Systèmes d'information industriels et

intégration des modèles pour l'Analyse de Risques

Zina Brik
APSYS
22 quai Galliéni
92100 Suresnes
France

Emmanuel ARBARETIER.

1. INTRODUCTION

Les approches de l'analyse systémique développées jusqu'à ce jour ont montré que tout système peut se ramener à un ensemble de constituants physiques humains, matériels, et logiciels susceptibles de se trouver dans différents états décrits de manière discrète ou continue, et interagissant entre eux. Ces interactions s'effectuent par l'intermédiaire de flux échangés entre constituants et dont l'évolution dans le temps peut se décrire à travers des équations d'états, des réseaux de transition, et des scripts de simulation faisant appel à des langages plus ou moins sophistiqués.

Lors de la conception, l'industriel doit maîtriser la fiabilité et la sécurité de ces systèmes devenus de plus en plus complexes. Pour cela il a besoin de traduire cette décomposition fonctionnelle et hiérarchique par un langage de modélisation formel qui lui permettra de résoudre le problème d'explosion combinatoire lorsqu'on modélise les aspects dysfonctionnels des systèmes, ainsi que d'effectuer des analyses quantitatives et qualitatives : les analyses qualitatives déterminent les séquences de défaillance des composants élémentaires qui mènent le système dans un état non souhaité; les analyses quantitatives permettent de calculer les différents indicateurs de SdF tel que les facteurs FMDS.

Cet article traite de la problématique de passage de connaissances technologiques ou physique vers différentes formes de représentations possibles reposant le plus souvent sur des langage et des conventions sémantiques : il s'agit de modèles réalisant toujours une vue abstraite du système et plus ou moins conformes avec la réalité du système en fonction des points de vue envisagés.

Les tâches d'études préliminaires à la conception, amènent souvent les ingénieurs de développement à élaborer des modèles, en utilisant des langages de modélisation de nature différente. Cet article a pour objet d'aborder la difficulté de récupération et d'exploitation de la connaissance dans un contexte industriel. Cette connaissance peut se présenter sous différentes formes :
- modèles de spécification externes tels qu'utilisés dans la réalisation de Cahiers des Charges Fonctionnel,
- modèles de conception préliminaire utilisés typiquement dans le cadre des tâches d'Analyse Fonctionnelle Interne,
- modèles de Sûreté de Fonctionnement générés systématiquement à partir des modèles précédents,
- modèles d'Analyse de Performance susceptibles d'être associés aux modèles de Sûreté de Fonctionnement précédents,

La modélisation de système dans le monde de l'ingénierie peut s'appuyer sur différents types de langages: langages plus ou moins appliqués, dont la vocation est orientée 'méthodologie de conception' ou 'quantification-évaluation' par rapport à l'utilisation d'un certain nombre de critères (indicateurs de Sûreté de Fonctionnement, mesures de performance...), ou encore 'simulation', voire 'simulation formelle'... L'industriel est donc souvent amené à récupérer différents types de modèles associés à différentes parties d'un même système, et à consolider ces modèles dans un même environnement. Ces différents modèles peuvent être traduits ou enrichis d'un langage à l'autre, afin de disposer d'un ensemble de connaissances homogènes associées à un système et représentées à travers un formalisme unique. Nous illustrerons les notions d'équivalence, de compatibilité et d'enrichissement à travers un exemple de traduction d'un modèle d'Analyse Fonctionnelle en un réseau de Pétri, puis nous effectuerons une synthèse de ces expériences de 'traduction' de modèles d'un formalisme vers un autre en termes de méthodologie d'une part, et d'opportunité d'autre part, sachant qu'une activité de modélisation n'est jamais un but en soi, et qu'elle est toujours subordonnée à la réalisation d'objectifs d'évaluation ou d'optimisation allant au-delà de la simple formalisation.

2. LANGAGES DE MODELISATION : DIVERSITE ET CONTINUITE

Le mot 'intégration' semble résumer l'un des principaux enjeux de l'industrie du prochain millénaire; mais alors qu'on l'applique habituellement aux méthodes d'organisation industrielle, à la planification des processus, aux architectures informatiques, il faudrait tout naturellement l'appliquer au processus de modélisation qui apparaît en filigrane de toute tâche d'étude en phase de développement de système.

Le premier type de modèle qui est toujours utilisé dans le cadre des phases préliminaires renvoie à l'Analyse Fonctionnelle. De nombreuses méthodologies ont été mises au point par les industriels et les universités, reposant sur des langages apparemment aussi différents que les formalismes 'AFNOR', 'FAST', 'RELIASEP', 'SADT', 'SART', 'MERISE',...,dépendant en général des types d'applications pour lesquelles ils ont été conçus. Il est toujours possible d'analyser d'un point de vue purement formel, leurs points communs, ainsi que leurs éléments de divergence, mais on sait de toutes façons, qu'ils ont été conçus dans un but commun: permettre de décrire pourquoi un système est conçu, ce que l'on en attend, et comment il est sensé se comporter de manière nominale.

Dans une même continuité de point de vue, la notion d'Analyse Fonctionnelle' peut être étendue à la notion de 'Simulation Fonctionnelle'; d'un point de vue statique, concernant exclusivement une information descriptive, nous pouvons passer à un point de vue dynamique de connaissance exploitable à travers divers algorithmes d'exploration. Toutes les équations des constituants du système doivent être formalisées, et traduire le comportement nominal du système. Ces équations sont spécifiées pour un ensemble donné de conditions d'environnement, mais la simulation dynamique ne prend pas en compte les aspects de dégradation des flux fonctionnels.

La classification des langages de modélisation se complique, lorsqu'on considère les processus de défaillance. En effet, il est clair qu'une approche d'Analyse Fonctionnelle pure ne peut prendre en considération à elle seule les processus de défaillance.

On aborde alors les langages de simulation dysfonctionnelle tel que le langage ALTARICA, à travers lesquels on a la possibilité de dégrader les exigences attendues du système (niveaux de performance requis), de prendre en considération les possibilités d'influence néfaste de l'environnement extérieur, et de formaliser les états de défaillance de tous les constituants à travers chacune de leur équation d'état.

On peut développer les caractéristiques de ces langages de simulation dysfonctionnelle fonctionnelle: ces langages peuvent par exemple conduire à distinguer deux types d'états logiques de défaillance: les états intrinsèques correspondant aux défaillances physiques, pour lesquels il est nécessaire de mettre en oeuvre une action de réparation, afin qu'il soit supprimé; les états fonctionnels, susceptibles de recouvrir les défaillances fonctionnelles, correspondant généralement à l'état physique provisoirement inacceptable d'une fonction, mais qui peut être supprimé à partir de modification des entrées du système, ou de l'introduction d'un autre mode opératoire.
En fait, les états de défaillance physique sont donnés par des bibliothèques ou banques de données de fiabilité, alors que les états de défaillance fonctionnelle sont donnés par soit la physique qualitative (assertion logique traitant du comportement d'une fonction telle que: très élevé, stable, impur...), soit une description quantitative donnant un spectre continu ou discret de valeurs possibles pour les caractéristiques quantitatives de ces fonctions (tension d'alimentation égale à 5V ou comprise entre 2 et 5V).
Viennent ensuite les modèles de Sûreté de Fonctionnement qui peuvent être déduits des langages précédents, à condition éventuellement d'être enrichis auparavant: diagrammes de fiabilité, arbres de défaillance, arbres d'événement, ou des modèles plus sophistiqués tels que chaînes de Markov, réseaux de Pétri, réseaux de probabilité. Par rapport à cette 'généalogie ' de modèles de Sûreté de Fonctionnement, on peut formuler les remarques suivantes: tout d'abord ils sont de niveau de sophistication inégal; certains sont plus riches que d'autres; d'autre part, il existe des passerelles d'un modèle à l'autre, dont on traitera à la fin de cet exposé; enfin, si les modèles d'Analyse Fonctionnelle et Dysfonctionnelle précédents, traitant de la connaissance profonde d'un système peuvent être considérés comme des 'modèles entrées', en ce sens où ils sous-tendent la formalisation de connaissances de base d'un système, les modèles de Sûreté de Fonctionnement, dans la mesure où ils traduisent souvent un point de vue partiel de ce système (privilégiant une mission particulière, ou un évènement redouté spécifique par exemple) peuvent être considérés comme des 'modèles sorties'; de plus, en fonction du niveau de 'sophistication' du langage d'Analyse Fonctionnelle de départ (qui peut être un véritable langage de simulation comportementale tel qu'ALTARICA qu'on détaillera plus loin), ces modèles de Sûreté de Fonctionnement peuvent être générés automatiquement...Ainsi, les AMDEC peuvent être générées automatiquement, à travers la simulation de toutes les défaillances élémentaires des constituants du système, les arbres de défaillances, à partir du traitement des relations de dépendance logique entre les défaillances intrinsèques des constituants, et les défaillances fonctionnelles, les diagrammes de fiabilité, à partir de la génération de la fonction d'état du système correspondant à ses missions nominales et dégradées.

3. GENERATION DES MODELES DE SURETE DE FONCTIONNEMENT

La problématique de passage du fonctionnement au dysfonctionnement dans le cas général est bien illustrée par les deux exemples suivants qui traitent de :

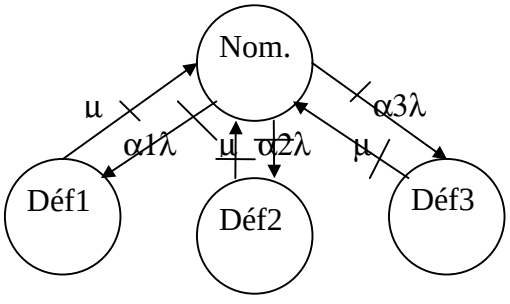
- l'obtention directe d'un réseau de Pétri à partir d'un modèle d'analyse fonctionnelle.
- l'enrichissement par dégradation progressive des automates de type ''state flow-state chart ''

3.1 GENERATION D'UN RESEAU DE PETRI A PARTIR D'UN MODELE D'ANALYSE FONCTIONNELLE

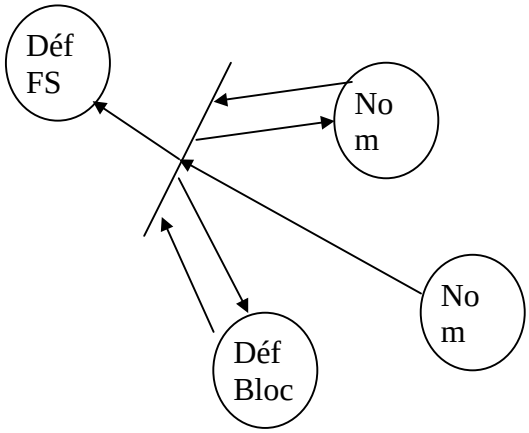
Partant d'un modèle d'Analyse Fonctionnelle, le modèle cible envisagé est alors de type Réseau de Petri Temporisé. Le mode d'exploitation des outils envisagés par la suite peut être soit analytique, soit simulatoire. Dans le premier cas, les transitions suivent une loi exponentielle. Dans le second cas, une liberté plus grande est laissée pour le choix des lois de transition (exponentielle, Weibull, gamma, lognormale, déterministe, uniforme...).
La plus-value des modèles cibles par rapport au modèle origine d'Analyse Fonctionnelle réside en la prise en compte des aspects suivants : dépendance entre événements, prise en compte des phénomènes de synchronisations et de la dynamique du système. Moyennant 'enrichissement' du modèle d'origine, il est également possible de modéliser les contraintes de maintenance (articles réparables ou non, nombre d'opérateurs disponibles, dépendance entre les temps de réparation et les modes de défaillance).
Lors du passage d'un modèle d'Analyse Fonctionnelle à un réseau de Pétri, la difficulté est d'identifier les places et les transitions du réseau de Pétri cible ; les places du réseau de Pétri seront les états des blocs fonctionnels et des fonctions ; quant aux transitions, elles seront de deux types : les transitions exprimant l'évolution des états intrinsèques de chaque éléments du système (en particulier les transitions de défaillance) et les transitions de causes à effets exprimant les liens entre les états des fonctions d'une même chaîne fonctionnelle.
Le *premier type de transition* correspond donc aux transitions entre états d'un même bloc fonctionnel. Il conduit à la génération d'un automate ayant son propre sous réseau de Pétri autonome :

- La transition entre un état nominal et un état dégradé suit une loi exponentielle de taux $\alpha.\lambda$ (ratio mode de l'état multiplié par le taux de défaillance du bloc), α étant égal à 1 dans le cas particulier de l'état de défaillance générale.

- La transition entre un quelconque état de défaillance et état nominal suit une loi exponentielle de taux μ (taux de réparation du bloc).



Les *transitions de causes à effets* sont celles qui expriment les interactions entre les différents sous-réseaux du modèle cible, qui correspondent quant à eux aux blocs du modèle d'Analyse Fonctionnelle d'origine. Ces transitions traduisent les dépendances entre les états des fonctions entrées des blocs et ceux de leurs fonctions sorties, tout au long des chaînes fonctionnelles :



Le sous réseau précédent exprime le fait que, une fonction entrée FE d'un bloc fonctionnel étant dans un état nominal, dès lors que ce bloc présente une défaillance, elle va transiter elle-même vers un état de défaillance. On a ainsi représenté une règle de traduction vers un réseau de Pétri d'une règle logique de dépendance du type 'et'. Il est facile d'obtenir cette règle de traduction pour une condition logique quelconque.
Suivant l'état de complétude ou le niveau de sophistication du modèle de départ, on obtient des réseaux de Pétri plus ou moins 'riches' :

- nombre d'états dysfonctionnels par bloc fonctionnel et par fonction : on peut avoir un modèle pur d'Analyse Fonctionnelle, auquel cas, la notion de défaillance est binaire pour les blocs, ou bien un modèle traitant le dysfonctionnel de manière plus détaillée
- temporisation des transitions : les transitions peuvent être déterministes à durée nulle (causes-conséquences) ou aléatoires (défaillances) suivant des lois quelconques ; la notion de 'travail' peut être également prise en compte au niveau des transitions
- des notions de politique de maintenance peuvent être également renseignées dans le modèle initial : les taux de réparation peuvent être différenciés par mode de défaillance et suivre des lois de distribution log-normal ou quelconques
- on peut également enrichir le modèle initial d'Analyse Fonctionnelle par des notions complémentaires de messages ou de ressources qui trouvent directement leur formalisation dans le réseau de Pétri cible ; ces notions peuvent être rattachées aux transitions des deux types précédents.

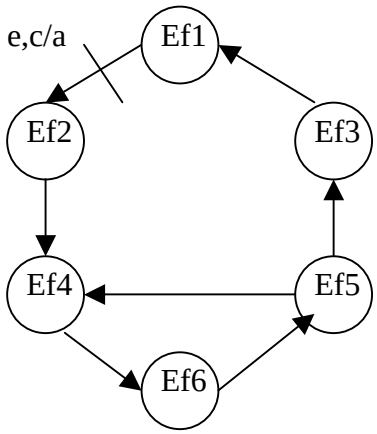
3.2 GENERATION D'AUTOMATES A ETATS FINIS A PARTIR D'UNE CONNAISSANCE FONCTIONNELLE DYSFONCTIONNELLE

Au delà des langages d'Analyse Fonctionnelle supportés pour la plupart par des représentation du type Bloc Diagramme Fonctionnel (BDF), la connaissance comportemental d'un système, peut, à un niveau de sophistication supérieur, se formaliser à travers l'utilisation d'automates à états finis synchronisés.

Nous allons voir dans ce paragraphe, sur l'exemple des automates à états finis, le processus délicat de ''passage'', ''enrichissement'' d'une connaissance fonctionnelle à une connaissance dysfonctionnelle.

L'automate fonctionnel est pour un process, le graphe d'état qui rend le mieux compte des règles de comportement de l'automate en mode nominal:

Les Ef_i représentent les états fonctionnels

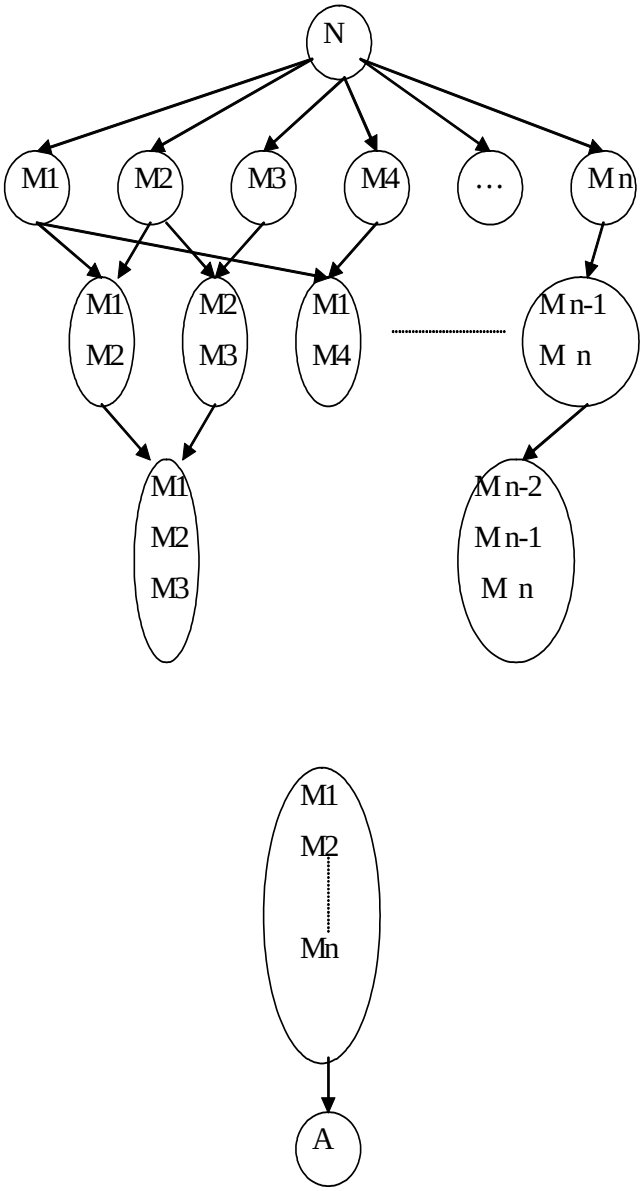


Chaque transition est du type e,c/a, c'est à dire événement, condition , action :
- un événement ou une condition est une expression logique sur des états des fonctions entrées du bloc
- une action est une liste d'états de fonctions sorties du bloc.

On n'a pas décidé de prendre en compte les aspects :
- ressources
- variables continues

A ce stade, les notions d'événement et de condition semblent équivalentes, elles prendront un sens différent au niveau de la simulation.

L'automate théorique des états intrinsèques, représentant les modes de défaillance du système se définirait de la manière suivante:



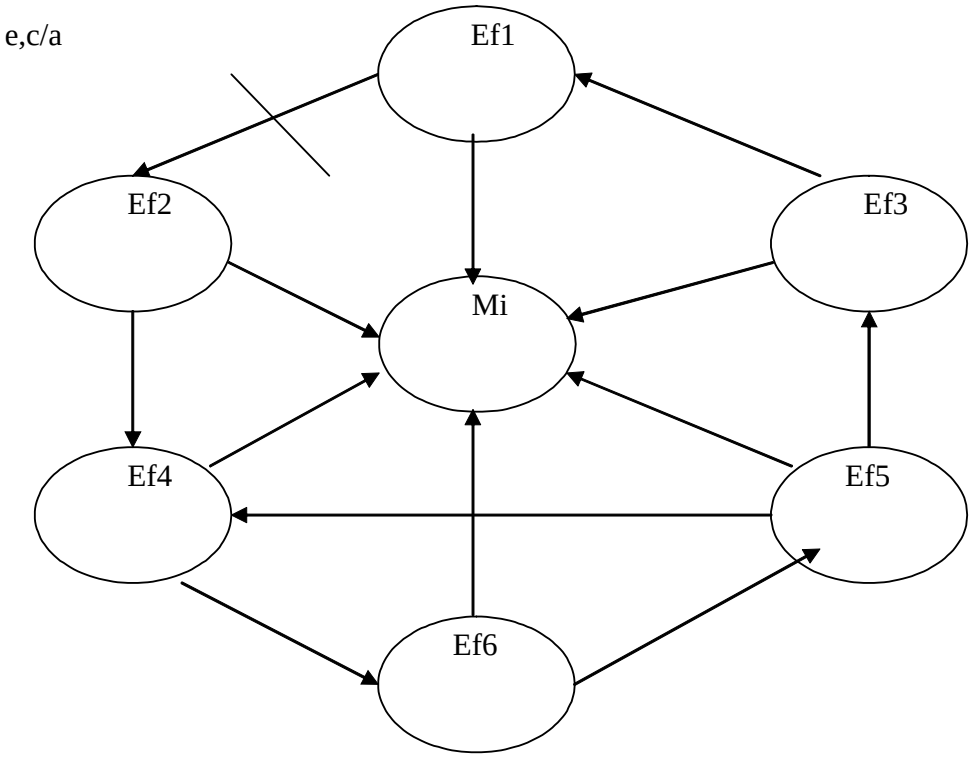
N, M1, M2, M3…représentent les modes de défaillance.

En réalité, il suffit d'envisager la dégradation de l'automate fonctionnel nominal par rapport à chacune des occurrences simples des états intrinsèques.

On obtient alors l’automate global du process qui n’est autre qu’une imbrication des deux automates fonctionnel et intrinsèque.

Selon que chaque état intrinsèque admet des conséquences franches ou intermédiaires sur l’automate fonctionnel, il va falloir renseigner deux types de graphes.

L'automate global des états peut donc avoir une première forme mettant en évidence le caractère absorbant des états intrinsèques par rapport à tout état fonctionnel :



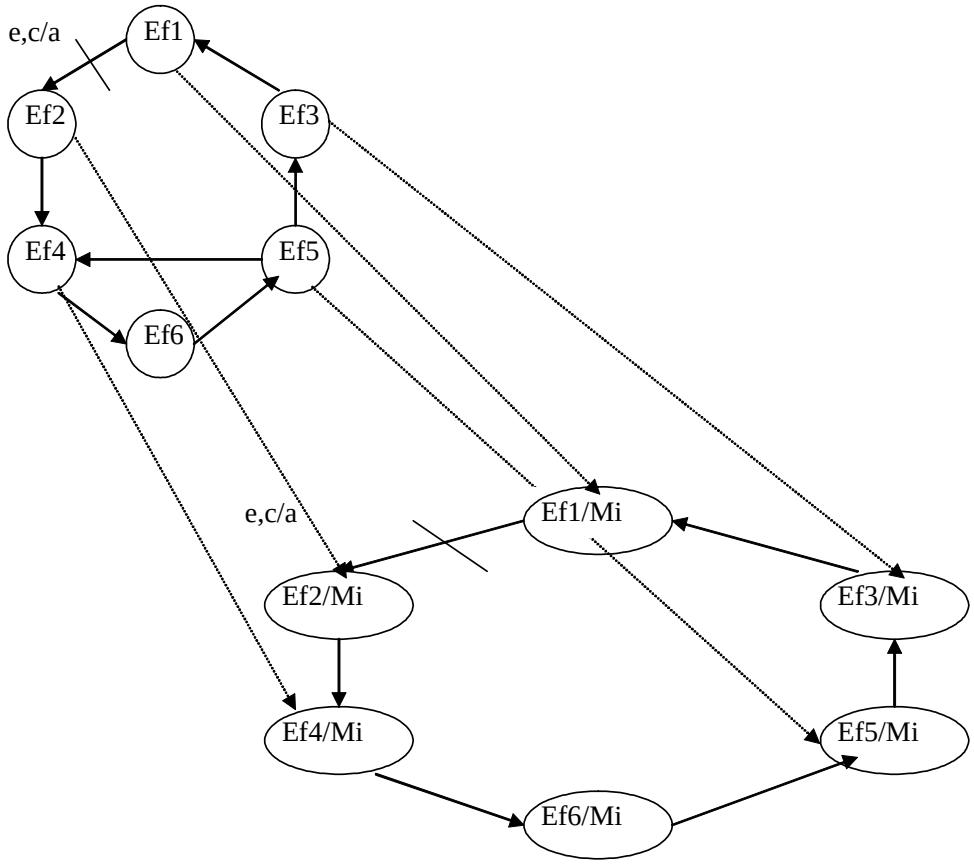
Ce cas, qui est le plus fréquent se présente lorsque :
- l’état intrinsèque Mi rend impossible l’obtention de l’un quelconque des état fonctionnels
- éventuellement certains modes de défaillances ne sont accessibles qu’à partir d’états fonctionnels précis, auquel cas, certaines transitions sont à supprimer

Il faut représenter au centre du graphe tous les états Mi à partir desquels, les états fonctionnels sont annulés.

Une autre forme d'automate global suppose qu'un mode Mi est compatible partiellement ou totalement avec l'automate fonctionnel : il s'agit de la formalisation pure et simple des modes dégradés.

Il faut alors étudier pour chaque état intrinsèque Mi :

- la robustesse de l'automate fonctionnel par rapport à l'état intrinsèque Mi : certaines transitions peuvent ne plus être possible, certains états ne plus exister…
- les possibilités de transition à partir de l'automate fonctionnel nominal vers chacun des états de l'automate dégradé



4.3 VERS UNE INTEGRATION COMPLETE DES ASPECTS FONCTIONNELS ET DYSFONCTIONNELS : LE LANGAGE ALTARICA

Le domaine de la SdF possède plusieurs outils d'analyse de systèmes. Chaque outil est basé sur un modèle spécifique tel que les arbres de défaillance, réseaux de Pétri, diagramme de fiabilité et autres. Ces derniers ne sont pas assez descriptifs de la réalité du système car ils ont l'inconvénient de créer une certaine distance entre le modèle et le système lui même (forte abstraction). Cela est dû au fait que chaque modèle est dédié à étudier une caractérisation partielle du système selon un point de vue précis.

En fait, idéalement, les ingénieurs auraient besoin d'un langage complet capable de traduire des hypothèses et mécanismes d'évolution extrêmement variés et sophistiqués, un langage capable de décrire l'aspect fonctionnel et dysfonctionnel d'un système d'une manière précise et d'effectuer toutes les analyses qualitatives et quantitatives nécessaires aux analyses de performance y compris la SdF.

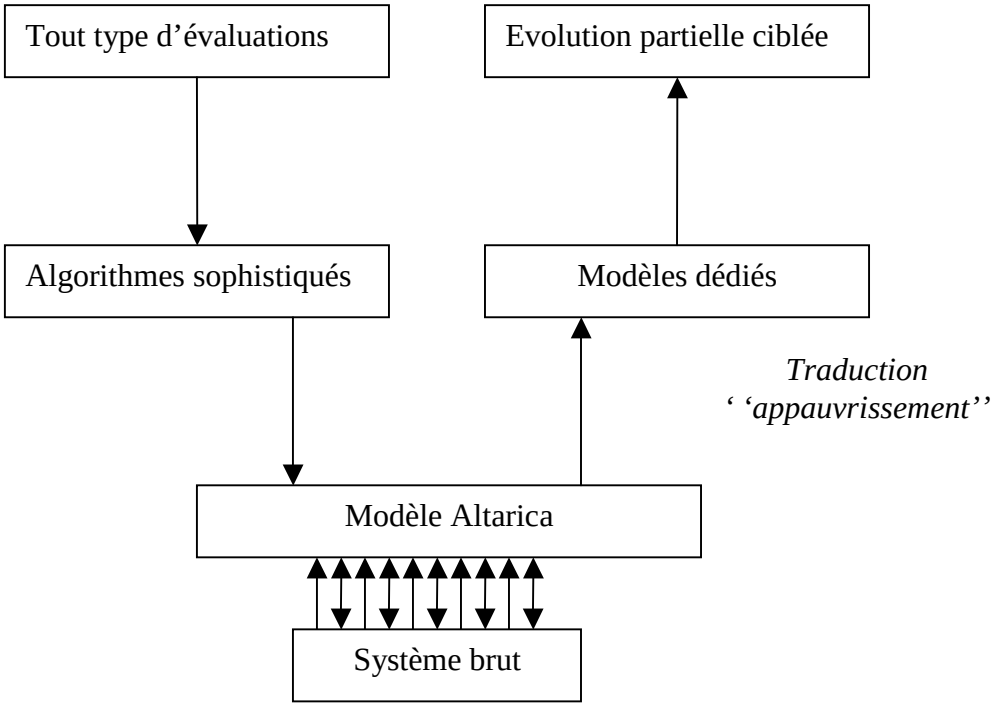
ALTARICA est un langage susceptible de répondre à ces critères et exigences. C'est un langage basé sur le formalisme des automates à états finis, et qui à travers sa richesse, offre une grande souplesse dans la description du système et reste proche de la connaissance technologique ou physique relatif à ce dernier. Il nous permet d'intégrer des contraintes plus sophistiquées, ce qui n'était pas toujours possible avec les autres langages cités précédemment.

Le langage ALTARICA offre aussi l'avantage et la possibilité de compilation vers des modèles existant déjà. On peut citer comme exemple les arbres de défaillance, les réseaux de pétri,...sans oublier que presque toutes les constructions syntaxiques ont une représentation graphique analogue.

L'atelier de simulation pour l'ingénierie système *''SIMFIA V2''*, travaille actuellement sur le développement d'outils de simulation, qui à partir d'une description AltaRica génèrent et décrivent le comportement dynamique du système en traçant les successions d'événement entraînant un état redouté de ce dernier.

Ces séquences ont la particularité de mener le système dans un état critique non souhaité. L'obtention de ces séquences permet une étude qualitative des scénarios de défaillance et la prise de décision sur l'amélioration des parties les moins fiables du système.

Sur le schéma suivant, on voit bien où se trouve la problématique d'utilisation et de gestion des modèles dans l'industrie. Il faut malheureusement souvent choisir entre des modèles à haut niveau d'abstraction, performant à exploiter par rapport à certain points de vue d'évolution, et des modèles comportementaux, proche de la réalité physique (ou logicielle), denses et lourds à exploiter quel que soit le point de vue à traduire, obligé notamment à recourir à des algorithmes de simulation sophistiqués.



LES OBJETS ALTARICA

Composant : Un système est constitué de composants. Dans la terminologie AltaRica un composant est situé au plus bas niveau dans la hiérarchie décrivant le système.

Etat : Les états d'un composant sont décrits par un ensemble de variables dites variables d'état. Un état d'un composant est alors une évaluation de ces variables.

Flux : ils informent partiellement le composant sur l'état de son environnement. Ces mêmes flux lui permettent de transmettre une information vers l'extérieur. Les variables dites de flux prennent leurs valeurs dans un domaine spécifié par le modélisateur (booléens, entiers, chaînes de caractères).

Evénement : d'un point de vue syntaxique, un événement d'un composant est simplement un nom ou une étiquette dont l'interprétation est laissée à la charge du modélisateur. Quand un événement se produit l'état du composant est modifié.

Interface d'un composant : c'est l'ensemble constitué des événements et des flux du composant.

Assertion : c'est une contrainte sur les variables du composant. Elle représente la relation existant entre l'état du composant et la valeur de ses flux.

Transition : Les événements modifient l'état d'un composant, ces changement sont appelés transitions

Nœud AltaRica : c'est une boite noire considéré depuis son environnement, il se comporte comme un composant et interagit par l'intermédiaire de son interface. La hiérarchie intrinsèque du nœud est totalement masquée à son environnement.

Hiérarchie : le langage autorise une décomposition fonctionnelle et hiérarchique du système. La forme textuelle de la hiérarchie est introduite par la rubrique **sub** dans la définition d'un modèle composant.

Exemple :

On considère le système constitué d'une source, d'un Objectif ou cible (target), unité de traitement, et une équipe de réparation. On suppose que la source, l'unité et la cible sont sujettes à défaillance et que seule l'unité peut être réparée. Supposant aussi que les flux booléens sont utilisés pour décrire la production. Un code AltaRica pour un tel système peut s'écrire comme suit :

domain stDomain = {marche, panne} ;

node source
```
  state s:stDomain;
  flow  outProd:bool:out;
  event défaillance ;
  trans
    (s= marche) |- défaillance -> s:= panne;
```

```
  init
    s:= marche;
  assert
    outProd =(s= marche);
  edon
```

node objectif

```
  state s:stDomain;
  flow  inProd:bool:in;
  event défaillance ;
  trans
    (s= marche) |- défaillance -> s:= panne;
  init
    s:= marche;
edon
```

node unité

```
  state s:{ marche, panne, réparation};
  flow  inProd:bool:in; outProd:bool:out;
  event défaillance,début de réparation, fin de réparation;
  trans
    (s=marche)     |- défaillance         -> s:=panne;
    (s=panne)      |- début de réparation -> s:=réparation;
    (s=réparation) |- fin de réparation   -> s:=marche;

  assert
    outProd = inProd and (s= marche);
  init
    s:= marche;
edon
```

node équipe de réparation

```
  state s:{occupé, libre};
  event début de travail, fin de travail;
  trans
    (s=libre)  |- début de travail   -> s:=occupé;
    (s=occupé) |- fin de travail     -> s:=libre;

  init
    s:=occupé;
edon
```

node main

```
  event début de réparation, fin de réparation;
  sub  S:source ; O:objectif ; U:unité ; R:équipe de réparation ;
  sync
     <Début de réparation, U.Début de réparation, R.début de travail>,
     <fin de réparation, U.fin de réparation, R.fin de travail> ;
  assert
    U.inProd = S.outProd,
    O.inProd = U.outProd,

Edon
```

5. CONCLUSION

L'intégration des systèmes d'information qui est le maître mot des enjeux organisationnels doit s'accompagner de l'intégration des modèles de connaissance qui sont la véritable matière première des ingénieurs de conception ; ce sont eux qui expriment véritablement le savoir-faire d'une entreprise, et il importe de pouvoir les capitaliser, réutiliser, modifier et exploiter dans les meilleures conditions possibles d'accès, de temps de réponse, et de lisibilité.
Malheureusement, la normalisation des modèles de connaissance (normes STEP, PDES : Product Data Exchange Standard) n'est pas encore bien diffusée, et l'aide à la décision dans le domaine de la conception, doit souvent s'appuyer sur des modèles qui sont sans cesse regénérés, actualisés, adaptés, mais rarement gérés en configuration.
Au delà de la maîtrise d'un système d'information parfaitement structuré, l'industriel doit donc pouvoir formaliser les règles lui permettant de construire rapidement les modèles évolués construits à partir d'agrégats d'information par trop souvent disparates et dispersés dans ses bases de données.
Il doit identifier les supports de modélisation et de traitement dont il a besoin pour répondre à ses préoccupations de simulation, évaluation, optimisation et prise de décision, et systématiser le processus d'utilisation de ces modèles afin de répondre à ce besoin au moindre coût et dans les meilleurs délais.

Les exemples présentés dans cet exposé montrent que quelque soit le formalisme adopté, la connaissance de conception à ‘‘capturer’’ reste toujours de même nature et il importe que des besoins spécifiques d'exploitation de cette connaissance ne conduisent pas à appauvrir les structures d'information d'origine.

Entre un référentiel de modèles de conception très dense exprimé dans un langage normalisé réunissant le maximum de possibilités d'hypothèses de modélisation, et un système de transformation ‘‘disparate’’ traitant en eux même certain aspects du système à modéliser ; la question doit se poser ; peut-on gager que les réponses dépendent de l'organisation en place (Système d'Information (SI), acteurs utilisant cette information) et du type d'utilisation de cette information ainsi que des modèles correspondants? La question reste ouverte.