIGARO0	**AltaRica**
objets (nœuds graphiques)	composants
attributs, et pannes	variables d'état
effets, et interfaces	variables de flux
règles d'interactions	assertions
règles d'occurrences	transitions et événements
liens entre objets (nœuds graphiques)	nœuds hiérarchiques

Tableau.III.6. Passage du modèle FIGARO0 vers le modèle AltaRica

III.5 Validation des modèles et résultats

III.5.1 Paramétrage graphique et validation du modèle FIGARO1

A- Paramétrage graphique (Bouissou, 2010; Torrente *et al.*, 2008)

Le paramétrage de la partie graphique (schéma physique du système) avec l'outil VisualFigaro passe par les étapes suivantes :

a- Affectation de la catégorie graphique et des icônes aux types FIGARO. Chaque type peut être vu comme un lien ou un noeud dans le graphe représentant le système. Si un type est considéré comme un lien dans le graphe il prendra la forme d'une flèche. Chaque nœud ou lien est représenté dans le graphe par une petite image appelée *icône.*

b- Définition des points de connexions. Les points de connections (**Figure.III.6.**) permettent de définir des interfaces graphiques « intelligentes » capables de connecter (ou refuser de connecter) un lien à un nœud; la connexion peut engendrer le remplissage des interfaces du nœud ou du lien par le nœud ou le lien connecté.

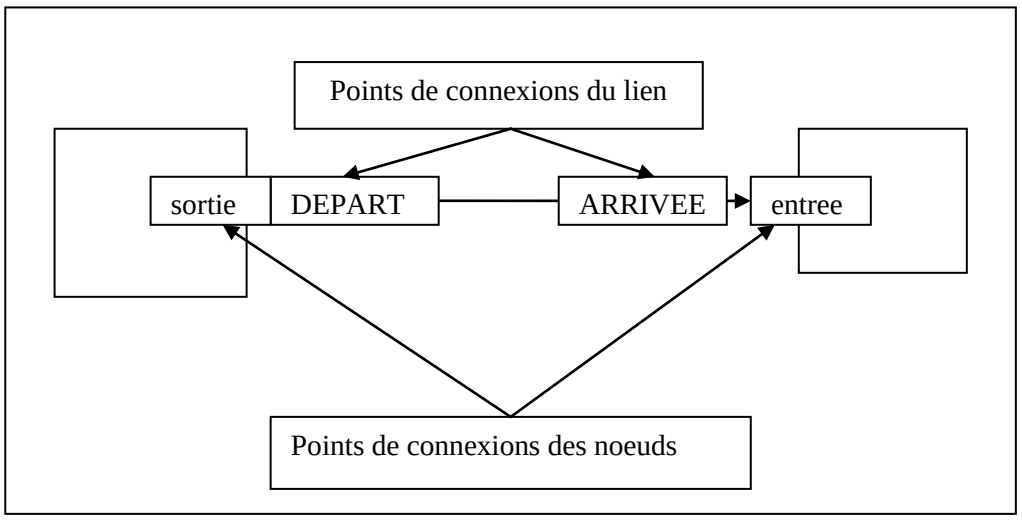


Figure.III.6. Points de connexions

c- Définition des visualisations. La visualisation permet de rendre expressif le schéma du système, et elle est utilisée aussi au cours de la simulation interactive pour voir le comportement du système sur son schéma. Il est possible de visualiser sur le schéma du système l'état des composants (valeurs de certaines variables) grâce aux variantes graphiques.

d- Définition des variantes graphiques. L'icône d'un composant peut changer de couleur (comme par exemple, elle devient rouge si le composant est défaillant), ou bien la valeur d'une de ses variables peut s'afficher sur son icône dans le schéma du système, on appelle ça des variantes graphiques.

B- Validation du modèle FIGARO1

a- Vérification syntaxique : La vérification syntaxique du modèle FIGARO1 est effectuée pendant :

- L'écriture du code avec l'outil VisualFigaro.
- Le chargement du modèle FIGARO1 et de son paramétrage graphique avec l'outil KB3.

b- Vérification du paramétrage graphique. A ce stade, le chargement du model FIGARO1 et de son paramétrage graphique avec KB3 permet de s'assurer aussi de l'absence de toute incohérence dans ce qui a été défini jusque là, et de vérifier par la saisie de petits assemblages de composants que les contraintes de connections et les règles de remplissage des interfaces fonctionnent comme on le souhaite (Torrente *et al.*, 2008).

c- Vérification sémantique : Le modèle FIGARO1 est validé après une série de tests sur des systèmes variés (grâce au simulateur interactif et à la visualisation sur le schéma physique du système) avec l'outil KB3. Le comportement du système modélisé doit correspondre au comportement du système réel.

Remarque : Nous utilisons l'outil KB3 (version 3.0 sous Windows) [site 4]

III.5.2 Validation du modèle AltaRICA

a- Vérification syntaxique : Le chargement du modèle AltaRica avec l'outil ARC (§ II.3.4) permet de s'assurer de l'absence des erreurs syntaxiques dans le modèle.

b- Vérification sémantique : Le modèle Altarica est validé après la simulation du comportement du système avec le simulateur interactif intégré à l'outil ARC, qui doit correspondre au comportement du système réel.

Remarque : Nous utilisons l'outil ARC (version 1.3.5 sous Linux) [site 5].

III.5.3 Génération de modèles et simulations

Une fois la modélisation du système, avec les deux langages validée, il devient possible d'effectuer des simulations sur ces modèles, et de générer les modèles de SdF tels que les arbres de défaillances, et les séquences d'évènements.

A- Génération des Arbres de défaillances

L'arbre de défaillances permet d'identifier les combinaisons de défaillances pouvants conduire le système à un état redouté. L'arbre de défaillances généré par l'outil KB3 est défini à partir d'un profil (valeurs initiales des variables) et d'un évènement indesirable, et représente la négation de l'arbre généré par chainage arrière des régles d'interactions instanciées du modèle FIGARO0 (le point de depart du chainage arrière est la négation de l'événement indésirable).

B- Simulations

a- Visualisation statique : La visualisation statique avec KB3, permet de visualiser l'état des composants sur le schéma physique du système à partir d'une configuration initiale (par exemple à partir d'une combinaison de défaillances).

b- Simulations interactives : La simulation interactive consiste à analyser le comportement du système**,** à chaque transition (occurrence d'un événement) sélectionnée par l'utilisateur. La simulation interactive peut se faire avec deux manières différentes grâce aux deux modèles FIGARO0 et AltaRica. Il devient possible d'effectuer deux types de simulations :

- *Simulation interactive avec FIGARO :* Cette simulation permet de visualiser les changements d'états des composants sur le schéma physique du système global.

- *Simulation interactive avec AltaRica :* Cette simulation consiste à analyser le comportement (changements de valeurs des variables d'état et de flux) du système décomposé de façon hiérarchique (système, sous-systèmes, et composants).

C- Génération des séquences d'événements

Avec l'outil ARC, il est possible de générer toutes les séquences d'évènements menant le système à un état redouté.

III.6 Conclusion

Dans ce chapitre, nous avons effectué, une comparaison des langages les plus utilisés en SdF, cette comparaison nous a amené à proposer une démarche rassemblant les plateformes AltaRica et FIGARO pour l'analyse des systèmes. Cette démarche commence par la modélisation du système avec FIGARO, après la validation du modèle on passe à la déduction du modèle AltaRica ainsi que sa validation, et on termine par la génération des modèles, et les simulations. Dans le chapitre suivant nous présentons une mise en œuvre expérimentale de notre démarche sur un système critique.

IV MISE EN ŒUVRE EXPERIMENTALE

IV.1 Introduction

Les sytèmes critiques de détection d'incendie ont fait l'objet d'une étude de sûreté de fonctionnement pour la génération manuelle des arbres de défaillances (Zwingelstein, 2009). Nous proposons d'étudier ce système dans le cadre spécifique de la démarche présentée dans le chapitre précédent, en regroupant les plates-formes FIGARO et AaltaRica, pour modéliser le système de façon globale, déduire ensuite sa modélisation hierarchique, générer automatiquement les arbres de défaillances, simuler le comportement du système face aux défaillances suivant les deux approches, et déduire les séquences d'évènements menant aux états redoutés.

L'étude est composée des étapes suivantes :

1. Présentation du système de détection d'incendie
2. Modélisation suivant FIGARO
3. Modélisation suivant AltaRica
4. Validation des modèles et résultats

IV.2 Le système de détection d'incendie

Les systèmes de détection d'incendie font partie des systèmes critiques dont les défaillances mettent en jeu la sécurité des personnes, des biens, et de l'environnement.

IV.2.1 Mode de fonctionnement

Un système de détection d'incendie dans un atelier de production dont les portes sont fermées comprend (Djebbar *et al.,* 2011; Zwingelstein, 2009) :

- un système de détection de chaleur
- un système de détection de fumée

- un système d'alarme à commande manuelle

Comme illustré dans la figure suivante (**Figure.IV.1.**) (Djebbar *et al.*, 2011), nous avons intégré le système d'alarme à commande manuelle dans le système de détection de chaleur étant donné que le relais à commande manuelle est relié au système de détection de chaleur par le pressostat. De plus nous avons généralisé le nombre de composants qui sera défini durant l'étude du système, et nous avons aussi simplifié le fonctionnement du voteur.

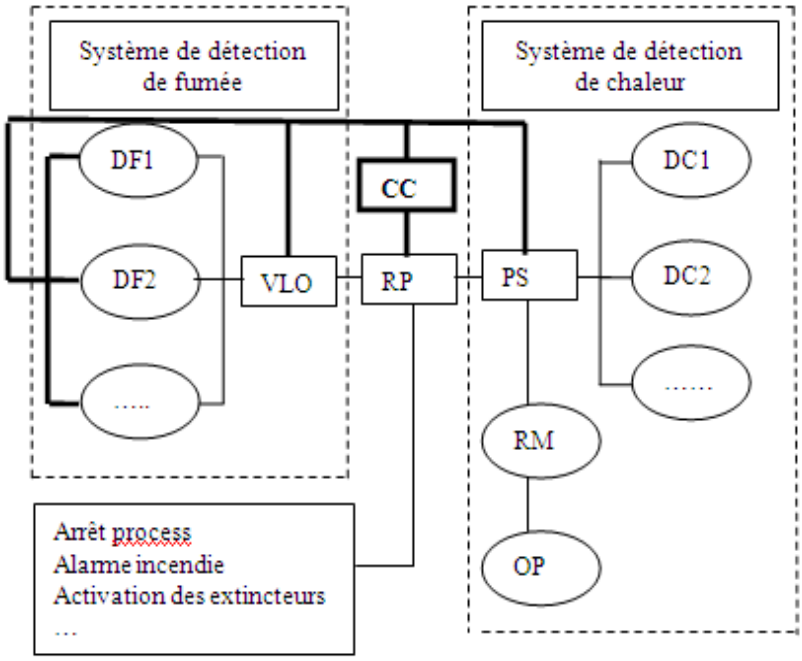


Figure.IV.1. Schéma du système de détection d'incendie

Le système de détection de chaleur comprend des détecteurs de chaleur (DC1, DC2, etc.) composés de cellules fusibles à 72 °C mises en série et reliés à un circuit sous pression.

En cas de dépassement de 72 °C les fusibles fondent, la pression dans le circuit baisse et déclenche le pressostat PS qui actionne le relais principal RP. Le pressostat PS et le relais RP nécessitent la présence de la source de tension continue (batterie) CC. Il suffit qu'un seul détecteur de chaleur se déclenche pour obtenir la perte de pression. L'alarme à commande manuelle est sous la responsabilité d'un opérateur OP toujours présent dans l'atelier. En cas d'urgence, il peut déclencher le relais à commande manuelle RM qui active le pressostat PS et donc le relais RP.

Le système de détection de fumée comprend des détecteurs de fumée (DF1, DF2, etc.) reliés à un système de vote VLO. Si au moins un détecteur est activé, le système de vote actionne le relais principal RP (VLO nécessite la présence de la source de tension CC). Les détecteurs de fumée nécessitent aussi la présence de la source de tension CC.

IV.2.2 Modes de défaillances

Pour l'étude de l'impact des défaillances des composants sur le système global, nous avons repris les modes de défaillances des composants identifiés dans (Zwingelstein, 2009), et nous avons identifié leurs conséquences (**Tableau.IV.1**.).

Composant	**Mode de défaillance**	**Conséquences de la défaillance sur la mission du composant**
DC	PANNE	Pas de signal d'incendie
RM	PANNE	Pas de transmission de signal d'incendie de l'opérateur au PS
OP	PANNE	Pas de signal d'incendie
PS	PANNE	Pas de transmission de signal du système de détection de chaleur au RP
RP	PANNE	Pas de transmission de signal du RP
DF	PANNE	Pas de signal d'incendie
VLO	PANNE	Pas de transmission de signal du système de détection de fumée au RP
CC	PANNE	Pas d'alimentation en courant continu pour : RP, et les systèmes de détection de chaleur et d'incendie.

Tableau.IV.1. Modes de défaillances

IV.3 Modélisation suivant FIGARO

IV.3.1 Modélisation FIGARO1

A- Détermination des types et de leurs interfaces (Djebbar *et al.,* 2011)

Les types sont très simplement définis comme représentation directe des composants présents dans le système, auxquels on a ajouté les types : « nœud », « nœud_signal», « nœud_detecteur », et « noeud_signal_intermediaire » afin d'y déclarer les caractéristiques communes. La figure suivante (**Figure.IV.2.**) représente l'héritage entre les types.

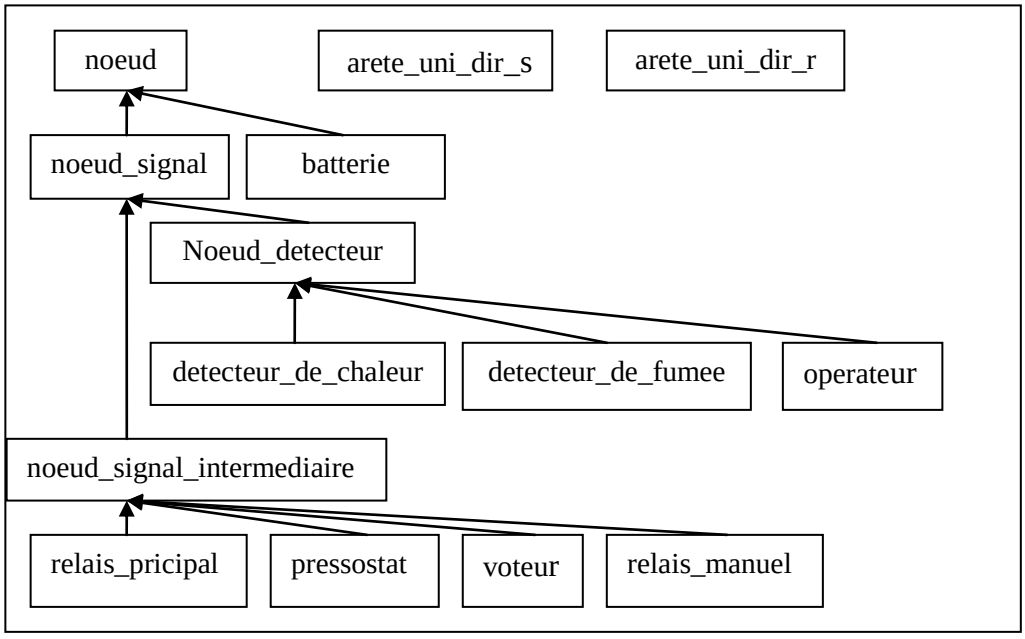


Figure.IV.2. Héritage entre les types

Les variables héritées sont présentées dans le tableau suivant (**Tableau.IV.2.**). La variable « incendie » représente la détection de l'incendie au niveau des nœuds détecteurs. Les variables « signal » et « relie » représentent respectivement la propagation du signal d'incendie et la propagation du courant.

TYPE	SORTE_DE	PANNE	CONSTANTE	ATTRIBUT	EFFET
nœud	X	Def	fonction lambda mu	X	relie
noeud_signal	noeud	X	X	X	signal
noeud_detecteur	noeud_signal	X	X	incendie	X
noeud_signal_ intermediaire	noeud_signal	X	X	X	X

Tableau.IV.2. Variables héritées

Pour représenter les connexions entre les composants nous avons défini les interfaces (**Tableau.IV.3.**) dans les types « arete_uni_dir_s » et « arete_uni_dir_r ». Le type « arete_uni_dir_s » permet de transmettre le signal d'incendie, et le type « arete_uni_dir_r » permet de transmettre le courant continu.

TYPE	NOM DE L'INTERFACE	TYPE DE L'INTERFACE
arete_uni_dir_s	depart_s	noeud_signal
	arrivee_s	noeud_signal_intermediaire
arete_uni_dir_r	depart_r	batterie
	arrivee_r	noeud_signal

Tableau.IV.3. Interfaces

B- Définition des règles (Djebbar *et al.,* 2011)

Nous avons déclaré une seule règle d'occurrence commune à tous les composants dans le type nœud :

```
IL_PEUT_SE_PRODUIRE     DEFAILLANCE def
 LOI EXP(lambda);
```

Selon cette règle tout composant peut passer de l'état « MARCHE » à l'état « PANNE ».

Notons que la syntaxe de la règle impose une loi de fiabilité. Nous avons adopté « la loi exponontielle » avec le taux de défaillance « lambda » (§ I.2.4).

Pour faire interagir les composants afin de propager les effets nous avons déclaré des règles d'interactions au niveau des nœuds et des liens. Par exemple, la règle d'intection du nœud « batterie » est la suivante :

```
SI MARCHE  ALORS relie;
```

- *Au niveau du nœud « batterie » :* le nœud « batterie » génère du courant continu (relie =VRAI) s'il est en marche (def = FAUX).
- *Au niveau des nœuds détecteurs :* les nœuds détecteurs déclenchent le signal si (incendie = VRAI). Si le nœud détecteur nécessite d'être relié au courant électrique alors « relie » doit avoir la valeur « VRAI » pour le déclenchement du signal.
- *Au niveau des liens « arete_uni_dir_r » :* pour la propagation du courant, le nœud « depart_r » (batterie) doit être relié au courant (relie = VRAI) et le nœud « arrivee_r » doit être en marche « def=faux ».
- *Au niveau des liens « arete_uni_dir_s » :* pour la propagation du signal, le nœud « depart_t » doit avoir « signal = VRAI », et le nœud « arrivee_s » doit être en marche « def = faux » et « relie = VRAI » pour les nœuds nécessitant d'être reliés au courant continu.

IV.3.2 Paramétrage graphique et validation du modèle FIGARO1

A- Paramétrage graphique

a- Affectation de la catégorie graphique et des icônes aux types FIGARO. Les types «arete_uni_dir_s» et « arete_uni_dir_r », qui représentent un fil seront évidement considérés dans l'interface graphique comme un lien. Les types « detecteur_de_fumee », « detecteur_de_chaleur », « operateur », « pressostat », « voteur », « relais_pricipal », « relais_manuel », « batterie », et « operateur » seront considérés comme des noeuds du graphe et représentés sous la forme d'une icône. Les types « nœud », « noeud_signal », « noeud_detecteur », et « noeud_signal_intermediaire » seront considérés comme des nœuds, mais ne figurent pas dans le graphe du système. La figure suivante illustre la déclaration des nœuds et des liens avec VisualFigaro :

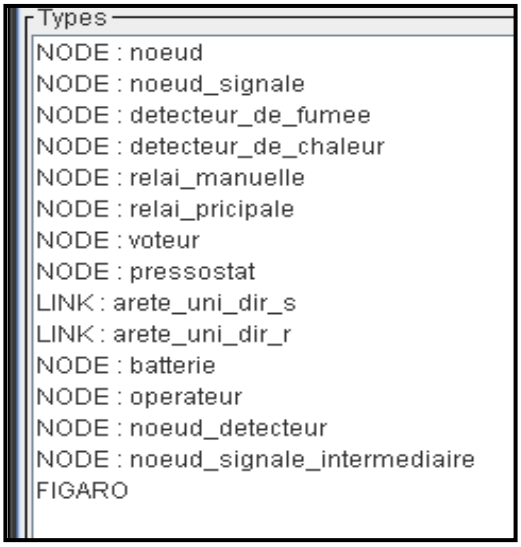


Figure.IV.3. Déclaration des nœuds et des liens

Le tableau suivant montre les icônes et les abréviations des types graphiques, que nous avons adoptés :

TYPE	ABREVIATION	ICONE
detecteur_de_chaleur	dc	
relais_manuel	rm	
detecteur_de_fumee	df	
pressostat	ps	
voteur	vote	
batterie	btr	
operateur	opr	
relais_pricipal	rp	
arete_uni_dir_s	art_ud_s	
arete_uni_dir_r	art_ud_r	

Tableau.IV.4. Representation graphique des composants

b- Définition des points de connexions. A fin d'éviter les répétitions, nous avons déclaré tous les points de connections à l'exception du point de connexion (sortie_r propre à la batterie) dans le type « nœud_signal » (**Tableau.IV.5.)**. Tous les points de connections des nœuds (sauf la batterie) serons hérités du type « noeud_signal ».

Nous avons défini trois points de connections dans le type « nœud_signal » : « entree_s », « sortie_s », et « entree_r », ces points de connexions peuvent être accrochés à n'importe quel type de liens (type FIGARO). Nous avons préféré spécifier les contraintes de connections dans les liens.

NOM	**POSITION**	**CONNEXION_ACCEPTEE**	
		NOM	**TYPE_POINT_CONNEXION**
entree_s	(0,0)	ARRIVEE	FIGARO
sortie_s	(0,0)	DEPART	FIGARO
entree_r	(0,0)	ARRIVEE	FIGARO

Tableau.IV.5. Points de connections du type « nœud_signal »

Les liens ont deux points de connexions : « DEPART » et « ARRIVE » qui correspondent aux deux extrémités du lien. Les points de connections des liens sont présentés dans les tableaux suivants : (**Tableau.IV.6.)** pour le lien « arete_uni_dir_s », et (**Tableau.IV.7.)** pour le lien « arete_uni_dir_r ».

NOM	**CONNEXION_ACCEPTEE**		
	NOM	**TYPE_POINT_CONNEXION**	**INTERFACE REMPLIE PAR LE NOEUD**
DEPART	sortie_s	noeud_signal	depart_s
ARRIVEE	entree_s	noeud_signal	arrivee_s

Tableau.IV.6. Points de connections du lien « arete_uni_dir_s »

NOM	CONNEXION_ACCEPTEE		
	NOM	TYPE_POINT_CONNEXION	INTERFACE REMPLIE PAR LE NOEUD
DEPART	sortie_r	batterie	depart_r
ARRIVEE	entree_r	pressostat	arrivee_r
	entree_r	detecteur_de_fumee	arrivee_r
	entree_r	relais_principal	arrivee_r
	entree_r	voteur	arrivee_r

Tableau.IV.7. Points de connexions du lien « arete_uni_dir_r »

c- Définition des visualisations. Il est possible de visualiser sur le schéma du système l'état des nœuds (valeurs des variables « signal » et « def ») pour cela nous avons défini une seule visualisation « Visu ».

d- Définition des variantes graphiques. La visualisation « Visu » est définie grâce aux variantes graphiques. Nous avons défini trois variantes graphiques comme illustré dans le tableau suivant :

NOM	DECLARE DANS LE NOEUD	HARITE PAR	CONDITION	VISUALISATION
VG1	noeud	Tous les nœuds graphiques	def = VRAI	Le fond de l'icône devient rouge
VG2	noeud_signal	Tous les nœuds graphiques à l'exception du nœud batterie	signal = VRAI	L'icône est commentée par le texte : « signal »
VG3	noeud_signal	Tous les nœuds graphiques à l'exception du nœud batterie	signal = FAUX	L'icône est commentée par le texte : « pas de signal »

Tableau.IV.8. Variantes graphiques

B- Validation du modèle FIGARO1

a- Vérification syntaxique : La vérification syntaxique a été effectuée pendant l'écriture du code FIGARO1 avec l'outil VisualFigaro, et ensuite par le chargement du modèle avec l'outil KB3.

b- Vérification du paramétrage graphique : Après le chargement de la base de connaissances (modèle FIGARO1 et son paramétrage graphique) avec l'outil KB3, nous avons vérifié le paramétrage des points de connections (contraintes de connections et règles de remplissage d'interface) par l'assemblage des composants. Des exemples sont donnés dans les figures : (**Figure.IV.4.**), et (**Figure.IV.5.**).

La figure (**Figure.IV.4.**) présente un message d'erreur à la suite d'une tentative de connexion d'un nœud de type « batterie » à un nœud de type « detecteur_de_chaleur » avec un lien de type « arete_uni_dir_r » (la connexion du type « detecteur_de_chaleur » ne fait pas parti des connections acceptées par le point de connexion « ARRIVEE » du type « arete_uni_dir_r ») (**Tableau.IV.7.)**.

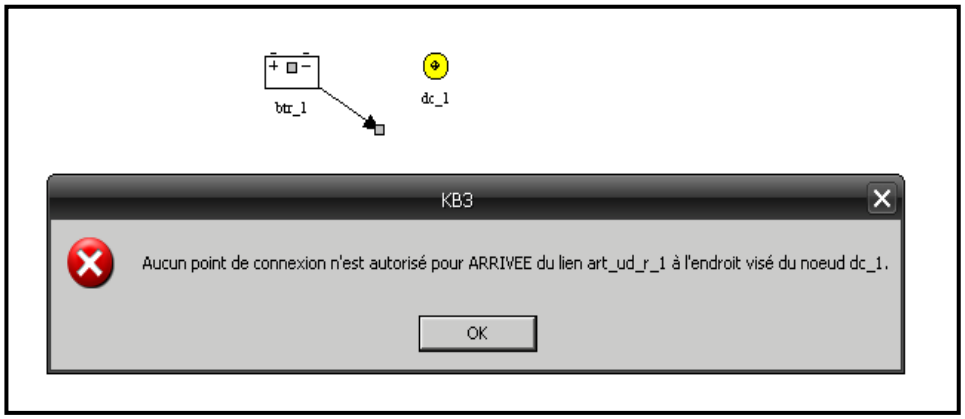


Figure.IV.4. Message d'erreur pour la connexion de la batterie à un détecteur de chaleur

La figure (**Figure.IV.5.**) montre que les interfaces du lien « art_ud_r_1 » sont belles et bien remplies par les nœuds « ps_1 » et « btr_1 » (« depart_r » correspond au nœud « btr_1 », et « arrivee_r » correspond au nœud « ps_1 ») **(Tableau.IV.7.)**.

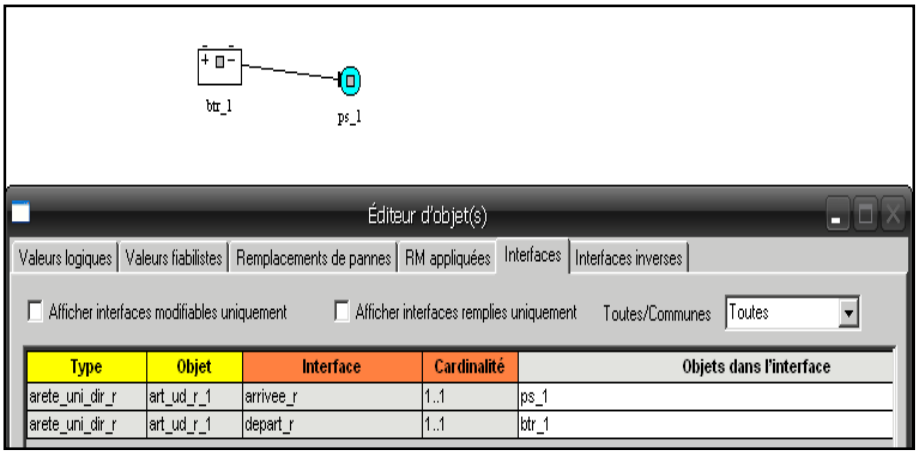


Figure.IV.5. Remplissage des interfaces « depart_r » et « arrivee_r » du nœud « art_ud_r_1 »

Nous avons ensuite vérifié le paramétrage de la visualisation et des variantes graphiques, un exemple est donné dans la figure (**Figure.IV.6.**). Cette figure montre la visualisation des états des nœuds « btr_1 », et « ps_1 ». Le nœud « btr_1 » présente une défaillance (def= VRAI), et il n ya pas de signal d'incendie dans le nœud « ps_1 » (signal= FAUX), ceci valide une partie des spécifications des variantes graphiques (VG1 et VG3) dans le tableau (**Tableau.IV.8.**).

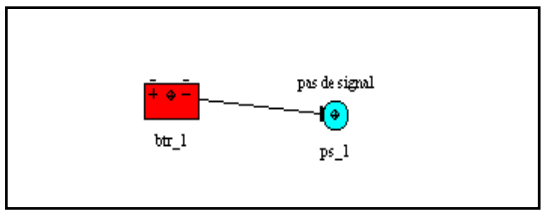


Figure.IV.6. Visualisation des états des nœuds « btr_1 » et « ps_1 »

c- Vérification sémantique : Grâce à la visualisation, et au simulateur interactif intégré dans l'outil, nous avons vérifié la sémantique du modèle FIGARO1 qui correspond parfaitement aux spécifications du système présentées précédemment (§ IV.2).

Les figures suivantes donnent un aperçu de la vérification sémantique d'un système de test où on a :

- Un détecteur de chaleur (dc_1)

- Un pressostat (ps_1)
- Un relais principal (rp_1)
- Une batterie (btr_1)

La figure (**Figure.IV.7**) représente la visualisation statique des états du système, on remarque que les règles d'interactions (§ IV.3.1) pour la transmission du signal d'incendie dans les types : « detecteur_de_chaleur », « arete_uni_dir_s », et « pressostat », ainsi que la transmission du courant dans les types : « batterie », « arete_uni_dir_r », et « pressostat » fonctionne comme on le souhaite.

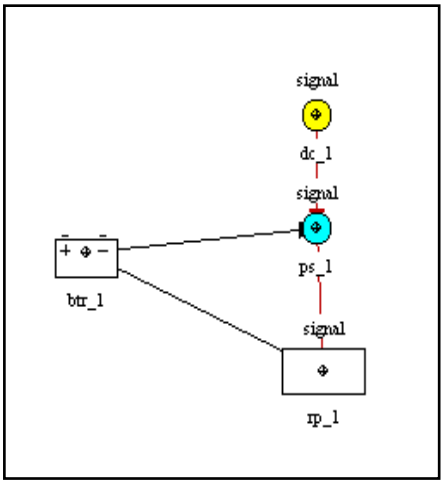


Figure.IV.7. Propagation du signal et du courant
dans le système de test

Dans la figure (**Figure.IV.8**) on considère la défaillance de la batterie « btr_1 » avec le taux de défaillance lambda = 1e-5, le comportement du système est cohérent et logique : il n ya plus de transmission de signal entre le nœud « dc_1 » et « ps_1 » étant donné que ce dernier n'est plus relié au courant continu, et par conséquent il n y a plus de transmission de signal vers « rp_1 ».

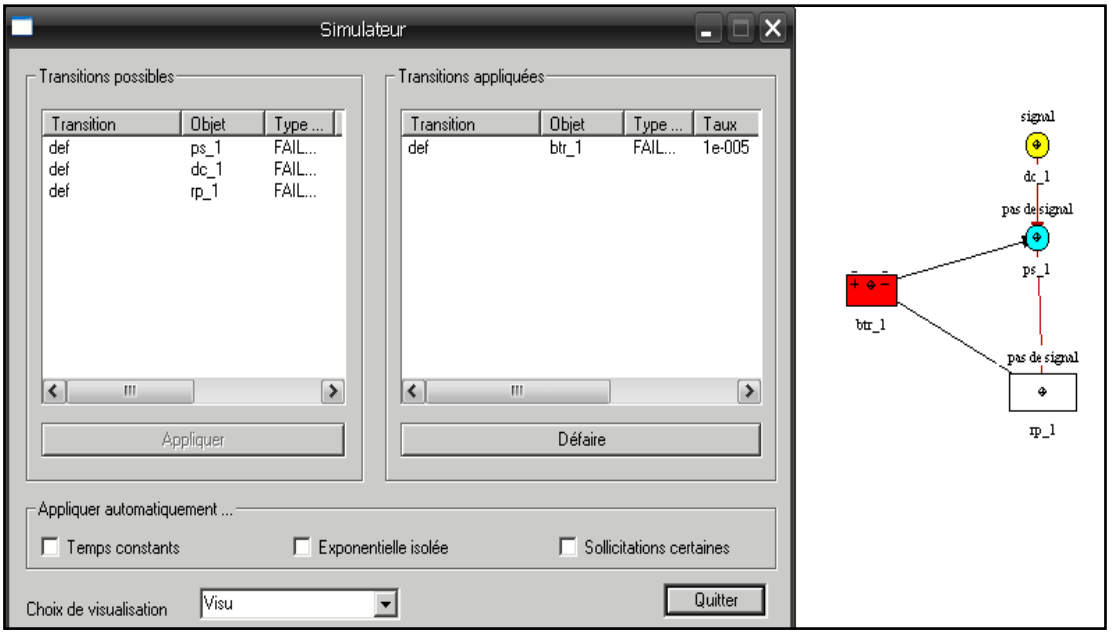


Figure.IV.8. Simulation interactive du système de test avec la défaillance de la batterie

IV.3.3 Modélisation FIGARO0

Grâce à la base de connaissances développée précédemment (modèle FIGARO1 et son paramétrage graphique) et à l'outil KB3, nous avons modélisé le schéma physique complet du système de détection d'incendie comme présenté dans la figure (**Figure.IV.9.**) (Djebbar *et al.*, 2011). Le système comprend :

- Un opérateur (opr_1)
- Un relais manuel (rm_1)
- Deux détecteurs de chaleur (dc_1 et dc_2)
- Deux détecteurs de fumée (df_1 et df_2)
- Un voteur (vote_1)
- Un pressostat (ps_1)
- Un relais principal (rp_1)
- Une batterie (btr_1)

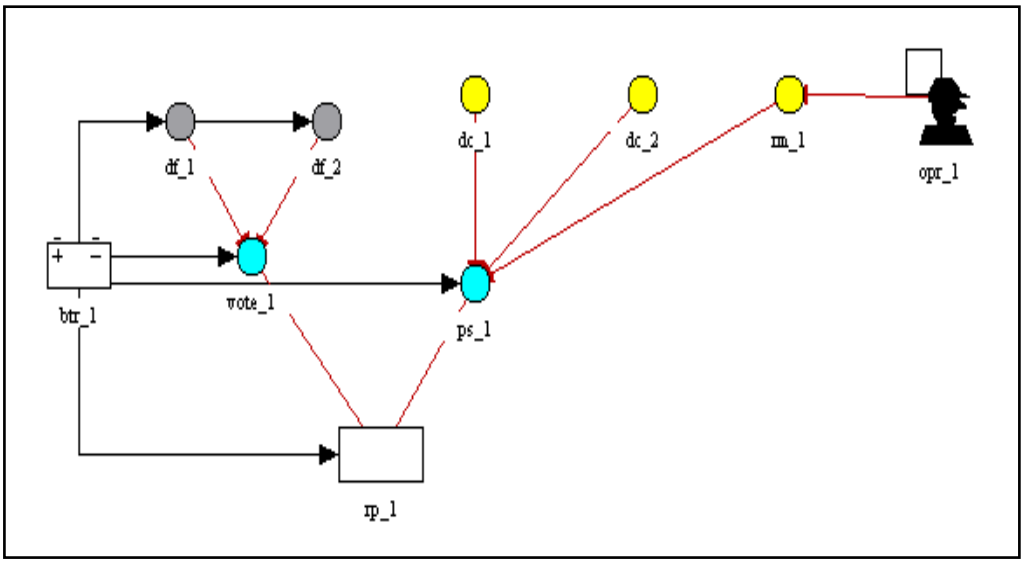


Figure.IV.9. Schéma du système de détection d'incendie

Le schéma du système (**Figure.IV.9.**) correspond à la liste des objets suivants :

```
OBJET opr_1 EST_UN operateur;
OBJET rm_1 EST_UN relais_manuel;
OBJET dc_1 EST_UN detecteur_de_chaleur;
OBJET dc_2 EST_UN detecteur_de_chaleur;
OBJET df_1 EST_UN detecteur_de_fumee;
OBJET df_2 EST_UN detecteur_de_fumee;
OBJET ps_1 EST_UN pressostat;
OBJET vote_1 EST_UN voteur;
OBJET rp_1 EST_UN relais_principal;
OBJET btr_1 EST_UN batterie;
OBJET art_ud_s_1 EST_UN arete_uni_dir_s;
   INTERFACE depart_s = opr_1; arrivee_s = rm_1;
OBJET art_ud_s_2 EST_UN arete_uni_dir_s;
   INTERFACE depart_s = rm_1; arrivee_s = ps_1;
OBJET art_ud_s_3 EST_UN arete_uni_dir_s;
   INTERFACE depart_s = dc_1; arrivee_s = ps_1;
OBJET art_ud_s_4 EST_UN arete_uni_dir_s;
   INTERFACE depart_s = dc_2; arrivee_s = ps_1;
OBJET art_ud_s_5 EST_UN arete_uni_dir_s;
```

```
    INTERFACE depart_s = df_1; arrivee_s = vote_1;
OBJET art_ud_s_6 EST_UN arete_uni_dir_s;
    INTERFACE depart_s = df_2; arrivee_s = vote_1;
OBJET art_ud_s_7 EST_UN arete_uni_dir_s;
    INTERFACE depart_s = ps_1; arrivee_s = rp_1;
OBJET art_ud_s_8 EST_UN arete_uni_dir_s;
    INTERFACE depart_s = vote_1; arrivee_s = rp_1;
OBJET art_ud_r_1 EST_UN arete_uni_dir_r;
    INTERFACE depart_r = btr_1; arrivee_r = df_1;
OBJET art_ud_r_2 EST_UN arete_uni_dir_r;
    INTERFACE depart_r = btr_1; arrivee_r = df_2;
OBJET art_ud_r_3 EST_UN arete_uni_dir_r;
    INTERFACE depart_r = btr_1; arrivee_r = vote_1;
OBJET art_ud_r_4 EST_UN arete_uni_dir_r ;
    INTERFACE depart_r = btr_1; arrivee_r = ps_1;
OBJET art_ud_r_5 EST_UN arete_uni_dir_r;
    INTERFACE depart_r = btr_1; arrivee_r = rp_1;
```

Le modèle FIGARO0 est généré à partir de cette liste et au modèle FIGARO1, par exemple : le code FIGARO0 correspondant aux objets « btr_1 » et « art_ud_r_1 » est le suivant :

```
OBJET btr_1 EST_UN batterie;
  CONSTANTE
    fonction DOMAINE 'detecteur_de_fumee' 'detecteur_de_chaleur'
                     'batterie' 'relais_pricipal' 'relais_manuel'
                     'pressostat' 'operateur' 'autre' =
                     'batterie';
    lambda DOMAINE REEL = 1e-005;
    mu DOMAINE REEL = 0.1;
  ATTRIBUT
    def DOMAINE BOOLEEN = FAUX;
  ATTRIBUT
    relie DOMAINE BOOLEEN REINITIALISATION FAUX;
```

```
    INTERACTION
      regle1
         ETAPE etape_par_defaut
         SI def DE btr_1 = FAUX
         ALORS relie DE btr_1 <-- VRAI;
    OCCURRENCE
      xx1
         GROUPE groupe_simu
         SI def DE btr_1 = FAUX
         IL_PEUT_SE_PRODUIRE
           DEFAILLANCE def
           LOI EXP (1e-005)
           PROVOQUE def DE btr_1 <-- VRAI;

OBJET art_ud_r_1 EST_UN arete_uni_dir_r;
    INTERFACE
      depart_r = btr_1;
      arrivee_r = df_1;
    INTERACTION
      regle1
         ETAPE etape_par_defaut
         SI relie DE btr_1 ET (def DE df_1 = FAUX)
         ALORS relie DE df_1 <-- VRAI;
```

Remarque : La génération du modèle FIGARO0 se fait automatiquement par l'outil KB3 à partir du schéma physique du système et de la base de connaissances.

IV.4 Modélisation suivant AltaRica

IV.4.1 Modélisation AltaRica

A partir du modèle FIGARO0 nous avons déduit le modèle AltaRica. Le nœud pricipal « main » est composé de :

- *Un système de détection de chaleur « sys_chal_1 » :* ce nœud contient deux detecteurs de chaleur « dc_1 »,« dc_2 », un pressostat « ps_1 », un opérateur « opr_1 », et un relais manuel « rm_1 ».
- *Un système de détection de fumée « sys_fum_1 » :* ce nœud contient deux detecteurs de fumée « df_1 », « df_2 », et un voteur « vote_1 ».
- *Un relais principal « rp_1 ».*
- *Une batterie « btr_1 ».*

Les variables de flux déclarées dans le modèle sont

- *e_signal* : entrée du signal d'incendie.
- *s_signal :* sortie du signal d'incendie.
- *e_relie :* entrée du courant continu.
- *s_relie :* sortie du courant continu.

Les variables d'état déclarés dans le modèle sont :

- *marche :* le composant est en état de marche.
- *incendie :* le composant a détécté un incendie.

Les événements déclarés dans chaque composant sont :

- *DEFAILLANCE* : occurrence d'une défaillance.

Les transitions représentant le passage de l'état « marche » à l'état « panne »déclarés dans chaque composant sont :

```
marche |- DEFAILLANCE -> marche := false;
```

Nous avons déclaré aussi un enssemble d'assertions pour le contrôle des interactions entre les sous-nœuds, et pour la propagation du signal ou du courant continu dans les composants.

Comme par exemple les assertions représentant la propagation du courant continu dans le nœud « btr_1 » et dans le nœud principal « main » sont :

- *Le nœud « btr_1 » :*

```
s_relie = marche;
```

- *Le nœud « main » :*

```
btr_1.s_relie = rp_1.e_relie;
btr_1.s_relie = sys_chal_1.e_relie;
btr_1.s_relie = sys_fum_1.e_relie;
```

Le code AltaRica des nœuds « main », et « btr_1 » est le suivant :

```
node main
   sub
       sys_chal_1 : system_detect_chaleur;
       sys_fum_1 : system_detect_fumee;
       rp_1 : relais_principal;
       btr_1 : batterie;
   flow
       s_signal :bool ;
   assert
       btr_1.s_relie = rp_1.e_relie  ;
       btr_1.s_relie = sys_chal_1.e_relie;
       btr_1.s_relie = sys_fum_1.e_relie;
       rp_1.e_signal = (sys_chal_1.s_signal or
                        sys_fum_1.s_signal;
       s_signal = rp_1.s_signal;
edon

node batterie
     flow
         s_relie : bool;
         state
         marche :bool;
     event
         DEFAILLANCE;
     Trans
```

```
            marche |- DEFAILLANCE -> marche := false;
        assert
            s_relie = marche;
        init
            marche := true;
edon
```

IV.4.2 Validation du modèle AltaRica

a- Vérification syntaxique : La vérification syntaxique a été effectuée pendant le chargement du modèle par l'outil ARC.

b- Vérification sémantique : la vérification sémantique a été effectuée avec le simulateur interactif integré à l'outil ARC. Le comportement du système est conforme aux spécifications précédemment, présentées sur le système (§ IV.2).

Un aperçu des tests effectués est présenté dans les figures (**Figure.IV.10.** et **Figure.IV.11.)** (pour la vérification du comportement du sous-système de détection de fumée).

Les assertions du sous-système « sys_fum_1 » sont :

```
vote_1.e_signal = (df_1.s_signal or df_2.s_signal);
s_signal = vote_1.s_signal;
e_relie = vote_1.e_relie;
e_relie = df_1.e_relie;
e_relie = df_2.e_relie;
```

Les assertions des nœuds « df_1 », « df_2 », et « vote_1 » sont :

- *« df_1 » et « df_2 » :*

```
s_signal = (incendie and  marche and e_relie);
```

- *«vote_1 » :*

```
s_signal = (e_signal and marche and e_relie);
```

La figure (**Figure.IV.10.**) montre la propagation du signal, et du courant continu dans le sous-système de détection de fumée « sys_fum1 ».

La valeur de la variable de flux (e_relie= true) des nœuds « df_1 », « df_2 », et « df_3 » montre que les assertions pour le passage du courant dans le sous_système de détection de fumée fonctionnent comme on le souhaite

La valeur des variables de flux « s_signal » du sous-système de détection de fumée « sys_fum_1 » et des nœuds « df_1 », « df_2 », et « vote_1 » (et « e_signal » pour le nœud « vote_1 ») montre que les assertions pour le passage du signal dans le sous-système de détéction de fumée fonctionnent comme on le souhaite.

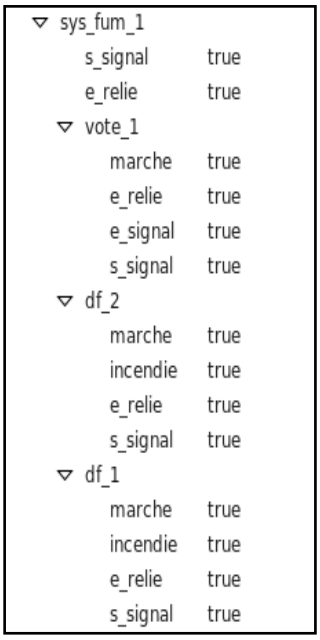


Figure.IV.10. Propagation du signal et du courant dans « sys_fum_1 »

Dans la figure (**Figure.IV.11**) on considère la défaillance de la batterie « btr_1 », le comportement du sous système de détection de fumée « sys_fum_1 » est cohérent et logique :

- Il n ya plus de transmission de courant de la batterie vers le sous-système de détection de fumée « sys_fum_1 », et par conséquent vers le voteur « vote_1 », et vers les détecteurs de fumée « df_1 » et « df_2 ».
- Il n ya plus de transmission de signal des composants nécessitant d'être reliée au courant continu : « df_1 », « df_2 », et « vote_1 », et par conséquent pas de transmission de signal du sous-système de détection de fumée « sys_fum_1 ».

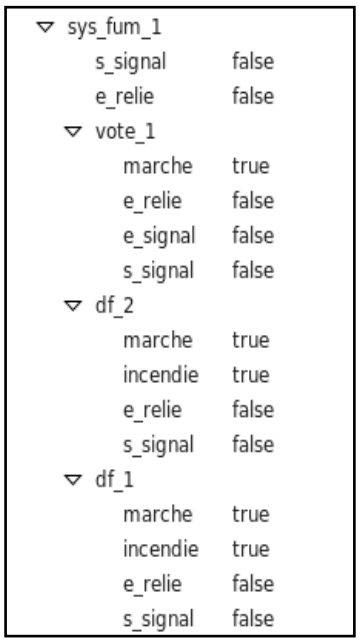


Figure.IV.11. Simulation interactive du sous-système « sys_fum_1 » avec la défaillance de la batterie

IV.5 Génération de modèles et simulations

La dernère étape donne les resultats obtenus sous forme d'un modèle SdF suivant un AdD, des simulations, et des séquences d'évènements. Cette étape représente une interface pour les études qualitatives et quantitatives de SdF à partir de ces résultats.

IV.5.1 Génération de l'arbre de défaillances

A- Génération automatique (Djebbar *et al.*, 2011)

L'événement redouté dans un système de détection d'incendie est « l'absence de signal en sortie du système ». Le composant représentant le point de sortie du signal du système est le relais principal, alors l'événement redouté est : « signal (rp_1) = FAUX ». Avec KB3, l'arbre de défaillances (**Figure.IV.12.**) représente la négation de l'arbre généré par chainage arrière des régles instanciées definies dans le modèle FIGARO. Le point de depart du chainage arrière est la négation de l'événement indésirable.

Nous avons definis l'événement indésirable « E1 » par un renvoi système sur la variable (signal = FAUX) du relais principale « rp_1 ».

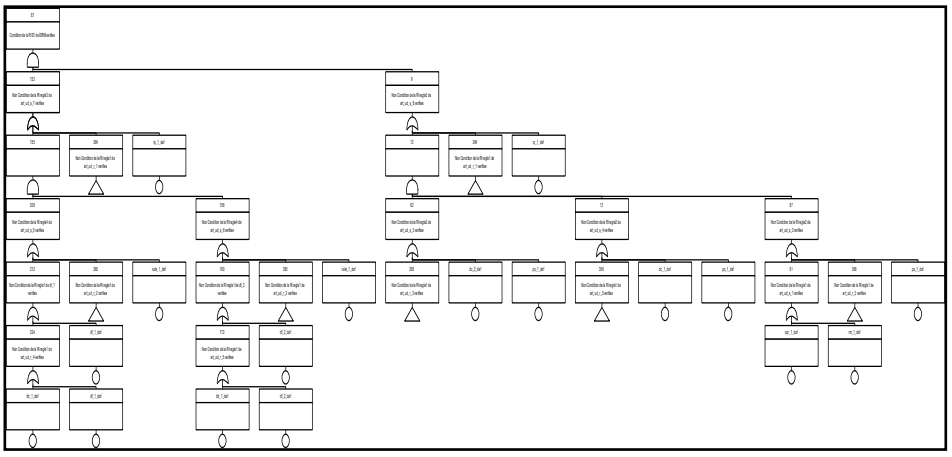


Figure.IV.12. Aperçu de l'arbre de défaillances généré automatiquement

B- Réduction de l'arbre

Etant donné la taille de l'arbre généré, nous proposons un arbre réduit (**Figure.IV.13.**) obtenu par la démarche suivante :

- Modification de l'événement indésirable afin de décomposer l'arbre en sous arbres.
- Génération de chaque sous arbre avec KB3, et nous avons procédé à leurs réduction,
- Rassemblement des sous arbres réduits, ce qui donne l'arbre global réduit.

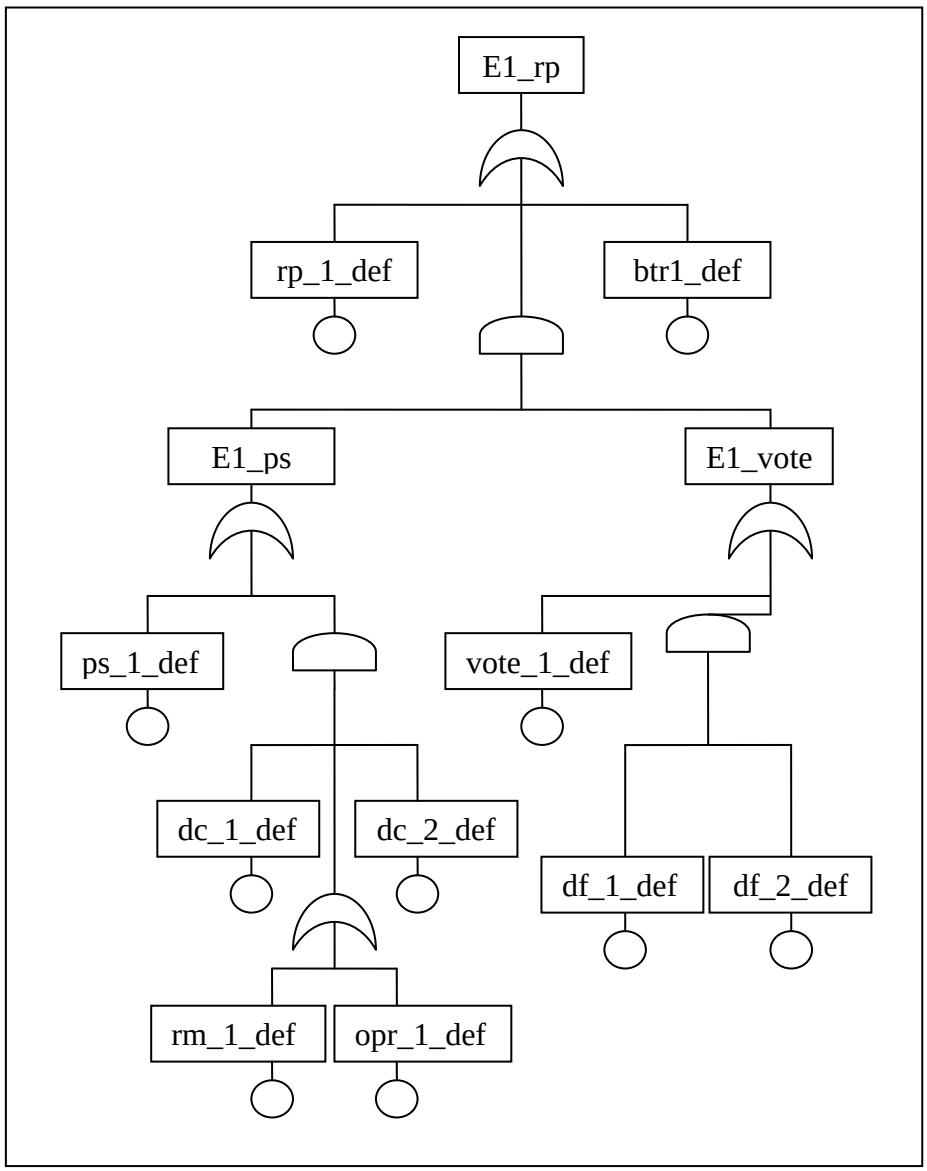


Figure.IV.13. Arbre de défaillances global réduit

A partir du modèle généré il devient possible de déduire les combinaisons de défaillances menant le système à une absence de signal d'incendie, et d'établir une analyse qualitative et quantitative du système de détection d'incendie.

IV.5.2 Simulations

A- Visualisation statique des états

On peut visualiser l'impact des combinaisons des défaillances sur le schéma du système grâce au simulateur inetégré à l'outil KB3.

Des exemples de combinaisons de défaillances sont présentés dans les figures suivantes :

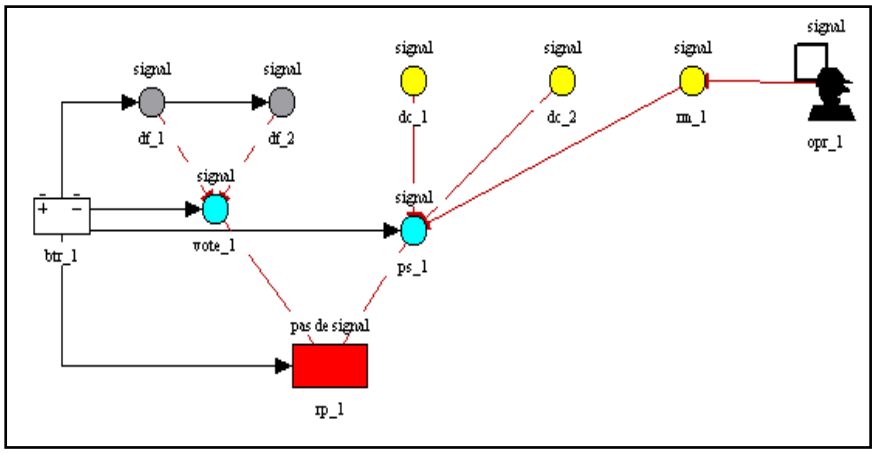


Figure.IV.14. Visualisation statique du système global avec la défaillance du relais principal

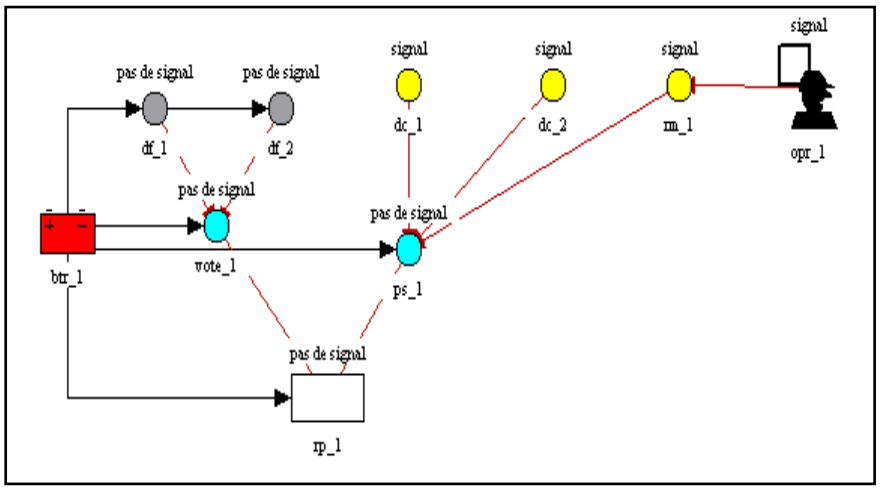


Figure.IV.15. Visualisation statique du système global avec la défaillance de la batterie

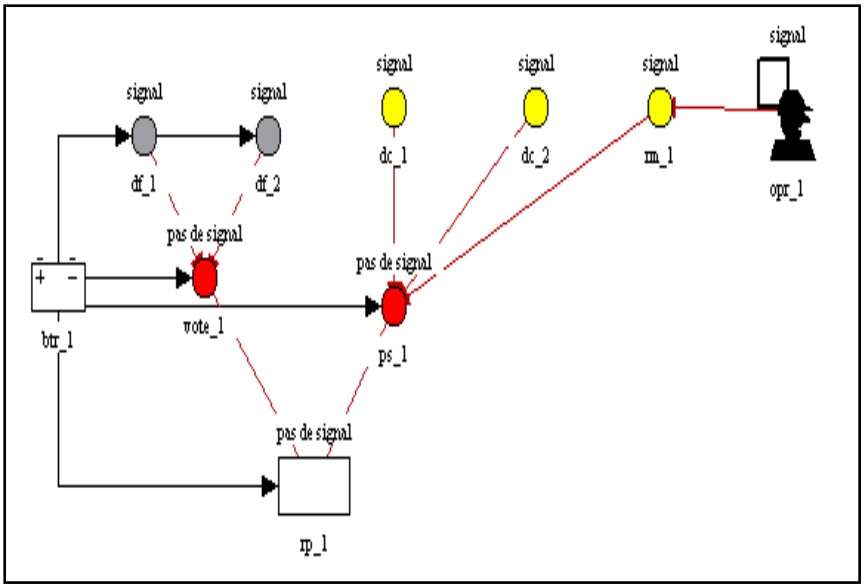


Figure.IV.16. Visualisation statique du système global avec la défaillance du voteur et du pressostat

B- Simulations interactives

a- Simulation interactive avec FIGARO. Il est possible d'observer l'évolution du système suite à une séquence d'évènements.

L'exemple suivant (**Figure.IV.17.** et **Figure.IV.18.**) représente l'évolution de l'état du système suite à la défaillance du voteur, puis la défaillance du pressostat. La séquence d'évènements est : vote_1_def -> ps_1_def.

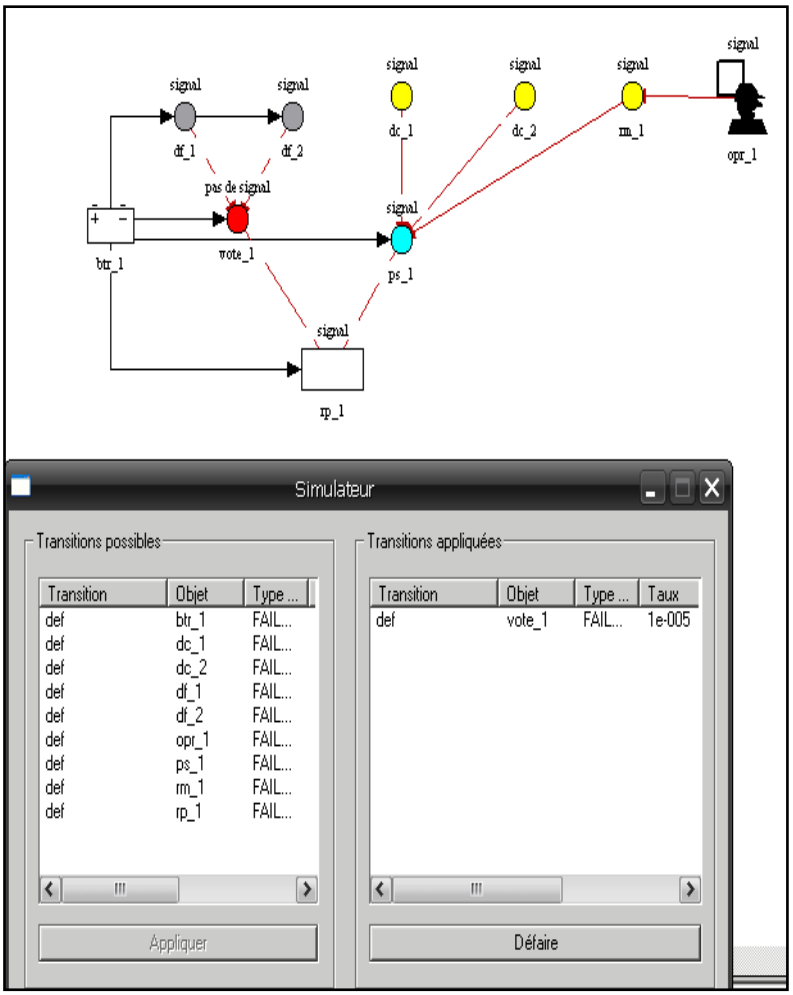


Figure.IV.17. Sélection de la défaillance du voteur, et la visualisation de ses effets sur le système global

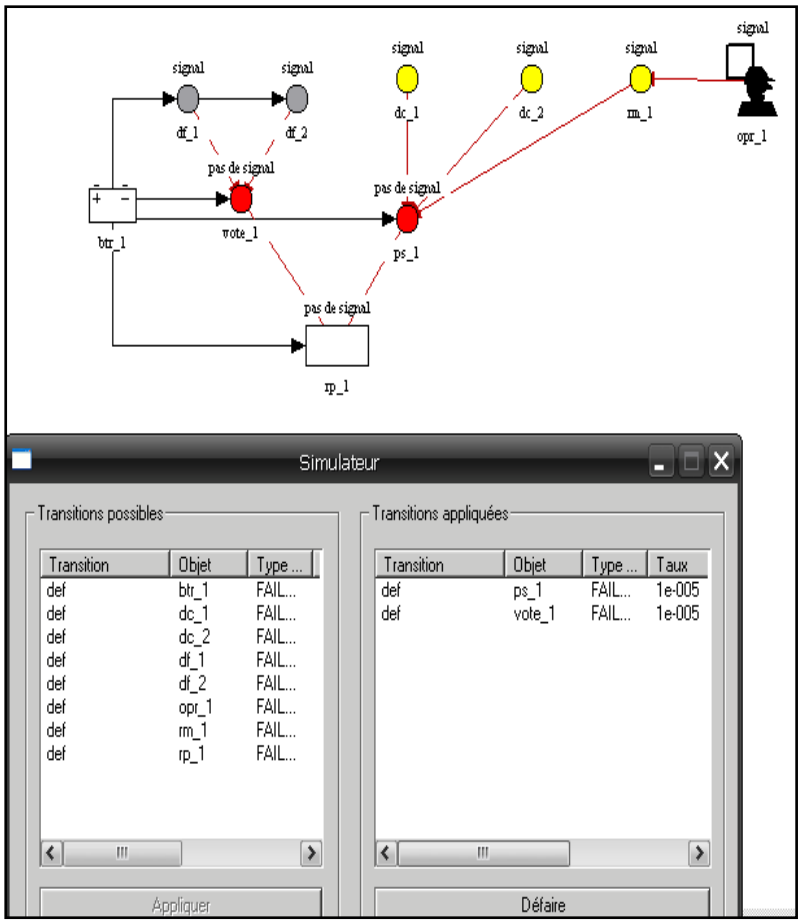


Figure.IV.18. Sélection de la défaillance du pressostat, et la visualisation de ses effets sur le système global

b- Simulation interactive avec AltaRica. Avec le simulateur integré dans l'outil ARC, on peut observer l'évolution d'un composant, sous système, ou système global suite à l'occurrence d'évènements, en visualisant les changements de valeurs des variables.

Comme exemple nous présentons l'évolution du système de détection d'incendie à la suite de la défaillance du voteur puis la défaillance du pressostat (**Figure.IV.19.**, **Figure.IV.20.**, **Figure.IV.21., Figure.IV.22.**).

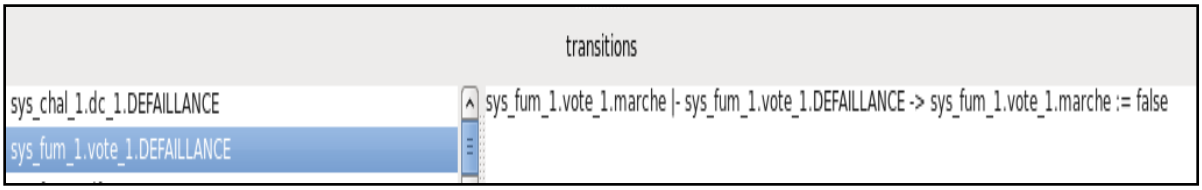


Figure.IV.19. Selection de la défaillance du voteur

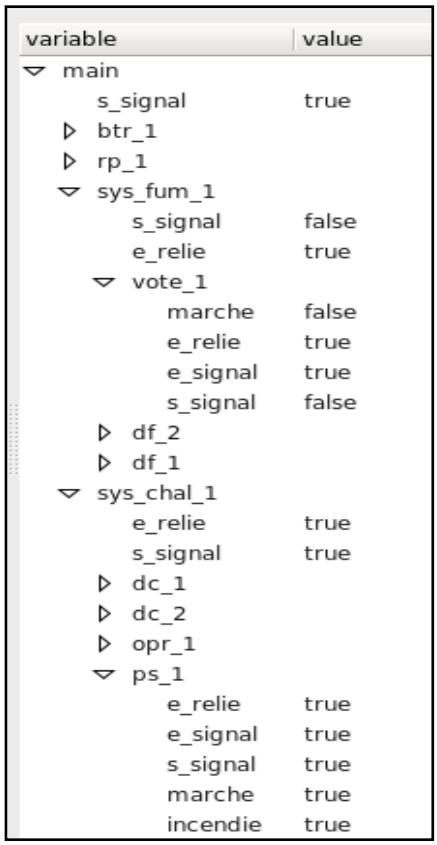


Figure.IV.20. Visualisation des effets de la défaillance du voteur sur le système global

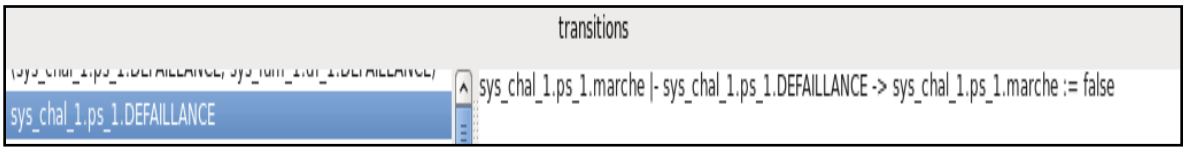


Figure.IV.21. Selection de la défaillance du pressostat

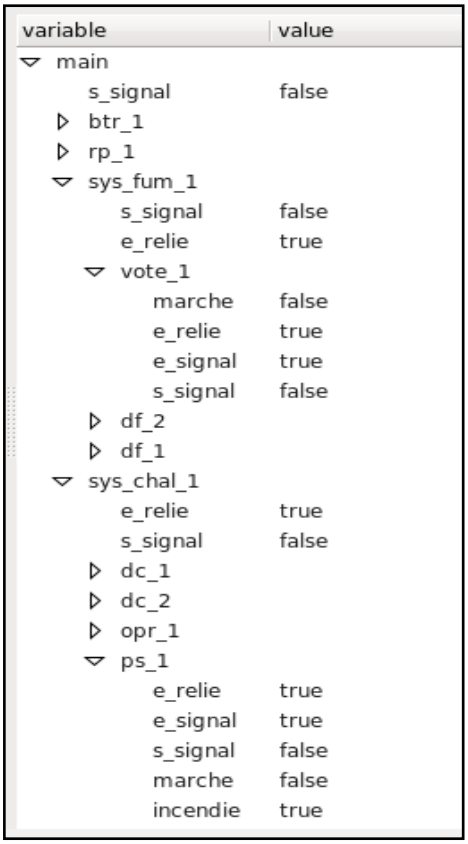


Figure.IV.22. Visualisation des effets de la défaillance du pressostat sur le système global

C- Génération automatique des séquences d'événements

Avec l'outil ARC, il est possible de générer toutes les séquences d'évènements menant le système à la non-détection d'un incendie (s_signal=false du nœud principal). Un aperçu des séquences générées est présenté dans la figure suivante (**Figure.IV.23.**).

```
DEFAILLANCE, sys_fum_1.df_1.DEFAILLANCE) (sys_chal_1.dc_1.DEFAILLANCE) (sys_fum_
1.vote_1.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE, sys_fum_1.df_1.DEFAILLANCE) (sys_chal_1.dc_1.DEFAILLANCE) (sys_fum_
1.df_2.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE, sys_fum_1.df_1.DEFAILLANCE) (sys_fum_1.vote_1.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE, sys_fum_1.df_1.DEFAILLANCE) (sys_fum_1.df_2.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE) (sys_chal_1.dc_1.DEFAILLANCE, sys_fum_1.vote_1.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE) (sys_chal_1.dc_1.DEFAILLANCE, sys_fum_1.df_2.DEFAILLANCE) (sys_fum_
1.vote_1.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE) (sys_chal_1.dc_1.DEFAILLANCE, sys_fum_1.df_2.DEFAILLANCE) (sys_fum_
1.df_1.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE) (sys_chal_1.dc_1.DEFAILLANCE, sys_fum_1.df_1.DEFAILLANCE) (sys_fum_
1.vote_1.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE) (sys_chal_1.dc_1.DEFAILLANCE, sys_fum_1.df_1.DEFAILLANCE) (sys_fum_
1.df_2.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE) (sys_chal_1.dc_1.DEFAILLANCE) (sys_fum_1.vote_1.DEFAILLANCE) ]
[ (sys_chal_1.opr_1.DEFAILLANCE) (sys_chal_1.ps_1.DEFAILLANCE) (sys_chal_1.dc_2.
DEFAILLANCE) (sys_chal_1.dc_1.DEFAILLANCE) (sys_fum_1.df_2.DEFAILLANCE) (sys_fum
_1.df_1.DEFAILLANCE) ]
```

Figure.IV.23. Aperçu des séquences d'evenements menant à la non-signalisation d'un incendie

IV.3 Conclusion

Nous avons présenté dans ce chapitre une mise en œuvre expérimentatale de notre démarche sur un système de détection d'incendie. La modélisation FIGARO, puis le passage vers le modèle AltaRica permet d'exploiter au mieux les apports des deux langages.

CONCLUSION GENERALE

Dans ce mémoire, nous avons présenté une étude suivie d'une démarche pour la modélisation orientée langage, dédiée à la sureté de fonctionnement (SdF) des systèmes. Afin de parvenir à cet objectif, nous avons mené ce travail en plusieurs étapes présentées tout au long de ce document et que nous résumons dans ce qui suit.

Au départ, nous nous sommes intéressés à l'étude de la sureté de fonctionnement, et ses différents analyses, méthodes et modèles; nous avons retenu quelques modèles compte tenu de leurs apports dans les analyses qualitatives et quantitatives et leurs champs d'utilisation

Nous nous sommes intéressés par la suite dans la seconde étape aux langages permettant la génération des modèles et nous avons étudié trois langages réputés incontournables à cause de leurs apports en SdF.

Nous avons comparé dans la troisième étape ces langages et cette comparaison a permis de retenir les langages FIGARO et AltaRica, considérés comme complémentaires. Nous avons ensuite proposé une démarche pour la modélisation orienté langage des systèmes regroupant les plates-formes FIGARO et AltaRica, suivant une approche en plusieurs étapes intégrant des phases de modélisation et de validation des modèles obtenus.

Dans la quatrième étape, nous avons réalisé une mise en œuvre expérimentale de notre démarche sur un système critique de détection d'incendie.

Les résultats obtenus nous ont permis de constater que notre démarche répondait à nos attentes de départ, c'est-à-dire générer les modèles classiques de SdF pour permettre diverses analyses qualitatives et quantitatives et ce à partir d'une modélisation orienté langages.

Nous avons montré qu'il peut être avantageux de modéliser le système de façon globale en déclarant ses caractéristiques avec un langage proche du langage naturel tel que FIGARO1, de l'instancier à partir du schéma physique du système vers FIGARO0, et de déduire par la suite sa modélisation hiérarchique avec AltaRica en respectant certaines contraintes, afin de tirer profit des apports des deux plates formes dédiées à ces langages.

Nous proposons donc comme perspectives à notre travail de :

- Elargir le champ d'expérimentation sur d'autres systèmes.
- Intégrer d'autres outils à la démarche tel que :
  - L''outil YAMS (§ II.2.4) pour l'analyse quantitative de la SdF à partir du modèle FIGARO0.
  - L'outil AltaRica-Studio intégré dans les nouvelles versions de l'outil ARC (§ II.3.4) pour la génération des graphes d'états des systèmes à partir d'un modèle AltaRica.
- Intégrer les deux langages dans un même atelier sous Windows ou Linux.
- Développer un atelier graphique pour le langage AltaRica-Dataflow, intégrant un ensemble d'outils pour la génération des modèles tel que les arbres de défaillances.
- Contribuer à l'évolution du langage AltaRica, et de l'atelier ARC qui est open source.
- Développer un interpréteur du langage FIGARO0 vers le langage AltaRica.

BIBLIOGRAPHIE

Alani T., « Introduction au diagnostic des défaillances », *Cours Diagnostic des défaillances*, Laboratoire A2SI-ESIEE-Paris, Octobre 2006.

Belhadaoui B., Medromi H., Saadi J., Malasse O., « Nouvelle Approche d'Analyse de Fiabilité pour la Sécurité de Fonctionnement à base des Systèmes Multi Agents: Application aux Systèmes de Commande Industriels », *5ème Conférence Internationale CPI'2007*, Rabat (Maroc), 22-24 Octobre 2007.

Bernard R., Analyses de sûreté de fonctionnement multi-systèmes, Thèse de Doctorat en Informatique, Université de Bordeaux 1, Novembre 2009.

Bouissou M., « Modélisation des systèmes programmés en langage FIGARO », *Présenté en réunion du Club 63 (Systèmes Informatiques de Confiance* , LAAS-CNRS, Toulouse (France), 12 Octobre 2000.

Bouissou M., « Dix petits problèmes de modélisation en sûreté de fonctionnement des systèmes », *Article paru dans la revue Phoebus N°34*, juillet-août-septembre 2005.

Bouissou M., Seguin C., « Comparaison des langages de modélisation AltaRica et FIGARO », *15ème Congrès MR-SdF, Lambda-Mu15, Lille*, 10-12 Octobre 2006.

Bouissou M., « Etude de fiabilité des systèmes : les processus de Markov sortent de leur réserve mathématique grâce au BDMP », *La lettre des techniques de l'ingénieur*, Décembre 2006.

Bouissou M., « Manuel de developement de bases de connaissances pour le logiciel KB3 version 3», EDF DOC, Novembre 2010.

Bozzano M., Cimatti A., Katoen J-P., Nguyen V.H., Noll T., Roveri M., Wimmer R., «A Model Checker for AADL», *22nd International Conference CAV2010*, Edimbourg (Royaume-Uni), July 15-19, 2010.

Djebbar N., Noureddine M., « Simulation et Base de connaissances pour l'analyse des défaillances », *Article accepté pour les Journées Doctorales LIO 2011*, Université d'Oran Es-Sénia, 31 mai 2011 et 01 juin 2011.

Dumas X., Pagetti C., Sagaspe L., Bieber P., Dhaussy P., «Vers la génération de modèles de sûreté de fonctionnement » *Journées FAC'2008*, Toulouse, 3-4 Avril 2008.

Hladik P., Peres F., Shi X., « Analyse d'un modèle AADL à l'aide de Pola », *10es journées Francophones AFADL*, Poitiers, 9-11 Juin 2010.

Hugues J., Singhoff F., « Développement de systèmes à l'aide d'AADL – Ocarina/Cheddar », *Tutoriel présenté à l'école d'été temps réel*, Paris, Septembre 2009.

Idiri R., Test d'applications AADL, Master 2, Recherche en informatique, Université de Bretagne Occidentale, Brest, Juin 2009.

Joshi A., Vestal S., and Binns P., « Automatic Generation of Static Fault Trees », *Séminaire sur la modélisation des systèmes fiables* (DSN 2007), (Edinburgh, UK), 2007.

Loiret F., Tinap : Modèle et infrastructure d'exécution orienté composant pour applications multi-tâches à contraintes temps réel souples et embarquées,Thèse de Doctorat en Informatique, Université des Sciences et Technologies de Lille, Mai 2008.

Mareschal C., « Adaptation d'un langage de description d'architecture à l'expression du comportement ; applications », *Journées FAC'2004*, CERT-ONERA, Toulouse, 9-10 Mars 2004.

Megdiche M., Sûreté de fonctionnement des réseaux de distribution en présence de production décentralisée, Thèse de Doctorat en Génie Electrique, INP de Grenoble, Décembre 2004.

Messaoudi M., « Analyse et évaluation des défaillances dans les systèmes spatiaux », Mémoire de Magister en Informatique, Département Informatique, USTO Oran, 24 Novembre 2010.

Mortureux Y., « Arbres de défaillance, des causes et d'événement », *Techniques de l'Ingénieur, SE4050*, 10 Octobre 2002.

Nedjari D., Noureddine M., Etude de fiabilité dans les applications multipoints, Rapport de Projet de Fin d'Etudes Master, Département Informatique, USTO Oran, Juin 2009.

Noureddine M., « Analyse des défaillances pour les systèmes d'extincteur à poudre », *6ème Conférence Internationale CPI'2009*, Fès (Maroc), 19-21 Octobre 2009.

Pagetti C., « module sureté de fonctionnement », *Support de cours, 3éme TR - option SE*, ENSEEIHT, 13 janvier 2010.

Peytavin A., « Glossaire de SdF », *Collection de la division SDF*, CENA/SDF, 1998.

Point G., AltaRica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement, Thèse de Doctorat en Informatique, Université de Bordeaux 1, Janvier 2000.

Rugina A., « Construction de modèles de sûreté de fonctionnement à partir du langage AADL », *7ème Congrès EDSYS*, Tarbes, 11 mai 2006.

Rugina A., Modélisation et évaluation de la sûreté de fonctionnement de AADL vers les réseaux de Petri stochastiques, Thèse de Doctorat, Institut National Polytechnique de Toulouse, 19 novembre 2007.

Torrente G., Bouissou M., « Méthodologie de développement de bases de connaissances pour la SdF avec l'environnement Open-Source VISUAL FIGARO », 16ème *Congrès MR-SdF, Lambda-Mu16*, Avignon, 7-9 Octobre 2008.

Vergaud T., « AADL un langage pour la modélisation et la génération d'applications », *Séminaire MeFoSyLoMa,* 14 Février 2005.

Vergaud T., Modélisation des systèmes temps-réel répartis embarqués pour la génération automatique d'applications formellement vérifiées, Thèse de Doctorat, École doctorale d'informatique, télécommunications et électronique de Paris, 1 Décembre 2006.

Villate N., « Manuel utilisateur de KB3 V3 », EDF DOC, Novembre 2005.

Villemeur A., « Sûreté de fonctionnement des systèmes industriels », *Collection de la Direction des Études et Recherches d'Électricité de France (EDF),* Eds Eyrolles, 1988.

Zalila B., Configuration et déploiement d'applications temps-réel réparties embarquées à l'aide d'un langage de description d'architecture, Thèse de Doctorat en Informatique et Réseaux, ENST de Paris , Novembre 2008.

Zwingelstein G., « Sûreté de fonctionnement des systèmes industriels complexes », *Techniques de l'Ingénieur, S 8 250v2 & S 8 251*, 10 Juin 2009.

Webographie

[site 1] : http://www.previnfo.net

[site 2] : http://innovation.edf.com/recherche-et-communaute-scientifique/logiciels/kb341214.html

[site 3] : http://www.apsys.eads.net/fr/17/Nos-Progiciels.html

[site 4] : http://sourceforge.net/projects/visualfigaro/files/

[site 5] : http://altarica.labri.fr/forge/embedded/altarica/downloads.html

Résumé. Ce mémoire de magister propose une démarche pour la modélisation orientée langage des systèmes, dans un contexte de sûreté de fonctionnement (SdF), et dans le but de permettre par la suite diverses analyses qualitatives et quantitatives des systèmes.

Notre démarche est basée sur un méta modèle regroupant les langages FIGARO et AltaRica, considérés à la fois comme complémentaires et incontournables dans les études de SdF. Nous intégrons des phases de modélisation et de validation des modèles obtenus, en utilisant les plates formes et outils dédiés à ces langages pour la génération des modèles classiques de SdF tels que les arbres de défaillances et les séquences d'évènements.

Nous avons validé notre démarche par une mise en œuvre expérimentale sur un système critique de détection d'incendie.

Mots-clés: Système, sûreté de fonctionnement, modèle, langage, FIGARO, AltaRica, arbre de défaillances, séquence d'évènements.

Abstract. This magister memory proposes an approach oriented language for modeling systems in a context of dependability, and in order to permit various qualitative and quantitative analysis systems.

Our approach is based on a meta model grouping the languages AltaRica and FIGARO, Considered as both complementary and indispensable in studies of dependability. We integrate the phases of modeling and validation of models obtained using the platforms and tools dedicated to these languages for the generation of the classic models of dependability as fault trees and event sequences.

We validated our approach with an experimental implementation on a critical system for fire detection.

Keywords: System, dependability, model, language, FIGARO, AltaRica, failure tree, event sequences.