Contribution à l'analyse des systèmes pilotés par calculateurs : extraction de scénarios redoutés et vérification de contraintes temporelles

Malika Medjoudj

▶ **To cite this version:**

Malika Medjoudj. Contribution à l'analyse des systèmes pilotés par calculateurs : extraction de scénarios redoutés et vérification de contraintes temporelles. Automatique / Robotique. Université Paul Sabatier - Toulouse III, 2006. Français. NNT : . tel-00082568

HAL Id: tel-00082568

https://theses.hal.science/tel-00082568

Submitted on 28 Jun 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse

Préparée au

Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

En vue de l'obtention du titre de

Docteur de l'Université Paul Sabatier de Toulouse

Spécialité : **Systèmes Industriels**

Par

Malika MEDJOUDJ
Ingénieur UMMTO

Contribution à l'analyse des systèmes pilotés par calculateurs : Extraction de scénarios redoutés et vérification de contraintes temporelles

Soutenance le jeudi 9 mars 2006 devant le jury :

Hamid DEMMOU	Directeur de thèse
Robert VALETTE	Co-Directeur de thèse
Yves DUTUIT	Rapporteur
Pierre Etienne LABEAU	Rapporteur
Mario PALUDETTO	Examinateur
Sarhane KHALFAOUI	Examinateur

Thèse

Préparée au

Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

En vue de l'obtention du titre de

Docteur de l'Université Paul Sabatier de Toulouse

Spécialité : **Systèmes Industriels**

Par

Malika MEDJOUDJ
Ingénieur UMMTO

Contribution à l'analyse des systèmes pilotés par calculateurs : Extraction de scénarios redoutés et vérification de contraintes temporelles

Soutenance le jeudi 9 mars 2006 devant le jury :

Hamid DEMMOU	Directeur de thèse
Robert VALETTE	Co-Directeur de thèse
Yves DUTUIT	Rapporteur
Pierre Etienne LABEAU	Rapporteur
Mario PALUDETTO	Examinateur
Sarhane KHALFAOUI	Examinateur

À mes parents pour leur soutien et leur confiance

À ceux qui m'aiment et qui attendent avec impatience ma réussite

En espérant être à la hauteur de leurs attentes

Remerciements

Les travaux présentés dans ce mémoire ont été effectués au Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS).

Je tiens à remercier Monsieur Jean-Claude Laprie, Directeur de recherche de classe exceptionnelle et Monsieur Malik Ghallab, Directeur de recherche CNRS, qui ont assuré la direction du LAAS-CNRS depuis mon entrée, de m'avoir permis d'accomplir mes travaux dans ce laboratoire.

Je remercie également Messieurs Robert Valette, Directeurs de recherche CNRS et Mario Paludetto, professeur de l'Université Paul Sabatier (UPS) de Toulouse, responsables respectifs des groupes de recherche Organisation et Conduite des Systèmes Discrets (OCSD) et Ingénierie Systèmes et Intégration (ISI) au sein des quels j'ai effectué mes travaux de recherche pour leur accueil et pour la confiance qu'ils m'ont accordée.

J'exprime ma profonde reconnaissance à Monsieur Robert Valette, Directeur de Recherche CNRS, et Monsieur Hamid Demmou, Maître de Conférences à l'UPS de Toulouse, qui ont dirigé mes travaux de thèse, pour m'avoir encadré et pour leurs conseils.

J'exprime ma profonde gratitude à Monsieur Mario Paludetto, Professeur de l'UPS, pour l'honneur qu'il me fait en présidant mon jury de thèse, ainsi qu'à :

- M. Yves DUTUIT, Professeur Université Bordeaux 1
- M. Pierre-Etienne LABEAU, Professeur à l'Université Libre de Bruxelles
- M. Sarhane KHALFAOUI, Chef de Projet PSA Peugeot Citroën
- M. Robert VALETTE, Directeur de Recherche au CNRS
- M. Hamid DEMMOU, Maître de Conférences à l'Université Paul Sabatier de Toulouse

pour l'honneur qu'ils me font en participant à mon jury de thèse. Je remercie tout particulièrement Monsieur Yves DUTUIT et Monsieur Pierre-Etienne LABEAU qui ont accepté la charge d'être rapporteurs. Je les remercie pour leurs remarques et conseils qui ont permis d'améliorer la qualité de ce manuscrit. Je voudrai remercier Ayda Saidane pour ses conseils de programmation et sa relecture attentive qui a permis d'améliorer ce manuscrit. Mes remerciements vont naturellement à tous les membres du groupe ISI, permanents, doctorants et stagiaires ainsi que la secrétaire de notre groupe Eliane. Je remercie particulièrement le professeur Sahraoui Abdelkader pour son soutien et ses conseils.

Dans ces moments importants, je pense très fort à ma famille en commençant par mes parents, Tassadit et Mohamed Akli, mes frères et sœur : Lynda, Mourad, Karim et Moukrane ainsi que ma grand mère. Je remercie en particulier ma tante Ghenima et sa famille Allam : Rabah, Tina et Massil, de m'avoir accueilli à mon arrivé à Toulouse ainsi que mon frère Mourad. Je remercie tous mes proches pour leur soutien : Aldjia, Ouiza et Kahina.

Je ne peux ne pas mentionner les gens qui m'ont soutenu, conseillé et aidé pendant toutes ses années : Ayda, Raouda, Carmen-luisa et Siba pour leur présence dans les moments les plus difficiles. Merci à toi Hamid pour ton aide et soutien ces derniers mois. Je remercie mon amie de toujours Lynda pour son amitié et soutien. Un grand merci à tous mes amis Toulousains, qui ont toujours été là par leur présence ou pensée. Que ceux que j'oublie me pardonnent, Stéphanie, Dalila, Carmen, Eduardo, Mohamed, Aimed, Dan, Frencesco, Angel, El Abbassia, Carmen-lopez, Tarik, Souad, Badia, Madona, Hani, Adel, Mina, Salam, Iryna, Jihad, Nadia, Caroline, Nahida, Fadia, Rania, Hakim, Taoufik, Ryma...

Je remercie également tous ceux qui ont contribué de près ou de loin au parachèvement de ce travail de thèse, soit par leur savoir scientifique ou par leur amitié.

Table des matières

Table des figures

Introduction générale

Les systèmes pilotés par calculateur sont des systèmes énergétiques (mécaniques, hydrauliques ou électriques) commandés et contrôlés par un (ou plusieurs) calculateur (s) (informatiques et électroniques). Ce calculateur utilise des informations fournies par des capteurs pour établir la commande à appliquer au système piloté au moyen d'actionneurs. Ces systèmes appelés plus généralement embarqués sont utilisés dans le domaine de la défense, du spatial, du nucléaire (contrôle des centrales nucléaires) et dans les secteurs automobile et avionique (systèmes mécatroniques, calculateurs de vol, etc.). La flexibilité logicielle et matérielle de ces systèmes a permis une intégration progressive de l'électronique dans les secteurs automobile et avionique ce qui a amélioré le confort et les services rendus. Toutefois, cela a complexifié la conception des systèmes pilotés par calculateurs, ce qui rend difficile la maîtrise de leur fiabilité. Par ailleurs, la phase de conception doit être rapide et peu coûteuse (le moins de prototypes possible, le plus tard possible) avec un niveau de sécurité garanti. De plus, les ressources en moyens matériels étant limitées, pour des raisons de coûts et de mise en œuvre, les concepteurs évitent au maximum les redondances matérielles.

Des études de sûreté de fonctionnement réalisées dès la phase de conception permettent une meilleure maîtrise des risques et de la fiabilité des systèmes conçus. En effet, les points faibles mis en évidence lors de l'évaluation du niveau de sûreté des systèmes permettent aux concepteurs de spécifier des stratégies de pilotage et des modes de reconfiguration avant les premiers essais sur un prototype réel. Les systèmes pilotés par calculateurs sont hybrides: la dynamique continue est associée à la partie énergétique et la dynamique discrète est liée à la commande numérique et à l'existence d'événements discrets (défaillances, dépassements de seuils). L'étude de la sûreté de fonctionnement de tels systèmes doit nécessairement tenir compte des interactions existantes entre leurs paramètres physiques (température, pression, vitesse…) et le dysfonctionnement de leurs composants. Les méthodes classiques de la sûreté de fonctionnement, comme les arbres de défaillances sont insuffisantes pour de tels systèmes complexes et hybrides car ils sont dynamiques. La sûreté de ces systèmes doit tenir compte du temps et de l'ordre d'apparition des événements. Pour résoudre le problème de la ***rareté*** de ces scénarios auquel sont exposées les méthodes basées sur la simulation, des techniques d'accélération de la simulation ont été développées et largement utilisées, avec succès, dans l'ingénierie nucléaire notamment. Les méthodes d'analyse de modèles à événements discrets (automates, Réseau de Petri) ont leur contribution dans ce domaine mais l'utilisation de graphe d'accessibilité est vite confrontée au problème d'explosion combinatoire.

Un moyen d'évaluer la sûreté de fonctionnement des systèmes hybrides complexes est la ***recherche des scénarios redoutés*** menant à un ***état catastrophe.*** Comme nous l'avons mentionné, l'analyse qualitative visant la mise en évidence des scénarios critiques dans les systèmes pilotés par calculateurs est confrontée au problème de l'explosion combinatoire du nombre d'états du

graphe d'accessibilité à cause de la rareté de ces scénarios. Afin de contourner ce problème, [Khalfaoui 03] dans sa thèse entre PSA et le LAAS a utilisé directement le modèle Réseau de Petri (RdP) pour extraire les scénarios redoutés sans générer le graphe d'accessibilité associé et la logique linéaire pour représenter les modèles RdP et en extraire des scénarios. L'avantage est que la logique linéaire permet de construire un ordre partiel de franchissement des transitions et focalise la recherche sur les parties intéressantes du modèle pour l'analyse de la sûreté de fonctionnement. Cette approche est basée sur l'équivalence entre l'accessibilité dans les RdP et la prouvabilité du séquent associé en logique linéaire. La nature hybride et dynamique des systèmes mécatroniques est respectée par le choix d'une modélisation associant RdP et équations différentielles. Le modèle RdP décrit le fonctionnement nominal, les défaillances et les mécanismes de reconfiguration. Les équations différentielles représentent l'évolution des variables continues de la partie énergétique du système.

Partant de l'état redouté, il est possible de revenir en arrière à travers la chaîne des relations de cause à effet et d'extraire tous les scénarios possibles menant vers l'état dangereux (critique). Chaque scénario est donné sous la forme d'un ordre partiel entre les évènements nécessaires à l'apparition de l'évènement redouté ce qui diffère d'un arbre de défaillance qui donne un ensemble de combinaisons statiques des états partiels nécessaires pour l'obtention de l'état redouté. L'inconvénient de cette approche est que du fait qu'elle opère uniquement sur l'aspect discret du modèle, de nombreux scénarios incohérents vis-à-vis de la dynamique continue sont générés. De plus l'ordre d'occurrence des événements n'est pas pris en compte et les scénarios générés ne sont pas minimaux.

Objectif de la thèse

Le formalisme RdP PTDS choisi dans les travaux de [Khalfaoui 03] modélise au mieux l'aspect discret et continu des systèmes pilotés par calculateurs. Son approche basée sur l'extraction des scénarios directement à partir d'un modèle RdP du système est intéressante et semble bien adaptée pour une analyse qualitative de la sûreté de fonctionnement des systèmes pilotés par calculateurs d'un point de vue fiabilité dynamique. C'est pourquoi nous avons choisi de poursuivre sur la voie qu'il a laissé ouverte.

Nous avons repris cette approche, en précisant au mieux le concept de scénario et en particulier de scénario minimal. Cela nous a permis d'élaborer une nouvelle version de l'algorithme qui tient partiellement compte de l'aspect continu et plus particulièrement des instants auxquels les seuils associés à certaines transitions dans le modèle RdP sont atteints. Cela permet de déterminer plus précisément les conditions exactes de l'occurrence de l'événement redouté ; ce qui pousse le système à quitter son fonctionnement normal et à évoluer vers l'état redouté. Cette nouvelle approche permet d'éliminer des scénarios incohérents vis-à-vis de la dynamique continue du système. L'aspect continu dans nos travaux est pris en compte par des abstractions temporelles.

D'autre part, la notion de scénario pouvant être également très utile dans le cadre de la vérification de propriétés, nous nous sommes intéressés à l'applicabilité de notre approche pour vérifier des propriétés temporelles comme l'obtention de la durée maximale d'un scénario ou celle du temps maximal pouvant s'écouler entre deux événements comme deux commandes successives.

L'automatisation de toutes les étapes de notre approche nous a paru indispensable dans le cas des systèmes complexes où le risque d'erreur humaine est très important. C'est pourquoi, nous avons développé un outil ESA_PetriNet (Extraction & Scenarios Analyser by PetriNet model) qui permet d'extraire les scénarios critiques qui mènent vers l'état redouté à partir d'un modèle Réseau de Petri temporel et de vérifier certaines propriétés des systèmes pilotés par calculateurs.

Plan du manuscrit

Ce manuscrit est composé de six chapitres. Nous introduirons dans le premier chapitre les notions relatives à la sûreté de fonctionnement, les systèmes pilotés par calculateurs et leur nature hybride. Puis nous ferons un état de l'art des méthodes de sûreté de fonctionnement adaptées à des systèmes complexes, leurs limites et les différentes méthodes de modélisation de ces systèmes hybrides. Nous présentons les techniques de vérification et un tour d'horizon des principaux travaux qui, comme dans notre cas, ont proposé des approches pour la conception des systèmes sûrs de fonctionnement.

Dans le deuxième chapitre, nous étudierons différents formalismes, à base de réseaux de Petri, qui ont été étendus afin de prendre en compte l'aspect temporel et hybride des systèmes complexes. Nous introduirons également dans ce chapitre un bref rappel de la logique linéaire ainsi que la notion de scénarios, sur lesquels nos travaux seront basés pour l'évaluation de la sûreté de fonctionnement et la vérification des systèmes pilotés par calculateurs.

Dans le troisième chapitre, nous présentons une approche pour une analyse qualitative de la sûreté de fonctionnement des systèmes pilotés par calculateurs (mécatroniques) d'un point de vue fiabilité dynamique. Nous proposons une nouvelle version de l'algorithme qui tient compte partiellement de l'aspect continu du système hybride étudié. Les scénarios obtenus caractérisent comment le système quitte le fonctionnement normal pour évoluer vers l'état redouté en déterminant la suite d'actions et de changements d'états conduisant à l'état critique. La méthode et l'algorithme sont illustrés sur un cas d'étude. Nous allons montrer comment un grand nombre de scénarios cohérents vis-à-vis de la partie discrète du modèle hybride mais incohérents vis-à-vis de sa partie continue ne sont plus générés.

Dans le quatrième chapitre, nous présentons l'automatisation de toutes les étapes de notre algorithme dans le prototype *ESA_PetriNet* (Extraction & Scenarios Analyzer by PetriNet model). Cet outil que j'ai développé en Java permet à partir d'un modèle Réseau de Petri temporel d'extraire les scénarios critiques qui mènent vers l'état redouté.

Dans le cinquième chapitre nous appliquons l'outil ESA_PetriNet pour la génération des scénarios redoutés dans un système industriel de taille complexe. Il s'agit d'un système de contrôle des trains d'atterrissage d'un avion.

Dans le sixième chapitre nous adaptons notre approche d'extraction des scénarios redoutés dans les systèmes hybrides à la vérification de certaines propriétés temporelles. Ces propriétés peuvent être la durée maximale d'un scénario. Nous appliquerons notre approche sur l'exemple des trains d'atterrissage d'un avion pour vérifier la durée maximale d'un scénario de maintien d'une commande. Dans un deuxième exemple, les contraintes temporelles que nous allons vérifier

concernent la durée entre deux commandes. Nous utilisons notre outil pour la vérification de la durée entre deux commandes dans un système de calculateurs de vol qui est assez complexe. Enfin, en conclusion, nous donnerons les perspectives de notre travail.

Chapitre 1 : Sûreté de fonctionnement des systèmes pilotés par calculateurs

1.1. Introduction

Ce chapitre est consacré à la présentation des concepts de base de la sûreté de fonctionnement, ainsi que des différentes notions permettant de préciser le cadre et le contexte de nos travaux. Nous nous intéresserons aux systèmes pilotés par calculateurs qui font partie des systèmes hybrides, notamment le cas des systèmes mécatroniques. Puis, nous ferons un état de l'art des méthodes de sûreté de fonctionnement adaptées à des systèmes complexes, leurs limites et les différentes méthodes de modélisation de ces systèmes hybrides. Nos travaux seront placés dans le cadre de la fiabilité dynamique et plus précisément dans le cadre de la recherche de scénarios menant à des états redoutés. Nous nous intéresserons ensuite aux techniques de vérification puisque, comme nous le verrons, il y a de fortes relations entre la recherche de scénarios redoutés et la vérification des contraintes temporelles des systèmes pilotés par calculateurs. Nous terminerons par un tour d'horizon des principaux travaux qui, comme dans notre cas, ont proposé des approches pour la conception des systèmes sûrs de fonctionnement.

1.2. Sûreté de fonctionnement

Cette section rapporte les concepts de base de la sûreté de fonctionnement tels qu'ils sont définis dans [Laprie et al 96], [Laprie et al 04] et [Villemeur 88], travaux dans lesquels les lecteurs trouveront une vue plus détaillée de l'ensemble du domaine.

1.2.1. Concepts de la sûreté de fonctionnement

La sûreté de fonctionnement (SdF) peut être définie comme la science des défaillances [Villlemeur 88]. Elle inclut ainsi leur connaissance, leur évaluation, leur prévision, leur mesure et leur maîtrise. Au sens strict, elle est l'aptitude d'une entité à satisfaire une ou plusieurs fonctions requises dans des conditions données.

Pour [Laprie et al 04], la sûreté de fonctionnement d'un système est son aptitude à délivrer un service de confiance justifiée. Cette définition mettant l'accent sur la justification de la confiance, cette dernière peut être définie comme une dépendance acceptée, explicitement ou implicitement. La dépendance d'un système vis-à-vis d'un autre système est l'impact, réel ou potentiel, de la sûreté de fonctionnement de ce dernier sur la sûreté de fonctionnement du système considéré.

Selon la ou les applications auxquelles le système est destiné, l'accent peut être mis sur différentes facettes de la sûreté de fonctionnement, ce qui revient à dire que la sûreté de fonctionnement peut être vue selon des aspects différents mais complémentaires, qui permettent de définir ses *attributs* :

- le fait d'être prêt à l'utilisation conduit à la *disponibilité* ;
- la continuité du service conduit à la *fiabilité* ;
- l'absence de conséquence catastrophique pour l'environnement conduit à la *sécurité-innocuité* ;
- l'absence de divulgations non-autorisées de l'information conduit à la *confidentialité* ;
- l'absence d'altérations inappropriées de l'information conduit à l'*intégrité* ;
- l'aptitude aux réparations et aux évolutions conduit à *la maintenabilité*.

L'association, à la confidentialité, de la disponibilité vis-à-vis des actions autorisées, conduit à la *sécurité-immunité*[1].

Les entraves de la sûreté de fonctionnement : fautes, erreurs et défaillances sont les circonstances indésirables, causes ou résultats de la non sûreté de fonctionnement.

Un *service correct* est délivré par un système lorsqu'il accomplit sa fonction. *Une défaillance de service* souvent simplement dénommée défaillance, est la cessation de l'aptitude d'une entité à accomplir une fonction requise. Le service défaille soit parce qu'il ne respecte plus la spécification fonctionnelle, soit parce que la spécification fonctionnelle ne décrivait pas de manière adéquate la fonction du système. L'effet par lequel une défaillance est observée peut prendre plusieurs formes qui sont les *modes de défaillances*, et qui sont classés selon leur *gravité*. Lorsque la fonction du système comporte un ensemble de fonctions élémentaires, la défaillance d'un ou plusieurs services remplissant ces fonctions peut laisser le système dans un *mode dégradé*, qui offre encore un sous-ensemble de services à l'utilisateur. Dans ce cas, le système est dit avoir souffert de *défaillances partielles*.

Le service délivré étant une séquence d'états externes, une défaillance du service signifie qu'au moins un état externe dévie du service correct. La déviation du service correct est une *erreur*. La cause adjugée ou supposée d'une erreur est une *faute*. Les fautes peuvent être internes ou externes au système. La présence antérieure d'une *vulnérabilité*, c'est-à-dire d'une faute interne qui permet à une faute externe de causer des dommages au système, est nécessaire pour qu'une faute externe entraîne une erreur, et, potentiellement, une défaillance.

Une *reconfiguration* est l'action de modifier la structure d'un système qui a défailli, de telle sorte que les composants non défaillants permettent de délivrer un service acceptable bien que dégradé.

---

[1] L'association des qualificatifs *innocuité* et *immunité* permet de lever l'ambiguïté associée à *sécurité*. Il est à noter que cette ambiguïté n'existe pas en anglais, qui dispose de *safety* pour la sécurité-innocuité et de *security* pour la sécurité-immunité, sûreté de fonctionnement étant *dependability*.

Le développement d'un système sûr de fonctionnement passe par l'utilisation combinée d'un ensemble de méthodes, appelées *moyens* de la sûreté de fonctionnement, qui peuvent être classées en :

- *prévention des fautes* : comment empêcher l'occurrence ou l'introduction de fautes ?
- *tolérance aux fautes* : comment fournir un service à même de remplir la fonction du système en dépit des fautes ?
- *élimination des fautes* : comment réduire la présence (nombre, sévérité) des fautes ?
- *prévision des fautes* : comment estimer la présence, le taux futur, et les possibles conséquences des fautes ?

Nos travaux concerneront particulièrement l'évaluation de la sécurité. Dans le paragraphe suivant nous détaillerons les principales méthodes d'analyse de la sûreté de fonctionnement.

1.2.2. Méthodes d'analyse de la sûreté de fonctionnement

L'évaluation de la sûreté de fonctionnement d'un système consiste à analyser les défaillances des composants pour estimer leurs conséquences sur le service rendu par le système. Il existe des ateliers logiciels comme SOFIA [Sofia] destinés à simuler quantitativement l'effet d'une défaillance sur le comportement du système. Les principales méthodes utilisées lors d'une analyse de la sûreté de fonctionnement sont décrites ci-dessous.

1.2.2.1 L'analyse fonctionnelle

Une analyse fonctionnelle, en général, précède une étude de sûreté de fonctionnement. Une première analyse fonctionnelle dite externe permet de définir avec précision les limites matérielles du système étudié, les différentes fonctions et opérations réalisées par le système ainsi que les diverses configurations d'exploitation. L'analyse fonctionnelle interne permet de réaliser une décomposition arborescente et hiérarchique du système en éléments matériels et/ou fonctionnels. Elle décrit également des fonctions dans le système.

1.2.2.2 L'Analyse Préliminaire des Risques

L'Analyse Préliminaire des Risques (APR) est une extension de l'Analyse Préliminaire des Dangers (APD) qui a été utilisée pour la première fois aux Etats-Unis, au début des années soixante [Villemeur 88]. Depuis, cette utilisation s'est généralisée à de nombreux domaines tels que l'aéronautique, chimique, nucléaire et automobile.

Cette méthode a pour objectifs : 1) d'identifier les dangers d'un système et de définir ses causes. 2) d'évaluer la gravité des conséquences liées aux situations dangereuses et les accidents potentiels.

L'APR permet de déduire tous les moyens, toutes les actions correctrices permettant d'éliminer ou de maîtriser les situations dangereuses et les accidents potentiels. Il est recommandé de commencer l'APR dès les premières phases de la conception. Cette analyse sera vérifiée, complétée au fur et à mesure de l'avancement dans la réalisation de système. L'APR permet de mettre en évidence les événements redoutés critiques qui devront être analysés en détail dans la

suite de l'étude de sûreté de fonctionnement, en particulier par la méthode des arbres de défaillances qui sera décrite par la suite.

1.2.2.3 L'Analyse des Modes de Défaillance, de leurs Effets et de leurs Criticités

L'Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité (AMDEC) est une extension naturelle de l'Analyse des Modes de Défaillance et de leurs Effets (AMDE) utilisée pour la première fois à partir des années soixante pour l'analyse de la sécurité des avions [Villemeur 88]. L'AMDEC considère la probabilité d'occurrence de chaque mode de défaillance et la classe de gravité de ces défaillances, mais aussi les classes correspondantes de probabilités d'occurrence plus que les probabilités elles-mêmes. On peut ainsi s'assurer que les modes de défaillance ayant d'importants effets ont des probabilités d'occurrence suffisamment faibles, grâce aux méthodes de conception, aux diverses vérifications et aux procédures de test. Ce type d'analyse a été largement utilisé par la NASA et a été repris dans de nombreux domaines comme l'automobile.

1.2.2.4 Diagramme de Fiabilité

Un Diagramme de Fiabilité (DdF) permet le calcul de disponibilité ou la fiabilité du système modélisé, mais avec les mêmes restrictions qu'un Arbre de Défaillance (AdD) qui sera présenté dans le § 1.2.2.5), voir pires (pas d'événements répétés). Tous les chemins entre l'entrée et la sortie décrivent les conditions pour que la fonction soit accomplie. On suppose que les composants n'ont que deux états de fonctionnement (fonctionnement correct ou panne).

1.2.2.5 Les Arbres de Défaillances

L'analyse par Arbre de Défaillance (AdD) est une analyse déductive qui permet de représenter graphiquement les combinaisons d'événements qui conduisent à la réalisation de l'événement redouté. L'arbre de défaillance, dont la racine correspond à l'événement redouté, est formé de niveaux successifs tels que chaque événement soit généré à partir des événements du niveau inférieur par l'intermédiaire d'opérateurs logiques. Le critère d'arrêt de la décomposition arborescente d'un AdD est la connaissance que l'on a et l'appréciation de l'intérêt de la poursuite du processus de décomposition.

L'analyse qualitative par arbre de défaillance consiste à déterminer l'ensemble des coupes minimales. Une coupe est un ensemble d'événements entraînant l'événement redouté. Une coupe est minimale lorsque le retrait d'un événement de la coupe n'entraîne plus l'événement redouté (un arbre de défaillance a un nombre fini de coupes minimales). La recherche des coupes minimales se fait à partir d'une transformation de l'arbre en une expression booléenne et l'utilisation des lois de l'algèbre de Boole pour obtenir une expression booléenne réduite de l'événement redouté. Une autre méthode basée sur les diagrammes de décision binaire (BDD) est présentée dans [Sinnanmon et Andrew 97].

L'analyse quantitative consiste à assigner à chaque événement de base une probabilité d'occurrence. Une méthode d'évaluation de la probabilité d'occurrence de l'événement sommet, basée sur les diagrammes de décision binaire est présentée dans [Rauzy et Dutuit 97].

L'analyse par arbre de défaillance est largement utilisée dans les études de sûreté de fonctionnement car elle caractérise de façon claire les liens de dépendance, du point de vue dysfonctionnement, entre les composants d'un système. Bien que cette méthode soit efficace, elle présente des limites. L'une de ces limites est que l'ordre d'occurrence des événements menant vers l'état redouté n'est pas pris en compte. Or comme nous allons le voir par la suite, cette notion d'ordre dans les événements menant vers une défaillance est primordiale dans les systèmes pilotés par calculateurs qui sont hybrides. Nous présentons dans la section suivante ces systèmes.

1.3. Les systèmes pilotés par calculateurs

Un système piloté par calculateur est un système dont la dynamique d'évolution est contrôlée par un calculateur qui utilise des informations fournies par des capteurs pour élaborer sa commande qui sera appliquée au système piloté au moyen d'actionneurs. Aujourd'hui la majeure partie des procédés industriels et des systèmes embarqués est pilotée par calculateurs. On peut citer comme exemple les systèmes mécatroniques, les calculateurs de vols et les systèmes de contrôle des centrales nucléaires, etc.

1.3.1. Cas des systèmes mécatroniques

Un *système mécatronique* est un système combinant des technologies qui relèvent des domaines de la mécanique, de l'hydraulique, de la thermique, de l'électronique et des technologies de l'information. Il peut être décomposé en quatre entités en interaction (voir figure 1.1) : les capteurs, la partie opérative, le système de commande et de reconfiguration et les actionneurs.

Les *capteurs* mesurent des grandeurs physiques continues caractéristiques de la *partie opérative*. Le *système de commande et de reconfiguration* établit en fonction de ces mesures les actions à réaliser. Les *actionneurs* agissent sur la partie opérative.

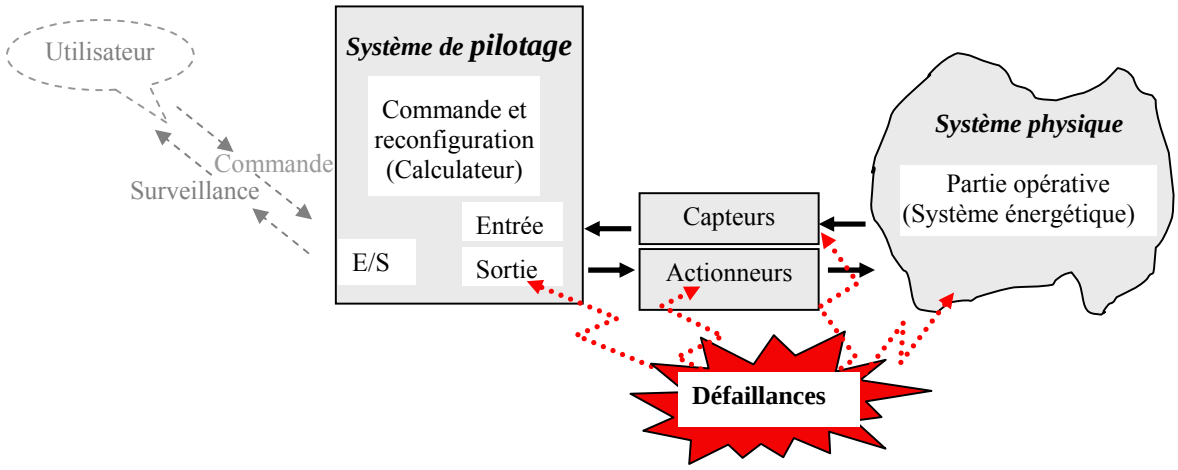


Figure 1.1. Eléments constitutifs d'un système mécatronique

Le système de commande a également pour objectif d'assurer que certaines grandeurs de la partie opérative soient maintenues dans un intervalle de sécurité. Lorsque certains événements relatifs à la sécurité du système se produisent, comme le franchissement d'un seuil de sécurité par une

variable caractéristique de la partie opérative, des actions sont mises en œuvre de façon à *reconfigurer* la partie opérative pour ramener les grandeurs caractéristiques de celle-ci dans les limites permises. Ainsi, pour ce type de système, la sécurité est assurée par des reconfigurations sans interruption de la mission. Ces systèmes sont donc des systèmes à reconfigurations dynamiques.

La reconfiguration peut être de type matérielle ou logicielle. La reconfiguration matérielle consiste à réparer un équipement ou à choisir une nouvelle architecture physique pour réaliser la fonction considérée. La reconfiguration logicielle se base sur la redondance, comme par exemple la redondance analytique des capteurs [Moncelet 98]. La différence entre ces deux types de reconfiguration est dans la vitesse d'exécution et le coût : un changement de configuration matérielle est toujours plus long et plus risqué. Une telle reconfiguration a une probabilité d'échec généralement non négligeable (défaillance à la sollicitation d'un composant).

Les systèmes mécatroniques sont des systèmes *hybrides*[2] dont la dynamique continue est associée à la partie énergétique (mécanique et hydraulique) et la dynamique discrète est liée à la commande numérique et à l'existence d'événements discrets (défaillances, dépassements de seuils).

Les spécificités des systèmes mécatroniques sont liées au fait que :

- ils sont reconfigurables et la décision de reconfiguration est prise par le système lui même.
- la réussite des reconfigurations dépend, dans le cas général, de la dynamique continue de la partie opérative, mais aussi du temps de réaction du système de pilotage.

Dans la partie de notre travail concernant la recherche de scénarios redoutés dans les systèmes mécatroniques nous supposerons que le système de commande et de reconfiguration (calculateur) est exempt de fautes. En effet ces fautes ne remettent pas en cause les bases de la démarche, mais ajouterait à la complexité du système étudié. Nous étudierons uniquement l'effet des défaillances des capteurs, des actionneurs qui causent la défaillance de la partie opérative. Dans le cadre de l'application à la vérification de propriétés il sera intéressant de considérer les fautes des calculateurs.

1.3.2. Exemple de systèmes mécatroniques

On peut énumérer plusieurs exemples de systèmes mécatroniques tels que : l'ABS (Anti Blocking System), l'EPS (Electronic Stability Program), l'ASR (Anti Slip Regulation) plus connu sous le nom Anti-patinage. Le TCS (Traction Control System), le MSR (Motor Schelepp Regulung), l'EBD (Electronics Brakeforce Distribution), le BAS (Braking Assistance System), l' AFU (Assistance au Freinage d'Urgence), la suspension hydraulique, la suspension pneumatique, etc. Ces systèmes sont utilisés dans le cadre de la sécurité active[3] [Kassaagi 01] des voitures

[2] Il s'agit de systèmes dont le comportement ne peut pas être décrit de façon purement continue ou purement discrète.

[3] La sécurité active s'intéresse à l'ensemble des fonctions permettant d'éviter un accident. Elle est complémentaire à la sécurité passive qui intervient lors d'accident (airbags, ceinture de sécurité, etc.).

d'aujourd'hui. Nous allons détailler ci-dessous quelques-uns de ces systèmes (les plus connus par les utilisateurs).

Système d'anti-blocage des roues (ABS) : comme son nom l'indique, ce système empêche le blocage des roues en cas de freinage brutal. Des capteurs situés au niveau des quatre roues, détectent le blocage (ou le risque de blocage) de chacune d'elles et informent toutes les 3 ou 7 millisecondes un calculateur. Ce dernier compare alors la pression exercée sur la pédale de frein, la vitesse du véhicule (détectée par un capteur) et la vitesse des roues. Si une roue est bloquée, le calculateur commande la relâche de la puissance de freinage appliquée sur la roue concernée. Cette action est réalisée par l'utilisation d'électrovalves (ou électrovannes du bloc hydraulique). Une pompe électrique remet de la pression dès que la roue a repris sa vitesse (cette opération peut s'effectuer jusqu'à 12 fois par seconde). Grâce à un temps de réaction très court, le système peut maintenir chaque roue à la limite du blocage du véhicule permettant alors une décélération maximale et une réduction des distances de freinage. Un deuxième avantage est que le véhicule conserve son adhérence et répond aux sollicitations de la direction. Le conducteur garde ainsi le contrôle de son véhicule. L'ABS a fait son apparition en 1952 sur les avions. Depuis son utilisation s'est élargie au domaine automobile dès 1978.

Afin d'assurer une grande fiabilité pour ce système vu sa criticité, sans compromettre ses performances et ses temps de réponse, les constructeurs l'ont doté de plusieurs stratégies de reconfiguration en cas de défaillance. Comme nous venons de le voir, le système ABS est composé d'un calculateur ABS, de cinq capteurs de vitesse, du système de freinage en parallèle avec quatre électrovannes. Les stratégies de reconfiguration sont nombreuses et complexes. Parmi les plus simples, celle qui consiste à estimer la vitesse de la roue lorsque le capteur associé est diagnostiqué défaillant (détection par le calculateur). La vitesse de la roue est donc estimée au travers des valeurs fournies par les autres capteurs. Selon le mode de défaillance du capteur (dérive, absence de signal, etc.), d'autres modes de reconfiguration sont prévus. Par exemple, en cas de dérive, le capteur est remis à zéro puis réactivé. La concordance de ses valeurs avec celles des trois autres permet de confirmer ou infirmer sa défaillance définitive et donc de poursuivre ou non l'estimation de la vitesse de la roue concernée. Le conducteur est averti à ce moment de la perte du capteur. Si deux capteurs sont défaillants, le voyant STOP est allumé.

Système de contrôle de trajectoire (EPS) : Ce système dit aussi de stabilité est une application récente de l'ABS. Il permet le contrôle de la stabilité du véhicule. L'EPS agit pour remettre la voiture dans une bonne trajectoire suite aux erreurs de conduite. Ce système détecte une tendance au dérapage et corrige en agissant indépendamment sur une ou plusieurs roues par l'intermédiaire des freins et /ou du moteur afin de rétablir la voiture dans son axe. Le système se déclenche suite à une perte d'adhérence du véhicule le plus souvent lors d'un sous-virage (l'avant chasse), d'un sur-virage (l'arrière chasse) ou en cas de manœuvre brutale. Un témoin lumineux signale la perte d'adhérence. De nombreux systèmes tels que l'ABS, EBD et AFU sont aujourd'hui intégrés à l'EPS avec éventuellement des capteurs supplémentaires pour corriger le dérapage du véhicule. L'EPS, comme de nombreux systèmes embarqués, illustre une forte interaction entre les sous-systèmes de l'automobile afin de réaliser une ou plusieurs fonctions. Par exemple le calculateur EPS contrôle le système ABS, le système de freinage et le couple moteur pour maintenir l'adhérence du véhicule.

En effet, l'interaction croissante entre les composants et les interconnections des fonctions pourraient être à l'origine de l'apparition de situations critiques (redoutées) non prévues du fait de la complexité croissante des systèmes automobiles. Ceci rend inefficaces les méthodes de travail des concepteurs basées sur des études et des validations séparées des différents sous-systèmes, auparavant indépendants mais fortement interconnectés aujourd'hui.

L'augmentation du nombre de composants et de fonctions, assurées par l'EPS, rend plus complexe l'élaboration des stratégies de reconfiguration. Une des plus simples, et qui est partagée avec le système ABS, est celle qui consiste à compenser une défaillance des électrovannes alimentant en pression hydraulique le circuit de freinage. Si une des électrovannes est bloquée en ouverture ou en fermeture, le système envoie des trains d'impulsions de forte intensité pendant un certain temps afin de secouer la bobine de commande de l'électrovanne et de la débloquer. Pendant ce temps, l'EPS continue de remplir sa mission dans un mode relativement dégradé en s'appuyant sur les autres électrovannes pour compenser l'excès ou le manque de pression dans le circuit hydraulique.

La réussite de certaines reconfigurations du système EPS est liée aux temps de réponse liés aux grandeurs énergétiques telle que la pression. Cette dépendance entre les aspects continus et discrets place les systèmes mécatroniques dans le cadre des systèmes dynamiques hybrides qui seront présentés dans la section qui suit.

1.3.3. L'aspect dynamique et hybride des systèmes pilotés par calculateurs

Un *système hybride* est un système qui nécessite dans sa description la prise en compte de sa dynamique continue et de sa dynamique discrète. La dynamique continue est représentée par des variables continues, la dynamique discrète représente les changements d'états dus à l'occurrence d'événements. Ces deux aspects rendent la modélisation hybride indispensable.

Dans son ouvrage [Zaytoon 01] a regroupé un grand nombre de travaux sur les systèmes dynamiques hybrides. Il donne quelques applications sur des systèmes industriels et présente plusieurs modèles dits « hybrides ». Ces modèles peuvent êtres classés selon deux approches : une *approche intégrée* qui intègre, au sein d'un même formalisme, les aspects continus et discrets ; une *approche séparée* qui sépare ces deux aspects en faisant coopérer deux modèles différents. L'approche intégrée englobe des modèles issus de l'extension de modèles continus comme les Bond Graphes à commutation [Buisson 93] et ceux issus de l'extension de modèles à événements discrets comme les réseaux de Petri hybrides [David et Alla 89]. L'approche séparée regroupe les modèles à base d'Automates hybrides, de Statecharts hybrides, de réseaux de Petri mixtes ou de réseaux de Petri Prédicats-Transitions-Différentiels (RdP PTD). Dans le chapitre suivant nous détaillerons les principales approches de modélisation de l'aspect hybride basées sur les réseaux de Petri, à savoir les RdP hybrides et les RdP PTD (les RdP Mixtes, peuvent être considérés comme une variante des RdP PTD)

Nous intégrons ainsi la connaissance des conditions d'apparition des défaillances et la réponse du système à ces défaillances dans notre modèle des systèmes pilotés par calculateurs. Ces défaillances agissent sur le processus continu et ce dernier joue un rôle important dans la réussite des reconfigurations et dans l'évaluation de la sûreté de fonctionnement de ces systèmes. Dans la section suivante, nous présentons la sûreté de fonctionnement des systèmes pilotés par

calculateurs en évoquant les limites des méthodes classiques ainsi que les différentes méthodes d'analyse et de modélisation.

1.4. Sûreté de fonctionnement des systèmes pilotés par calculateurs

1.4.1. Limites des méthodes classiques

Les méthodes classiques de la sûreté de fonctionnement, comme celles présentées dans le §1.2.2 sont statiques. Ces méthodes basées sur la logique booléenne pour représenter le système étudié sont adaptées à des systèmes à configuration statique, c'est-à-dire des systèmes dont les relations fonctionnelles entre leurs composants restent figées.

Dans le cadre de nos travaux, la prise en compte des mécanismes de reconfiguration dans les systèmes pilotés par calculateurs est essentielle. Cet aspect n'est pas pris en compte par les méthodes classiques de sûreté de fonctionnement ce qui les rend inefficaces. Par exemple la méthode des Arbres de Défaillance ne tient pas compte de l'ordre d'apparition des événements dans un scénario. En effet, une séquence d'événements peut conduire à un événement redouté alors que les mêmes événements se produisant dans un ordre différent ou à des dates différentes peuvent l'éviter. Le temps séparant deux événements n'est pas pris en compte dans la méthode des Arbres de Défaillance, les reconfigurations ne peuvent donc pas être représentées. Les défaillances temporaires ne sont pas non plus prises en compte. Plusieurs extensions des méthodes classiques ont été proposées afin d'élargir leurs champ d'application. Citons par exemple les arbres de défaillance avec les portes « A avant B ». Ces méthodes restent combinatoires et incapables de prendre en compte les changements d'états et les reconfigurations dans les scénarios redoutés. D'autres méthodes ont été introduites, comme les Diagrammes de Séquences d'Evénements (Event Sequence Diagrams notés ESD) [Kermisch et Labeau 02] afin de permettre une meilleure représentation visuelle d'événements ordonnés dans le temps. Bien que les ESD représentent de manière claire les scénarios en compétition, ils ne peuvent pas être générés automatiquement et nécessitent une définition des états et des transitions. Tous les états de commande et de reconfiguration doivent donc être énumérés par le concepteur, or dans le cas des systèmes dynamiques hybrides, le nombre d'états est infini si on prend en compte la partie énergétique. Il en est de même pour les méthodes fondées sur les graphes de Markov. Les limites des méthodes basées sur la simulation [Moncelet 98] sont dues à l'explosion combinatoire du nombre d'états, elles simulent alors la plu part du temps le fonctionnement normal du système puisque l'événement redouté est rare en général. Il faut cependant mentionner l'existence de développements théoriques et de méthodes destinés à apporter une solution au problème posé par la simulation de systèmes en présence d'événements rares [Villen-Altamirano]. Il existe en effet des techniques d'accélération de la simulation, largement utilisées avec succès, dans l'ingénierie nucléaire notamment. Dans le cas des réseaux de Petri, cette explosion combinatoire affecte principalement le graphe d'accessibilité, mais épargne, les réseaux de Petri originaux.

1.4.2. Fiabilité des systèmes dynamiques hybrides

La fiabilité des systèmes dynamiques hybrides, dite aussi *fiabilité dynamique* (*dynamic reliability)* ou encore *Probabilistic Dynamics* [Devooght et Smidts 92a], [Smidts et Devooght 92b], [Kermisch et Labeau 02], est une discipline récente dans la sûreté de fonctionnement.

[Kermisch et Labeau 02] définit cette discipline comme « la partie de sûreté de fonctionnement qui étudie de manière intégrée le comportement des systèmes industriels complexes affectés par une évolution dynamique continue sous-jacente ». Le fonctionnement des systèmes pilotés par calculateurs est régi par deux phénomènes : la variation continue et déterministe des paramètres énergétiques et aussi par les sollicitations et défaillances des composants du système, de nature discrète ou stochastique. Ces deux phénomènes sont en interaction, ce qui influence les paramètres de la sûreté de fonctionnement tels que la fiabilité et la sécurité. Les méthodes classiques de sûreté de fonctionnement sont incapables de prendre en compte de manière satisfaisante la dynamique des variables continues correspondant aux paramètres énergétiques [Dufour et Dutuit 02].

Les méthodes de fiabilité des systèmes dynamiques hybrides doivent prendre en compte les interactions du système. Dans la section suivante nous présentons les méthodes de modélisation des systèmes hybrides.

1.4.3. Méthodes de modélisation

Les méthodes les plus adaptées à la modélisation et à l'analyse des systèmes dynamiques hybrides sont les modèles états transitions tels que les graphes d'états (les graphes de Markov et les automates) et les approches basées sur les réseaux de Petri.

Les méthodes de modélisation peuvent être classées selon deux aspects : l'aspect dysfonctionnel qui permet de décrire les défaillances et les réparations ainsi que le comportement du système en présence de dysfonctionnements, l'aspect comportemental qui s'intéresse au comportement des systèmes dynamiques hybrides. La séparation entre ces deux aspects est la plus grande cause de l'inefficacité des méthodes classiques de sûreté de fonctionnement concernant la fiabilité dynamique. Ces deux aspects doivent être intégrés dans un même modèle de fiabilité en respectant leur interaction.

1.4.3.1 Modélisation de l'aspect dysfonctionnel

Graphes de Markov : cette méthode est utilisée pour analyser et évaluer la sûreté de fonctionnement des systèmes réparables. La construction d'un graphe de Markov consiste à identifier les différents états du système (défaillants ou non défaillants) et chercher comment passer d'un état à un autre lors d'un dysfonctionnement ou d'une réparation. A chaque transition, de l'état E_i vers l'état E_j, est associé un taux de transition L_{ij} défini de telle sorte que $L_{ij}.dt$ est égal à la probabilité de passer de E_i vers E_j entre deux instants très proches t et t+dt sachant que l'on est en E_i à l'instant de temps t.

La modélisation avec les graphes de Markov permet de prendre en compte les dépendances temporelles et stochastiques plus largement que les méthodes classiques. En dépit de leur simplicité conceptuelle et leur aptitude à pallier certains handicaps des méthodes classiques, les graphes de Markov souffrent de l'explosion du nombre des états, car le processus de modélisation implique l'énumération de tous les états possibles et de toutes les transitions entre ces états. Ce problème peut se poser même dans le cas de la modélisation du seul aspect dysfonctionnel, mais il devient un handicap énorme si on souhaite rajouter la description de l'aspect fonctionnel à celui de l'aspect dysfonctionnel. Pour surmonter ce problème, on peut alors s'orienter vers les réseaux de Petri stochastiques.

Réseaux de Petri stochastiques : les réseaux de Petri stochastiques [Molloy 82], [Natkin 80], [Chiol et al 93] sont obtenus à partir des réseaux de Petri classiques en associant des durées de franchissement aléatoires aux transitions. Ils permettent de prendre en compte, de manière plus structurée que les graphes de Markov, l'occurrence des défaillances et leur influence sur le comportement du système. En effet, le parallélisme étant pris en compte ils permettent d'expliciter l'architecture du système en décrivant indépendamment les états des divers objets composant le système et leurs interactions.

Une extension nommée "Réseaux de Petri Stochastiques Généralisés "(RDPSG) [Marsan et al 84] permet de prendre en compte, en plus de transitions avec des lois exponentielles, d'autres transitions dites « immédiates » tirées sans délai et qui sont prioritaires par rapport aux transitions à délai aléatoire. On peut citer d'autres extensions telles que les Réseaux de Petri Stochastiques Etendus (RdPSE) [Dugan et al 84] et les Réseaux de Petri Stochastiques et Déterministes (RdPSD) [Marsan et Chiola 86]. Les RdPSE permettent de prendre en compte des lois de distribution quelconques et les RdPSD combinent des délais exponentiellement distribués et des délais constants.

Dans le paragraphe ci-dessous, nous présentons les formalismes de modélisation de l'aspect fonctionnel : il s'agit donc de décrire le fonctionnement normal du système.

1.4.3.2 Modélisation de l'aspect fonctionnel

La prise en compte de l'aspect hybride des systèmes dynamiques hybrides est essentielle pour évaluer leur sûreté de fonctionnement en explicitant l'aspect fonctionnel.

Les automates : les *automates* sont l'un des formalismes états transitions les plus utilisés dans la description des systèmes à événements discrets. Ce formalisme a été étendu sous la forme des automates hybrides pour modéliser correctement les systèmes dynamiques hybrides. Les automates hybrides sont une extension des automates temporisés[4] [Alur et al 95]. Informellement, un *automate hybride* est l'association d'un automate à états finis et d'un ensemble d'équations dynamiques continues pilotées par ce dernier.

[4] Les automates temporisés sont des automates à états finis étendus par un ensemble d'horloges dont les valeurs croissent uniformément avec le passage du temps et qui peuvent être remises à zéro.

Le point faible de ce formalisme est l'explosion combinatoire du nombre d'états du graphe. Pour éviter ce problème, dans le cas de la modélisation des systèmes complexes pouvant être découpés en sous-systèmes, il est possible de construire un modèle d'automate pour chacun d'eux et de les composer ensuite pour élaborer l'automate correspondant au système global. La composition se fait par synchronisation entre les automates des différents sous-systèmes, soit par messages, soit par variable partagées. Toutefois cette composition entre automates rend difficile l'analyse de leurs propriétés. D'où le besoin de disposer de mécanismes de structurations plus puissants offerts par des modèles de plus haut niveau comme les Statecharts [Harel 87].

Les relations de cause à effet menant vers l'état redouté ne sont pas représentées d'une façon claire et homogène avec les automates. En effet au sein d'un automate représentant un objet séquentiel (élément d'un produit d'automates ou d'un ensemble d'automates communicants), les relations de cause à effet sont représentées par les événements qui relient, chacun, un état origine et un état destination. Chaque événement correspond à une causalité explicite entre deux états. Par contre entre deux automates, ces relations de cause à effet sont la conséquence directe ou indirecte de synchronisations par messages ou de communication par variables partagées. L'existence d'une relation de causalité ou non dépendra de la valeur du message ou de la façon suivant laquelle la variable partagée est modifiée et testée. Cela donne une représentation non unifiée des relations de cause à effet inter et intra automates. Dans le paragraphe suivant, nous verrons que ce n'est pas le cas dans le formalisme des réseaux de Petri.

Les réseaux de Petri et l'approche hybride : les *réseaux de Petri* sont très utilisés dans la modélisation des systèmes à événements discrets et dans les études de sûreté de fonctionnement des systèmes dynamiques. Ils sont caractérisés par une évolution asynchrone dans laquelle les transitions des composantes parallèles sont franchies les unes après les autres, et par une représentation explicite des synchronisations et des mécanismes d'allocation. Plusieurs extensions des réseaux de Petri ont été élaborées pour répondre à la modélisation des problèmes spécifiques et pour maîtriser la taille et la lisibilité des modèles. L'un des points forts des réseaux de Petri par rapport aux autres formalismes comme les Statecharts, repose sur ses fondements théoriques qui lui permettent de vérifier les propriétés générales d'un modèle (vivant, réinitialisable, sans blocage ou borné, etc.,) ainsi que l'accessibilité de certains marquages. Les méthodes de recherche de propriétés dans les réseaux de Petri sont basées sur l'élaboration du graphe des marquages accessibles, sur l'algèbre linéaire (calcul des invariants de places et des transitions), la réduction des réseaux ainsi que sur la logique linéaire qui permet de caractériser les relations d'ordre partiel.

Parmi les diverses extensions des réseaux de Petri pour prendre en compte l'aspect hybride, on peut citer les réseaux de Petri de haut niveau, les réseaux de Petri hybrides et les réseaux de Petri couplés avec les équations algébro-différentielles qui seront détaillés dans le chapitre suivant.

Nous abordons dans la section suivante les méthodes d'analyse qualitatives et quantitatives des systèmes dynamiques hybrides.

1.4.4. Méthodes d'analyse

Les études de sécurité des systèmes pilotés par calculateur sont souvent basées sur une analyse qualitative ayant pour objectif la détermination des scénarios aboutissant à l'occurrence de l'état redouté, suivie d'une analyse quantitative pour estimer la probabilité de leur apparition.

1.4.4.1 L'analyse qualitative

Cette analyse a pour but de caractériser les scénarios redoutés par des changements d'états et des enchaînements d'événements qui conduisent le système vers un état dit redouté. Quel que soit le formalisme choisi pour la modélisation du système, nous pouvons procéder par exploration des états. On génère les trajectoires possibles du système : les exécutions de l'automate ou les séquences de tirs du réseau de Petri. On détermine ensuite toutes les séquences permettant l'accessibilité d'un état donné par exploration du graphe des états.

Bien que cette approche ait l'avantage d'être exhaustive, elle souffre de deux inconvénients majeurs. Le premier, est qu'elle est limitée par le problème classique de l'explosion combinatoire du nombre d'états due d'une part à la taille des systèmes et d'autre part à l'augmentation du nombre d'états suite à la discrétisation des variables continues. Notons quand même que sans hypothèses ajoutées, le nombre de scénarios possibles est infini. Le second inconvénient est dû au traitement du parallélisme dans le graphe des marquages accessibles par entrelacement qui a pour conséquence l'obligation de ne décrire le comportement discret que par des séquences. Or dans une séquence, ce n'est pas parce qu'un franchissement de transition t_2 suit un franchissement de t_1 que t_1 est la cause nécessaire de t_2. Ce n'est le cas que si un jeton produit par *t1* est consommé par *t2*. On ne trouve pas cette information dans le graphe des marquages ce qui rend difficile l'automatisation de l'extraction des scénarios minimaux et des ordres partiels.

Il y a donc un besoin, celui de trouver un autre outil permettant de prendre en compte le parallélisme sans entrelacement et de décrire le comportement discret sans énumérer des séquences. Nous essaierons de répondre à ce besoin en introduisant, dans le chapitre suivant, la notion de scénario.

1.4.4.2 L'analyse quantitative

Méthodes analytiques : Ces méthodes consistent à résoudre les équations de Chapman-Kolmogorov associées au graphe de Markov. Celles-ci donnent l'évolution temporelle de la probabilité d'être dans un état, sous l'hypothèse que les processus de défaillance et de réparation des composants du système suivent une loi exponentielle à taux constant (hypothèse Markovienne).

Méthodes de discrétisation : Le problème majeur des systèmes pilotés par calculateurs provient de leur nature hybride. Une approche possible consiste à discrétiser les variables continues. Le temps peut également être discrétisé ou non. On peut citer la méthode la plus utilisée en fiabilité dynamique : les Arbres Dynamiques Discrets (Discrete Dynamic Event Trees ou DDET).

Cette méthode a pour but de générer les scénarios redoutés obtenus par propagation des défaillances des composants élémentaires du système. Elle est basée sur la partition de l'espace des variables continues en cellules disjointes. A partir d'un modèle qui prédit la réponse du système aux défaillances, l'évolution des variables physiques est calculée à chaque pas du temps. A cet instant on calcule de même la probabilité de toutes les combinaisons des états des composants et on génère toutes les séquences d'événements constituant les scénarios possibles, et ainsi de suite jusqu'à ce qu'un état absorbant soit atteint. Une présentation détaillée de cette technique est donnée dans [Moncelet 98] et [Kermisch et Labeau 02].

Les méthodes comme DYLAM (Dynamic Logical Analytical Methodolgy) et DETAM (Dynamic Event Tree Analysis Method) s'inscrivent dans le cadre des DDET et ont été appliquées à des systèmes de taille industrielle [Devooght et Smidts 94] [Acosta et Siu 93]. Ces méthodes ont été développées afin de mieux gérer les multiples scénarios générés.

Simulation de Monte Carlo : Quand l'hypothèse Markovienne n'est pas vérifiée, ce qui est le cas d'un grand nombre de systèmes industriels complexes, les évaluations se font par la simulation de Monte Carlo. La simulation de Monte Carlo est une méthode numérique basée sur le tirage de nombres aléatoire [Kermisch et Labeau 02]. Elle permet d'estimer l'espérance mathématique d'une variable aléatoire qui est une fonction de plusieurs paramètres. Elle permet également d'estimer toute quantité, déterministe ou stochastique, dont la valeur a pu être associée à l'espérance mathématique d'une variable aléatoire qui n'est pad directement liée à la physique du problème étudié. Citons par exemple l'exemple historique de l'estimation du nombre π. L'utilisation de la simulation Monte Carlo dans l'étude de sûreté de fonctionnement, permet de lever l'hypothèse markovienne et permet de traiter des systèmes industriels. Différentes méthodes ont été développées, comme les méthodes de transition forcées depuis le milieu des années 80 qui permettent de réduire le nombre d'histoires à simuler qui est un problème majeur dans le cadre des systèmes mécatroniques [Moncelet 98]. D'autres techniques se sont basées sur la réduction de temps d'une histoire. Des méthodes pour accélérer la simulation ont été développées dans [Champagnat 98].

D'autres approches, couplant la simulation de type DDET à du Monte Carlo, ont également été développées afin d'investiguer de manière plus efficace l'ensemble de l'arbre d'événements consécutif à un événement initiateur. Citons dans ce cadre l'approche Monte Carlo Dynamic Event Tree (MCDET) développée et testée par [Hofer et al 01] et les skeleton-based techniques. L'approche MCDET permet un traitement approximatif des transitions aléatoires continues et également des transitions aléatoires discrètes. Une évaluation de l'erreur d'approximation est fournie par cette méthode.

Discussion: Une idée récurrente est qu'il faut travailler avec des probabilités conditionnelles et donc déterminer des états, ou des classes d'états, de probabilité connue, et qui sont tels qu'à partir d'eux le système a une grande probabilité d'évoluer vers un état redouté. La détermination de tels états (ou de telles classes d'états) s'apparente fortement à un problème de vérification (pour la vérification, la grande probabilité est remplacée par la certitude). C'est pourquoi nous allons étudier les techniques de vérification et de preuve dans la section suivante.

1.5. La vérification de propriétés spécifiques

La vérification consiste à déterminer si le système satisfait certaines propriétés. La vérification a pour but de révéler les fautes de conception qui ont pu être introduites au cours de n'importe quelle phase du cycle de développement. Pour des raisons de coût et d'efficacité, il est important de les révéler au plus tôt, et les vérifications doivent être intégrées dès le début et tout au long du processus de développement.

On distingue quatre grandes classes de techniques de vérification : l'analyse statique, la vérification de modèle (ou *model-checking)*, la preuve et le test. Ces techniques sont en général utilisées d'une manière indépendante : une technique est choisie selon l'étape de développement pour un problème de vérification donné. On ne présente dans la suite que les techniques qui nous intéresse. Certains travaux présentés au paragraphe 1.5.4 ont choisi un couplage entre techniques pour exploiter leur complémentarité au sein d'un même problème de vérification.

1.5.1. La vérification de modèle

La vérification de modèle ou le *model checking* [Schnoebelen 99] est une technique de vérification formelle qui est développée afin de permettre une vérification automatique et exhaustive de systèmes. Elle s'applique à une large classe de systèmes : protocoles de communication, réseaux téléphoniques, industrie manufacturière, systèmes embarqués, etc.

1.5.1.1 Principe et Algorithme du model-checking

Le principe du *model-checking*, ou vérification de modèle, est de vérifier de façon exhaustive des propriétés portant sur un modèle du système étudié. La vérification d'un système par la méthode du *model-checking* nécessite trois étapes : la modélisation formelle du système sous forme d'un automate fini (ou une variante de cette représentation), l'expression formelle de la propriété souhaitée par une formule dans une logique temporelle et l'algorithme de *model-checking* qui parcourt de façon exhaustive l'automate en vérifiant la formule pour chacun de ses états. On montre en fait que l'automate est un modèle (au sens logique du terme) de la formule. Cet algorithme est implémenté dans le *model-checker* (outil). La plupart des *model-checkers* sont capables de générer une réponse positive si la propriété est garantie pour tous les comportements du modèle et une réponse négative complétée par un contre-exemple illustratif en cas de violation de la propriété. Une fois la vérification faite, elle n'est valable que pour un modèle donné. Si une modification est apportée au modèle, il faut reconstruire tous les états pour pouvoir de nouveau affirmer que la propriété vérifiée est vraie.

1.5.1.2 Les propriétés pouvant être vérifiées

Nous distinguons dans les travaux sur le *model-checking,* plusieurs types de propriétés qui peuvent être classées différemment. Nous présentons dans ce paragraphe une classification des propriétés temporelles souvent rencontrées dans les travaux de spécification des systèmes temps réel.

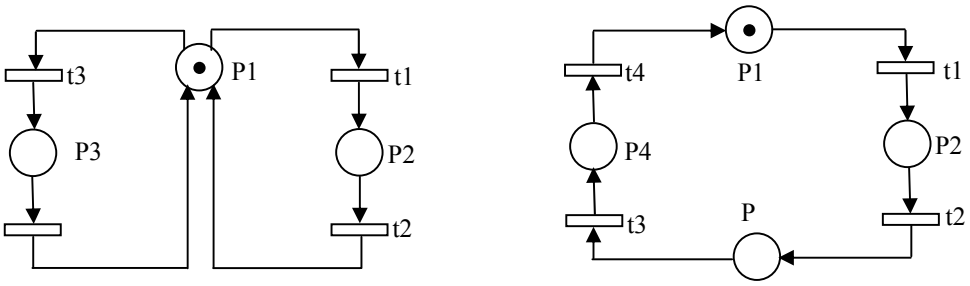
Propriétés d'atteignabilité *(reachability)* : certains auteurs parlent d'accessibilité, elles énoncent que, sous certaines conditions, un état du système peut être atteint ou non, par exemple,

une défaillance peut survenir, possibilité de revenir à l'état initial du système, possibilité d'effectuer certaines commandes, etc.

Propriétés de sûreté *(safety)* : qui énoncent que, sous certaines conditions, quelque chose n'arrive jamais. Par exemple, un état non souhaité ne se produit jamais, deux systèmes ne seront jamais simultanément en état critique. Les propriétés de sûreté peuvent être exprimées sous d'autres formes, par exemple la non atteignabilité.

Propriétés de vivacité *(liveness)* : qui énoncent que, sous certaines conditions, quelque chose finit par avoir lieu. Plusieurs formes de vivacité sont distinguées entre autre, la vivacité de progrès ou simple et la vivacité bornée : 1) *Propriétés de vivacité simple* (ou de progrès) qui énoncent qu'un état est toujours atteignable. Par exemple, une commande sera nécessairement émise, le système peut toujours retourner à l'état initial. 2) *Propriétés de vivacité bornée (bounded liveness)* qui énoncent que, sous certaines conditions, quelque chose finit par avoir lieu avant un certain temps. Par exemple : une panne arrivera dans la journée, toute requête finira par être satisfaite en moins de 5min.

Propriétés d'équité *(fairness)* : qui énoncent que, sous certaines conditions, quelque chose aura lieu (ou n'aura pas lieu) un nombre infini de fois. Par exemple : une ressource finira par être attribuée à un processus demandeur. La figure 1.2 représente la vivacité et l'équité dans le cadre des réseaux de Petri.



a) vivacité et non équité b) vivacité et équité

Figure 1.2. Vivacité et équité dans les réseaux de Petri

Propriétés d'absence de blocage *(no deadlock)* : qui énoncent que le système ne se trouve jamais dans un état qu'il ne peut plus quitter. Par exemple, une panne peut toujours être réparée. Des définitions formelles de ces propriétés sont données dans [Schnoebelen 99].

Le choix de la méthodologie de vérification et de spécification dépend du type de propriété à vérifier. Précisons que les propriétés d'atteignabilité sont souvent les plus intéressantes. Il est en effet possible de résoudre la vérification de certaines propriétés en les ramenant à un problème d'atteignabilité. Comme exemple, pour trouver des scénarios menant vers un état redouté [Kehren et Seguin 03], il suffit de considérer la propriété de sûreté comme la négation d'une propriété d'atteignabilité : « on ne peut pas atteindre un état tel que… ». Dans ce cas, le model-checker donne une réponse négative et un contre exemple qui est une exécution possible permettant d'atteindre l'état redouté. Toutefois, pour obtenir un autre scénario, il faut reformuler une

nouvelle propriété qui exclut le premier contre-exemple et recommencer, or cette formulation est généralement compliquée.

1.5.1.3 Logique temporelle

Parmi les principales logiques temporelles utilisées dans les outils de *model-checking* nous pouvons citer LTL (Linear Temporal Logic), PLTL (Propositional Linear Temporal Logic) [Pnueli 81], et CTL (Computation Tree Logic) [Clark et Emerson 81]-. Une présentation des algorithmes de *model-checking* dans chaque cas est donnée dans [Bérard et al 99].

LTL : la Logique Temporelle Linéaire (LTL) qui a été proposée pour la première fois pour la spécification temporelle des systèmes en 1977 par [Pnueli 77] est une extension de la logique classique avec des opérateurs temporels, tels que G et F (représentant respectivement "globalement ", "finalement "). Des formules de cette logique peuvent être interprétées comme des séquences infinies d'états, d'un système, produites par un automate. La LTL permet de s'intéresser à des propriétés portant sur l'exécution d'une succession d'états. Le temps est représenté par l'ordre des états dans la séquence et croît d'une manière linéaire à partir d'un état initial. Cette logique est dite linéaire car, chaque état ne possède qu'un seul successeur. PLTL est une extension de la logique LTL.

CTL : s'intéresse à des systèmes où, pour un état donné, nous avons potentiellement plusieurs futurs possibles, le temps est arborescent [Emerson et Clarke 80]. En CTL, il y a deux types de formules : les formules d'états qui sont évaluées pour un instant donné et les formules de chemin qui sont évaluées le long d'un chemin. Cette logique permet de vérifier pour chaque type si les formules sont vraies pour un chemin ou pour tous les chemins. La prise en compte d'un temps dense est possible grâce à une extension de CTL : TCTL « Timed Computation Tree Logic » [Alur et al 93] pour étendre la logique CTL.

1.5.1.4 Différents outils de model-checking

Les outils *Model-checkers* peuvent être répertoriés en deux classes : *model-checkers qualitatifs*, qui se basent sur une modélisation qualitative des systèmes (temps réel), et *model-checkers quantitatifs à temps continu*, qui se basent sur une modélisation quantitative des systèmes (temps réel).

Les *model-checkers qualitatifs* sont puissants et permettent de vérifier des propriétés telles que « ordre d'événement » et « fin de tâche ». Ils ne permettent pas de vérifier des propriétés faisant référence au temps quantitativement tel qu'une propriété de vivacité bornée « toute requête finira par être satisfaite en moins de 5 minutes ». Ceci est dû au fait que ces outils se basent sur des automates non temporisés. Pour traiter ce genre de propriétés il faut avoir recours à des *Model-checkers quantitatifs* se basant sur des automates temporisés ou hybrides rajoutant des structures permettant de calculer le temps.

1.5.1.4.1 Model-checker qualitatif

Les *model-checkers qualitatifs* se basent sur des logiques qualitatives pour la description du comportement et des propriétés à vérifier d'un système (temps réel). Nous citons par exemple :

SMV [McMillan 93] : est développé à l'université de Carnegie-Mellon (Pittsburgh, USA). Cet outil permet la description des systèmes temps réel et la vérification de ses propriétés décrites en logique CTL. Après vérification, SMV peut indiquer en sortie si la propriété considérée est vérifiée. Dans le cas d'échec, il indique un contre- exemple. Grâce à l'utilisation d'une approche *symbolique*, SMV a pu être utilisé sur des systèmes de taille très élevée comme l'ont montré les travaux de Clarke sur l'étude de cas d'un circuit logique [Clarke et al 94].

SPIN [Holzmann 97] est développé aux Bell Labs (Murray Hill, New Jersey, USA). Il se base sur un langage d'entrée spécifique, Promela [Loeffler et Serhouchni 97], et sur la logique temporelle PLTL. SPIN permet la simulation et la vérification d'algorithmes répartis. Ses points forts sont la mise en oeuvre de diverses méthodes de réduction de l'espace des états : compression des états, utilisation des ordres partiels. Parmi les applications industrielles de SPIN nous pouvons citer son utilisation sur un contrôleur de vaisseau spatial [Havelund et al 01].

Dans le cas de l'utilisation d'un réseau de Petri pour la représentation du système à vérifier, il est bien sûr toujours possible de générer le graphe des marquages accessibles (si du moins le réseau de Petri est borné) puis d'utiliser un outil fondé sur les automates finis. Toutefois un outil comme TINA [Berthomieu et Menasche 83], [Berthomieu et al 03] permet de lutter contre l'explosion combinatoire en ne générant qu'un sous-ensemble des marquages accessibles suffisant pour prouver (ou infirmer) la propriété considérée. En particulier une technique comme celle des pas couvrants permet de limiter l'explosion combinatoire due au parallélisme.

1.5.1.4.2 Model-checker quantitatif à temps continu

Les *model-checkers à temps continu* ont été introduits pour la vérification des systèmes temps réel temporisés ou hybrides. Ils sont considérés à temps continu, car ils utilisent des variables d'horloge valuées réellement. Nous citons les *model-checkers* les plus répandus :

UPPAAL [Larsen et al 97] : est développé par P.Pettersson à l'université d'UPPsala (UPP) en Suède et Kim Larsen à l'université d'AALborg au Danemark (AAL), d'où son nom. Cet outil présente l'avantage d'offrir une interface graphique de bonne qualité. D'autre part, il permet de simuler le fonctionnement des automates temporisés, ce qui peut être utile dans la phase de modélisation. Cependant, cet outil utilise uniquement un fragment de TCTL et celui-ci ne permet de vérifier que des propriétés d'accessibilité sur des automates temporisés et ne permet pas de vérifier l'ensemble des formules TCTL, les opérateurs de temps ne pouvant pas être imbriqués. Un exemple de succès de UPPAAL a été de permettre le diagnostic, et la correction, d'une faute de conception dans un protocole audio/vidéo [Havelund et al 97]. Ce protocole, utilisé depuis plusieurs années dans l'électronique grand public, manifestait parfois un comportement erroné mais il n'avait pas été possible jusque là d'en déterminer la cause.

KRONOS [Yovine 97], [Bogza et al 98] : a été développé au laboratoire VERIMAG (Grenoble, France). Il permet de déterminer si un système temporisé, décrit comme la composition parallèle de plusieurs automates temporisés, satisfait une propriété exprimée par une formule dans la logique TCTL. Le point fort de KRONOS est l'expressivité de sa logique : KRONOS implante un algorithme de *model-checking* pour la logique TCTL. Il peut ainsi vérifier des propriétés d'atteignabilité, de sûreté et il est l'un des rares outils permettant de spécifier et vérifier des propriétés temporelles qui ne se ramènent pas à une propriété d'atteignabilité, comme la vivacité.

KRONOS propose des techniques de réduction de l'espace d'états afin d'optimiser les vérifications. Il intègre des techniques d'ordre partiel [Pagani 96] et propose d'optimiser le nombre d'horloges dans un système [Daws et Yovine 96]. L'idée est d'éliminer les horloges inutiles en fonction des gardes et remises à jour des transitions. Seules certaines réductions sont implantées dans KRONOS puisque trouver le nombre minimal d'horloges nécessaires pour un automate temporisé est un problème indécidable et que l'heuristique proposée est NP-Complète. Un outil, TAXYS [Closse et al 01], a été développé intégrant KRONOS avec le langage ESTEREL [Berry et Gonthier 92].

HYTECH [Henzinger et al 97] : est développé à l'université de Berkeley (Californie, USA). Il est conçu pour la vérification des systèmes hybrides linéaires, (et donc utilisable pour les automates temporisés), pour lesquels la loi d'évolution de certaines variables réelles est régie par une équation différentielle linéaire. Dans le cas hybride il y a toujours des variables discrètes et des horloges. Ainsi HYTECH permet de vérifier des propriétés exprimées en TCTL à partir d'une spécification du comportement par des automates hybrides linéaires. Les analyses sont limitées à des propriétés d'atteignabilité. HYTECH a été utilisé dans plusieurs cas d'étude, principalement des systèmes de contrôle [Stauner et al 97].

CMC [Laroussinie et Larsen 98] : (*Compositional Model Checking*) est un outil dédié aux automates temporisés. Ce *model-checker* propose une approche de vérification de formules par composition. CMC a pour but d'utiliser la structuration des spécifications en un réseau d'automates temporisés communiquant pour éviter l'explosion combinatoire de l'espace de recherche.

Mec [Arnold 90], [Arnold et Brlek 95] **:** est développé au LaBRI (Bordeaux, France). Il est utilisé pour la recherche de points de blocage, ou d'autres propriétés d'un système de transitions donné. Le formalisme utilisé pour décrire les systèmes est le modèle de Arnold-Nivat [Arnold et Nivat 82], qui est suffisamment général pour s'adapter à de nombreux formalismes. Une extension MecV [Vincent 03] a été réalisée pour manipuler les modèles du langage Altarica qui sera présenté dans le paragraphe 1.6.6.

TINA [Berthomieu et Menasche 83], [Berthomieu et al 03] (*Time Petri net Analyzer*): si la représentation du système est fondée sur des réseaux de Petri, la prise en compte des contraintes temporelles dans un contexte de vérification se fait, à l'heure actuelle, le plus souvent sous la forme des réseaux de Petri t-temporels. Bien qu'il soit possible de développer des techniques permettant la traduction d'un réseau de Petri t-temporel, qui sera présenté dans le paragraphe 2.2.1.3, en un automate temporisé pour utiliser les outils décrits ci-dessus, il semble plus efficace de constituer directement une structure de Kripke (développée dans les années 60 par Saul Kripke), adaptée à la propriété à vérifier. Comme nous sommes en présence de système ayant un ensemble infini d'états (à cause de temps dense) ces structures sont formées d'un ensemble fini de classes d'états. L'outil TINA offre actuellement la possibilité de construire deux types de graphe de classes d'état : le type W est suffisant pour la vérification de propriétés exprimées en LTL alors que le type A, qui génère plus de classes, est nécessaire pour la vérification de propriétés exprimées en CTL. Dans les deux cas, les classes sont représentées par un marquage et un ensemble d'inéquations temporelles.

1.5.1.5 Limites du model-checking

Une des limites du *model-checking* est la représentativité du modèle formel du système dont le niveau d'abstraction ne permet pas de prendre en compte tous les aspects liés au système et à son environnement, en particulier ceux concernant les fautes physiques. Une autre limite du *model-checking* est l'explosion du nombre d'états du modèle formel utilisé. Certains auteurs ont apporté des solutions pour limiter l'explosion combinatoire. A titre d'exemple, une stratégie combinant deux approches d'exploration partielle est présentée dans [Ribert et al 02]. D'autres approches ont été proposées par [Merz 01]. Le *model-checking symbolique* [Burch et al 92] utilise des représentations compactes d'ensembles d'états, on évite ainsi d'avoir à les énumérer. En particulier, la représentation d'ensembles d'états par des BDD (*Binary Decision Diagrams*) [Bryant 86], [Bryant 92] est très utilisée dans ce cadre. Les techniques d'ordre partiel évitent, dans un système composé d'automates en parallèle, d'avoir à parcourir tous les chemins d'exécutions possibles (voir par exemple [Holzmann et Peled 94], [Ribet et al 02]). Le principe sous-jacent est que pour vérifier la propriété, certains entrelacements d'événements équivalents ne sont pas construits.

Comme nous l'avons déjà écrit, il y a une grande similitude entre la vérification d'une propriété par *model-checking* et la recherche des comportements redoutés. Dans le premier cas on cherche à montrer qu'un état est inaccessible et l'on obtient un comportement y menant s'il est en fait accessible. Dans le deuxième cas on désire caractériser tous les comportements menant à un état redouté avec une probabilité d'occurrence d'un tel comportement. S'il y a similitude, cela ne veut pas forcément dire que la recherche des comportements redoutés peut facilement se faire en utilisant le *model-checking*. En effet, si le comportement recherché est "événement *a*" en parallèle avec "événement *b*", l'outil de *model-checking* peut très bien produire des séquences comme "événement *a*" puis "événement *c*" puis "événement *b*", ou bien "événement *b*" puis "événement d" puis "événement *a*" où b et d correspondent à des événements qui se sont produits par hasard, sans lien de causalité avec le fait d'atteindre l'état redouté. Non seulement ces comportements risquent de ne pas être très clairs pour le concepteur, mais il sera de plus bien difficile de passer à une évaluation quantitative de la probabilité d'occurrence. En résumé il sera difficile de garantir la complétude des comportements et surtout d'obtenir une caractérisation simple et minimale de l'ensemble des comportements redoutés.

1.5.2. La preuve

La preuve consiste à établir par une suite finie d'étapes de raisonnement (par mécanisme déductif ou par un système de réécriture) appelées inférence, qu'une certaine propriété est la conséquence logique d'axiomes ou d'un ensemble d'hypothèses décrivant le système étudié.

La preuve peut être appliquée à toutes les phases de développement du système, on peut par exemple chercher à prouver :

- Qu'une spécification satisfait certaines bonnes propriétés
- Qu'une étape de conception est correcte : une spécification détaillée satisfait une spécification plus abstraite (preuve de raffinement)

- Qu'un programme est correct par rapport à une spécification détaillée (preuve de programme), ou qu'il termine toujours sous certaines conditions (preuve de terminaison).

Une preuve peut être formelle ou informelle.

1.5.2.1 Preuve informelle

Une preuve informelle est une démonstration donnée en langage naturel, en utilisant éventuellement quelques notions mathématiques pour faciliter le discours. Dans le domaine informatique, la plupart des algorithmes de tolérance aux fautes sont publiés avec une preuve informelle. Une preuve informelle est validée par consensus : elle est lue et acceptée par les pairs. Ceci la distingue de la preuve formelle, qui peut être vérifiée automatiquement par des outils car le problème de vérification a été entièrement exprimé à l'aide des mathématiques.

1.5.2.2 Preuve formelle

L'assistance d'outil de preuve dans la preuve formelle nécessite la formalisation complète du problème de vérification. La description de l'algorithme, les hypothèses sur son environnement, et les propriétés attendues doivent être exprimées dans un langage ayant une syntaxe et une sémantique formelles, s'appuyant sur un cadre logique bien défini. Dans ce cadre, une preuve se ramène alors à des manipulations syntaxiques de formules. Les règles d'inférence indiquent comment produire des formules à partir des formules déjà produites, et une preuve peut être vue comme la construction d'un arbre de formules, dont les feuilles sont des axiomes et la racine la formule à prouver. Un cadre classique de preuve est le calcul des prédicats du premier ordre : il permet de raisonner sur des formules qui comportent des variables. Deux exemples de systèmes de déduction pour ce calcul sont la déduction naturelle et le calcul des séquents [Monin 00].

Il n'existe pas de procédure de décision qui, pour toute formule, déterminerait s'il existe une preuve ou non. Les démonstrateurs de théorèmes doivent donc mettre en œuvre des heuristiques, sans garantie d'aboutissement. En pratique, l'intervention d'un opérateur humain est souvent nécessaire pour guider la construction de la preuve. Malheureusement, il y a toujours le risque de se retrouver dans un cas d'échec de preuve : on ne peut arriver à déterminer si la formule à prouver est vraie ou non. L'investissement qu'a représenté la preuve a alors été inutile, ce qui constitue un sérieux inconvénient à l'utilisation de cette technique de vérification.

Pour permettre à l'utilisateur de comprendre la structure de la preuve, et faciliter l'analyse des parties en échec, il est souhaitable que l'environnement offre des facilités pour visualiser l'arbre de preuve. L'outil doit offrir une certaine visibilité sur les heuristiques qu'il a essayé de mettre en œuvre. L'arbre peut être montré à différents niveaux d'abstraction, par exemple en détaillant ou non les tactiques utilisées et les preuves de certains lemmes intermédiaires. Une étude des caractéristiques souhaitables des environnements de preuve peut être trouvée dans [Rushby 93]

Nous présentons ci-dessous quelques outils et environnements de preuve existants.

1.5.2.3 Quelques outils et environnements de preuve

Il existe de nombreux systèmes de preuve. Selon le degré d'automatisation de la preuve, on parlera de vérificateur de preuve qui se contente de vérifier une preuve déjà construite, d'outil d'assistance à la preuve, ou de démonstrateur de théorèmes. Parmi ces outils nous citons :

Boyer-Moor : est un des ancêtres des démonstrateurs de théorèmes. Il est basé sur un système formel particulier, la logique de Boyer-Moor [Boyer et Moor 79] dans laquelle sont exprimées les propriétés à démontrer. Boyer-Moore comporte plusieurs heuristiques puissantes lui permettant de trouver des preuves non triviales, comme des preuves par introduction de lemmes. Il a été utilisé dans le projet SIFT [Wensly et al 78] dans les années 80.

LP (Larch Prover) [Garland et Guttag 91] : la spécification dans cet outil de vérification de preuve se fait en langage Larch [Guttag et al 85]. Le système est fondé sur un ensemble de fonctions déclarées, des propriétés et des axiomes sont exprimés sous forme d'équations, des règles de réécriture, des règles de déduction et des règles d'induction. La construction des preuves est assistée par le système TLP (*Temporal Logic Prover*) [Engberg 95] qui offre une extension du langage Larch pour la logique temporelle TLA (*Temporal Logic of Action*).

LCF *(Logic of Computable Functions*) [Gordon et al 79] : est un des premiers démonstrateurs de théorèmes. Il a été programmé en ML et son langage sert également à écrire des stratégies de preuve. LCF a aussi introduit le terme de *tactique* et l'idée de preuve à l'envers : un utilisateur commence avec le but désiré et divise ce but en sous-buts plus simples en appliquant des tactiques. C'est l'inverse de la façon dont les preuves mathématiques sont traditionnellement écrites.

Hol (Higher Order Logic) [Gordon et Melham 93] : il a été développé principalement à l'université de Cambridge. Ce démonstrateur de théorèmes a été utilisé essentiellement pour des preuves de conception de matériel notamment celle du microprocesseur tolérant aux fautes Viper pour le RSRE (*Royal Signals and Radars Establishment* du ministère de la défense Britannique). Un des aspects les plus attrayants de ce système est la possibilité de définir des tactiques de démonstration adaptées à certains problèmes et à certains langages de spécification. Le système HOL a aussi été utilisé dans les domaines de la vérification matérielle [Kropf 99] et de la vérification de systèmes distribués [Prasetya 95].

PVS (Prototype Vérification Système) [Owre et al 95] : est développé au SRI (International Computer Science Laboratory) à Palo Alto (USA). Le démonstrateur de théorème PVS a été appliqué à de nombreux problèmes réels tel que la spécification et la modélisation des systèmes de contrôle de vol tolérants aux fautes de la navette spatiale [Crow et Vito 96]. La majorité des travaux de preuve se font au niveau de la spécification. Ces travaux peuvent concerner les propriétés temporelles comme les travaux de Dutertre qui portent sur la preuve en PVS du PCP (*Priority Ceiling Protocol*) [Dutertre 00]. Il s'est intéressé aux propriétés de synchronisation et de respect d'échéance de tâche.

B [Abrial 96] : est une méthode formelle utilisée avec succès en milieu industriel notamment dans le domaine ferroviaire comme le projet Météor (ligne de métro automatisé) [Behnia et Waeselynck 99]. Elle couvre toutes les étapes de développement du logiciel, de la spécification

jusqu'à la génération de code exécutable. L'intérêt principal de la méthode B est dans la possibilité de raffiner une spécification de haut niveau en une spécification suffisamment détaillée pour pouvoir être automatiquement traduite dans un langage de programmation.

Les démonstrateurs de théorème AUTOMATH [De Bruijn 68], COQ [Barras et al 99] et le prover ISABELLE [Paulson 94] sont également utilisés avec succès en milieu industriel notamment au niveau de la vérification de programme.

1.5.2.4 Limites de la preuve

La vérification par preuve est difficile à utiliser et à automatiser. Particulièrement, en méthode B, la vérification des contraintes dynamiques d'un système est effectuée par une technique de preuve, ce qui requiert que l'utilisateur, en plus d'être un expert en spécification, soit également un expert dans l'utilisation de cette technique. En effet la vérification des contraintes dynamiques est confiée à un outil informatique de démonstration de théorèmes, mais dès que cet outil n'arrive plus à progresser seul vers la solution, il fait appel à l'utilisateur pour continuer à avancer. En pratique, la vérification par preuve de contraintes dynamiques est rarement obtenue de manière automatique. De plus, le démonstrateur de théorème est confronté d'une part à la vérification de la correction partielle d'une propriété, et d'autre part à une vérification de terminaison pour certains schémas de contrainte dynamique.

Une autre façon d'attaquer le problème de vérification par preuve (formelle) des propriétés de haut niveau est de décomposer ces dernières en des propriétés de bas niveau (locales) et de les vérifier séparément. Mis à part le problème de faisabilité d'une telle décomposition, des hypothèses implicites sur l'environnement peuvent être rajoutées sans pour autant être satisfaites en vie opérationnelle.

Pour les systèmes de contrôle/commande, une modélisation détaillée de ces derniers, reproduisant les interactions entre le programme de contrôle et l'environnement physique, en prenant en compte les lois physiques du procédé contrôlé, les dispositifs matériels pour assurer ce contrôle et les occurrences possibles de fautes pouvant affecter ces dispositifs, est généralement infaisable. L'utilisation des approches formelles pour la vérification des propriétés de haut niveau s'avère insuffisante. En effet, deux cas d'école, très utilisés par les approches formelles, ont montré leurs limites : la cellule de production [Lewerentz et Lindner 95] et la chaudière à vapeur [Abrial 96]. Ces deux études de cas ont servi d'illustration et de comparaison de plusieurs méthodes de spécification et de vérification formelles. Sur ces deux études de cas, certaines propriétés ont été prouvées par des approches formelles et ont pu être violées au cours de test sur simulateur de l'environnement physique en exécutant un code conforme à sa spécification [Waeselynck et Fosse 99], [AtelierB]. Ceci est dû au fait que la vérification formelle est effectuée sur un modèle du logiciel en formulant des hypothèses sur l'environnement qui ne sont pas nécessairement satisfaites en réalité.

Les outils de preuve actuels ne sont pas orientés vers une description simple et une preuve directe de la dynamique discrète de systèmes complexes. Ils cherchent à traiter simultanément les données et la structure de contrôle et ne sont donc pas adaptés à l'analyse des relations de cause à effet d'un système modélisé sous la forme d'un réseau de Petri ou d'un ensemble d'automates communicants. Les travaux de [Rivière 03], et de [Khalfaoui 03] que nous allons présenter

brièvement dans la suite de ce chapitre peuvent être vus comme une réponse à ce manque. En s'appuyant sur une logique non classique (la logique linéaire) ils ont construit une démarche de preuve [Rivière 03] qui à travers l'analyse des relations de cause à effet permet d'élaborer des scénarios menant à des situations critiques [Khalfaoui 03]. Avant d'aborder ce point nous allons présenter un tour d'horizon des techniques de preuve qui se posent en alternative pragmatique lorsque vérification et preuve sont en échec.

1.5.3. Le test

Le test se pose effectivement en alternative pragmatique de la vérification et de la panne, pourvu que l'environnement de test reproduise le plus fidèlement possible l'environnement opérationnel. En pratique, le programme de contrôle sera exécuté sur matériel cible en connexion avec un simulateur de l'environnement physique, offrant des possibilités d'injection de fautes, si la propriété ciblée doit être vérifiée en présence de telles fautes.

Le test est une technique de vérification dynamique qui consiste à exécuter un programme en lui fournissant des entrées valuées, appelées les entrées de test, et à vérifier la conformité des sorties par rapport au comportement attendu pour déterminer leur correction s'il y a lieu [Laprie et al 95]. Sa mise en œuvre nécessite de résoudre deux problèmes :

- Le problème de la *sélection* d'entrées de test.
- Le problème de l'*oracle* [Weyuker 82] ou comment décider de l'exactitude des résultats observés, fournis par le programme en réponse aux entrées de test.

Sauf cas exceptionnel, le test exhaustif d'un programme sur toutes les entrées possibles n'est pas praticable. Le testeur est donc amené à sélectionner de manière pertinente un petit sous-ensemble du domaine d'entrée. Cette sélection s'effectue à l'aide de critères de test, qui peuvent être liés à un modèle de la structure du programme : on parle alors de *test structurel*, ou à un modèle des fonctions que doit réaliser le programme : on parle alors de *test fonctionnel.*

Quel que soit le critère de sélection, la génération des entrées peut être déterministe ou probabiliste. Le premier cas définit le *test déterministe* où les entrées sont déterminées par un choix sélectif de manière à satisfaire le critère de test retenu. Le deuxième cas définit le *test statistique* où les entrées sont générées de manière aléatoire selon une distribution probabiliste sur le domaine d'entrées, la distribution et le nombre des données de test étant déterminés à partir du critère retenu [Waeselynk 9*3*].

Les solutions les plus satisfaisantes pour le problème d'oracle sont basées sur l'existence d'une spécification formelle du programme sous test. La spécification est alors utilisée pour déterminer les résultats attendus, soit lors de la sélection des entrées de test, soit à posteriori.

Le *test orienté Propriété* utilise la spécification de la propriété pour guider le processus de test [Fink et Bishop 97]. L'objectif est de valider le programme par rapport à une propriété cible en l'exécutant et en observant si cette propriété est violée ou pas. Etant donné que l'accent est mis sur la propriété et non pas sur la spécification entière, l'oracle se limite ainsi à la vérification du respect de la propriété, ce qui rapproche ce type de test des techniques de vérification.

1.5.4. Couplage de techniques de vérification

Le couplage entre les quatre grandes classes de techniques de vérification (analyse statique, *model-checking*, preuve de systèmes complexes et test) présentées précédemment est possible. Plusieurs travaux de recherche exploitent leur complémentarité pour résoudre le problème de la vérification. Les couplages les plus étudiés se situent autour du *model-checking.*

Couplage autour du *model-checking* : Un exemple réussi de couplage est l'intégration de techniques de *model-checking* dans l'environnement de preuve PVS [Rushby 99]. Le principe de l'approche est d'utiliser une interaction étroite entre la preuve de théorèmes, et la génération d'abstractions pour le *model-checking*. Chaque étape consiste à appliquer alternativement l'une des deux techniques (preuve ou *model-checking*), en exploitant les informations obtenues lors des étapes précédentes. La preuve est utilisée pour justifier les abstractions réalisées pour le *model-cheking*. Le *model-checking* aide à trouver les invariants à intégrer dans la preuve. A la fin de ce processus itératif, il donne un contre-exemple indiquant comment la propriété est violée, ou une preuve que cette propriété est satisfaite.

Besoin de scénarios : Lorsque l'on associe de la vérification ou de la preuve au test, cela veut dire que l'on a un modèle, soit des spécifications du système, soit de sa conception. La vérification ou la preuve peut permettre de déterminer des comportements dangereux du système final, susceptibles de violer une propriété qui doit être vérifiée. Lorsque l'on s'intéresse plus particulièrement à l'aspect discret du comportement, on va être amené à s'exprimer sous la forme de séquences d'événements. Face à l'explosion combinatoire des séquences induites par le parallélisme, on ressent le besoin de se fonder sur des ordres partiels pour décrire de façon compacte un grand nombre de séquences. Mais dans le domaine du test, tout comme dans celui de la vérification ou de la preuve, cette notion n'est prise en compte que de façon ponctuelle, l'outil de base restant la notion de séquence.

Avant de détailler notre travail, nous allons présenter rapidement un certain nombre de travaux qui comme nous s'intéressent à la sûreté de fonctionnement et la vérification des systèmes pilotés par calculateurs et qui d'une façon ou d'une autre se sont focalisés sur l'aspect discret du comportement dynamique.

1.6. Tour d'horizon : sûreté et recherche des scénarios

Dans ce tour d'horizon, nous regroupons les travaux qui sont proches de notre problématique de recherche de scénarios soit dans le cadre de la sûreté de fonctionnement, soit dans le cadre de la vérification.

1.6.1. Travaux de J.L Chabot

Ses travaux [Chabot 98] proposent une méthode pour le calcul de la probabilité d'extinction d'un scénario d'incendie. J.L Chabot se base sur la simulation hybride de ces scénarios. Ces scénarios sont caractérisés par l'intervention des aspects discrets et continus en interaction, ce qui situe ces travaux dans le cadre de la fiabilité dynamique. Les aspects discrets regroupent les défaillances des systèmes de détection et de lutte ainsi que l'intervention humaine. L'aspect continu regroupe les dimensions, le développement et les effets du feu. L'interaction mutuelle entre ces deux

aspects justifie l'intérêt d'une modélisation hybride. L'augmentation de la température dans les locaux et l'apparition de fumées ont une incidence directe sur la sollicitation des protections et sur l'intervention humaine à partir du franchissement d'un certain seuil. La réussite ou l'échec de ces actions ont une influence sur le développement de l'incendie.

Chabot a mis l'accent sur la vue discrète en utilisant des modèles RdP stochastiques temporisés synchronisés (supporté par l'outil MOCA_RP). Il les a mis en œuvre sur un problème classique de régulation de niveau d'un liquide dans un réservoir. Notons que l'introduction de l'exemple du réservoir comme illustration de techniques utilisées en fiabilité dynamique est du à Aldmir [Aldmir 87]. Le but de l'étude de Chabot est de déterminer les fréquences de débordement et d'assèchement du réservoir pour différentes durées de fonctionnement. L'évaluation de ces fréquences est faite par la simulation de Monte Carlo.

Une première méthode consiste à prendre en compte les variations du niveau du liquide en le discrétisant et en associant à chaque variation élémentaire (Δh) l'apparition ou la disparition d'un jeton de la place modélisant le niveau. Si la variation (Δh) correspond par exemple à 10 cm et le niveau initial du réservoir est égal à 4 mètres, la place modélisant ce niveau contiendra initialement 40 jetons. L'état redouté correspond soit à l'assèchement du réservoir quand il y a moins de 30 jetons dans la place correspondant au niveau, soit au débordement dans le cas ou le nombre de jetons est au-delà de 70 dans la même place. Cela revient donc à passer à un modèle complètement discret avec tous les problèmes que cela peut poser (impact de la discrétisation sur le comportement, précision des résultats et le risque d'explosion combinatoire des états).

Une deuxième méthode consiste à introduire un nouveau type de réseaux de Petri pour modéliser le comportement du réservoir associant des phénomènes discrets et continus. Deux types de transitions ont été introduits pour prendre en compte l'évolution du volume : les transitions dites d'activité continue et les transitions dites de jonction. Ces dernières mettent à jour les paramètres des équations algébriques ou différentielles. Quant aux transitions d'activités continues, lorsqu'elles sont sensibilisées, MOCA-RP exécute un algorithme de calcul qui donne leur délai de temporisation avant le franchissement. L'aspect hybride est donc mieux pris en compte.

Discussion : ces travaux ont montré la bonne adéquation des réseaux de Petri dans la modélisation des systèmes dynamiques hybrides non réparables. Le couplage entre un simulateur discret et un calcul continu permet de conserver la modélisation de la partie continue ainsi que de la partie discrète et de diminuer les hypothèses simplificatrices.

Le cas des scénarios d'incendie est un exemple de systèmes dynamiques hybrides et la simulation de Monte Carlo donne des résultats satisfaisants pour estimer la probabilité d'extinction du feu. Le temps de simulation est raisonnable car on n'est pas confronté à la problématique des événements rares. En effet on ne simule en fait que les scénarios redoutés. Les différents types d'incendie sont en effet directement modélisés. De plus, les scénarios étant très séquentiels, l'explosion combinatoire des états reste raisonnable.

1.6.2. Méthode des graphes de flux dynamiques

La méthode des graphes de flux dynamiques (Dynamic Flowgraph Methodology DFM) [Garret 93] est une méthode de modélisation et d'analyse pour l'étude des risques des systèmes embarqués. Elle a été appliquée dans les systèmes aérospatiaux et nucléaires. Cette approche consiste à identifier les scénarios redoutés et à élaborer une stratégie de test des systèmes. La DFM permet de prendre en compte l'aspect dynamique des systèmes embarqués et l'interaction existante entre leur partie logicielle et leur partie matérielle.

La méthode DFM est composée de deux étapes : une étape de modélisation et une étape d'analyse. L'étape modélisation consiste à construire un modèle, sous forme d'un réseau de nœuds, exprimant le comportement logique et dynamique du système. Ces nœuds sont reliés entre eux par des connecteurs pour exprimer les relations de causalité et temporelles entre les variables physiques de la partie opérative et les paramètres de commande du système de contrôle. Après cette étape de modélisation, débute l'étape d'analyse qui a pour but de déterminer comment le système atteint un état donné (normal ou indésirable). L'analyse se fait en inversant les relations de causalité à partir du modèle DFM pour construire les Arbres de Défaillance temporisés. Ces derniers peuvent exprimer les combinaisons booléennes d'événements (défaillance ou autres) pouvant amener le système dans un état donné comme les AdD classiques ainsi que les scénarios redoutés sous la forme de séquence d'occurrence d'événements.

Discussion : cette approche permet de déterminer comment un système peut atteindre un état donné. La modélisation permet la prise en compte explicite des interactions entre la partie opérative et la partie commande. Une fois un arbre de défaillance fait pour un modèle par rapport à un état redouté donné, la modification de modèle implique souvent de refaire un autre AdD, ce qui est souvent lourd. Or, dans cette approche l'unicité du modèle et la construction automatique des AdD donne une grande flexibilité aux concepteurs, puisqu'il suffit de faire les modifications sur le modèle et de générer automatiquement les AdD pour plusieurs états redoutés. Un autre avantage de cette méthode est qu'elle est basée sur la construction d'AdD (temporisés) bien connus dans la communauté des fiabilistes.

Cependant, l'inconvénient majeur de cette méthode est que l'on est vite confronté au problème de l'explosion combinatoire due à la discrétisation de toutes les variables du système étudié quand il s'agit de système de taille industrielle.

1.6.3. Travaux de G.Moncelet

La thèse de G.Moncelet [Moncelet 98] traite d'une manière qualitative et quantitative la sûreté de fonctionnement des systèmes mécatroniques. Ses travaux portent sur la détermination des séquences d'événements redoutés et l'estimation de leurs probabilités d'occurrence par la simulation de Monte Carlo. La démarche consiste à modéliser qualitativement le système à étudier avec le formalisme des réseaux de Petri colorés [Jensen 92] et générer le graphe des marquages accessibles (graphe d'occurrence). Tous les chemins menant vers un état redouté sont ensuite identifiés et caractérisés en termes d'enchaînements d'actions et de changement d'états des composants concernés du système. L'estimation de la probabilité d'occurrence de ces scénarios redoutés se fait par une simulation de Monte Carlo du modèle quantitatif du système.

Compte tenu de la complexité des systèmes mécatroniques, les résultats auxquels ont abouti les travaux de Gilles Moncelet sont principalement deux. Le premier est que la simulation de Monte Carlo du modèle quantitatif nécessite un temps prohibitif. Le deuxième est que l'analyse quantitative est confrontée au problème de l'explosion combinatoire du nombre d'états du graphe d'occurrence due à la discrétisation des variables continues.

En ce qui concerne l'analyse quantitative, la rareté des scénarios redoutés rend inefficaces les méthodes basées sur la simulation de Monte Carlo du modèle global. En effet le système passe la majorité de temps à simuler le fonctionnement normal qui n'apporte aucune information sur la sûreté de fonctionnement puisque seule nous intéresse la réaction du système face à l'occurrence d'une ou plusieurs défaillances. Ceci est dû au fait que le modèle décrit le système complètement, c'est-à-dire aussi bien son fonctionnement normal que son fonctionnement dans des situations critiques.

Discussion : en plus de la bonne prise en compte de l'aspect hybride, la principale contribution de G.Moncelet est d'avoir montré la nécessité de séparer les études de la sécurité des systèmes mécatroniques en deux parties : analyse qualitative suivie d'une analyse quantitative. L'analyse qualitative s'effectue à partir de la génération du graphe d'accessibilité et la simulation de Monte Carlo permet d'estimer les probabilités d'occurrence de ces scénarios. Le principal obstacle de l'analyse qualitative est l'explosion combinatoire du nombre d'états du graphe, et l'analyse quantitative souffre des temps de simulation prohibitifs dus à la prise en compte par discrétisation de la partie continue. En raison de l'entrelacement, les relations de cause à effet entre événements sont mal exprimées, ce qui rend l'extraction des scénarios redoutés à partir du graphe d'occurrence inefficace. Il est donc nécessaire d'éviter le passage par ce graphe.

1.6.4. Travaux de Raphaël Schoenig

Les travaux de thèse de [Schoenig 04] réalisés en collaboration entre le laboratoire CRAN et GFI Consulting s'inscrivent dans la lignée de travaux en sûreté de fonctionnement des systèmes mécatroniques, à savoir définir une méthodologie de conception des systèmes de contrôle-commande qui intègre les activités de conception tout au long des différents cycles de vie. L'objectif d'une méthodologie, basée sur les réseaux de Petri interprétés de commande, est d'apporter une garantie sur la qualité des systèmes développés, tout en considérant les contraintes de coût et de temps. Le principe général de ces travaux consiste à exploiter un modèle exprimé dans le formalisme des réseaux de Petri. Ces travaux sont orientés vers une méthode graphique et analytique : les graphes de Markov. De plus, la construction d'un tel modèle est facilitée de par son analogie avec les formalismes à états transitions (Statecharts, réseaux de Petri). Néanmoins, il est reconnu que la démarche traditionnelle souffre de certaines limitations liées à l'explosion combinatoire du nombre d'états, et d'hypothèses restrictives. Les méthodes de simplification trouvées dans la littérature semblent être inapplicables au niveau industriel. La méthode markovienne est adoptée afin que celle-ci puisse être facilement compréhensible. A partir d'un système embarqué quelconque, le graphe de Markov illustrera par exemple les différents modes de fonctionnement existant (modes nominaux, dégradés, états redoutés), et la façon d'y accéder. L'objectif principal de la méthode quantitative est de répondre à la problématique de représentation et d'évaluation de la fiabilité des SDH (Systèmes Dynamiques Hybrides). La complexité d'évaluation des attributs de sûreté de fonctionnement est liée à l'interaction permanente entre le comportement du système, et le processus de défaillance. Le principe général

de l'approche consiste à coupler les deux méthodes d'analyse classiques : la simulation et les graphes de Markov, afin de contourner les limites intrinsèques à ces méthodes (temps de simulation, explosion combinatoire). L'approche proposée consiste à réaliser une agrégation des états élémentaires d'un graphe de Markov. L'influence du système sur le processus de défaillance est intégrée dans le modèle par le biais de termes de pondération des taux de défaillance. Ceux-ci permettent de prendre explicitement en compte la dynamique interne du système. Ces coefficients sont évalués par une simple simulation, permettant ainsi de traiter des systèmes complexes. Ce type d'approche, qualifié de « dynamique », permet de traiter des systèmes fortement dépendant du temps (par exemple, en cas d'existence de reconfigurations matérielles ou logicielles).

Discussion : l'objectif de ces travaux est de développer une méthodologie de conception qui permet d'intégrer des nouvelles méthodes théoriques et des outils d'analyse fonctionnelle et de sûreté de fonctionnement pour des systèmes de plus en plus complexes. Ces méthodes, autrefois réservées au domaine universitaire, ne peuvent être bénéfiques que si elles s'intègrent dans un cadre méthodologique bien défini. Il n'est pas envisageable de bouleverser radicalement les habitudes de concepteurs, ni de les introduire brutalement sans se soucier de leur cohérence dans la démarche de conception.

Ces travaux ont consisté précisément à identifier les éléments permettant d'améliorer la démarche de développement en tenant compte de ce constat, à savoir : le choix d'un formalisme de modélisation, la vérification et la validation d'un modèle et l'évaluation de la sûreté de fonctionnement. L'objectif était également de prendre en compte la spécificité hybride des systèmes mécatroniques et la nature temps réel des systèmes de contrôle-commande. Ces travaux ont mis en évidence l'intérêt de l'utilisation du formalisme des réseaux de Petri interprétés de commande. Ce formalisme est en adéquation avec la nature discrète des systèmes de commande, et présente de nombreux avantages : son caractère graphique améliore la clarté des modèles, sa capacité à faciliter la vérification de propriétés par des analyses algébriques, la possibilité de représenter des défaillances grâce aux transitions stochastiques (pour les études de sûreté de fonctionnement) en font un outil puissant de modélisation et d'analyse.

L'accent a été également mis sur la simulation et l'avantage procuré par les méthodes formelles telles que le model-checking à propos de leur capacité à éliminer les fautes de conception.

1.6.5. Travaux de Claudia Betous-Almeida

Les travaux de [Betous 02] sont consacrés au développement d'une méthode de modélisation par affinements, de la sûreté de fonctionnement de systèmes complexes intégrant des composants logiciels et matériels du commerce (COTS *Commercial Off The Shelf*). L'approche proposée est basée sur les réseaux de Petri stochastiques généralisés et comporte trois étapes. La première, consiste en la construction d'un modèle basé uniquement sur les spécifications fonctionnelles de l'application, nommé modèle fonctionnel. Ensuite, à partir de ce modèle et de l'architecture du système, un modèle de sûreté de fonctionnement de haut niveau est construit. Les règles de construction de l'interface entre le modèle fonctionnel et le modèle structurel sont développées et présentées. Des règles d'affinement par étapes sont également utilisées. Ces règles permettent la transformation du modèle de haut niveau en un modèle plus détaillé. L'affinement peut être fait selon trois perspectives : décomposition des composants, affinement des états ou des événements et mise au point des distributions. Pour rendre plus efficace la modélisation de systèmes, une

bibliothèque composée de modèles de base a été crée. En effet, ces modèles, moyennant un minimum de modifications, ont permis de modéliser tous les composants de tous les systèmes étudiés dans ce travail. Cette approche a été illustrée dans le cadre de la sélection d'un système de contrôle-commande parmi les propositions de divers constructeurs en réponse à un appel d'offre, mais elle peut également être utilisée comme support au développement et à la validation de systèmes. Cette approche est appliquée à des systèmes de contrôle-commande des centrales nucléaires.

Discussion : cette approche a un double objectif :

- Modélisation homogène (au même niveau de détail) de plusieurs systèmes qui correspondent à des solutions possibles pour mettre en œuvre les spécifications fonctionnelles, pouvant être proposées par différents constructeurs.
- Une modélisation de plus en plus fine d'un même système, dès sa spécification fonctionnelle jusqu'à son exploitation.

Ce que nous retenons de ces travaux, c'est l'intérêt de concevoir l'architecture du système (ensemble d'objets ou d'entités communicants) de façon explicite dans le modèle et le fait que les réseaux de Petri sont bien adaptés dans ce cadre.

1.6.6. Langage de modélisation AltaRica

Le langage AltaRica est issu d'une collaboration entre le LaBRI (Laboratoire Bordelais de Recherche en Informatique) et plusieurs industriels en 1996 dans le but d'établir divers ponts entre : la sûreté de fonctionnement et les méthodes formelles, les analyses quantitatives des dysfonctionnements et les analyses qualitatives des comportements fonctionnels, afin de fournir aux ingénieurs concepteurs de systèmes complexes, un atelier outillé. Il est ainsi possible à partir d'une description dans ce langage, basé sur les automates à contraintes, de simuler un modèle et de générer un arbre de défaillance ou de vérifier qu'il satisfait des propriétés logiques. L'objectif depuis 1996, est de développer un outil générique permettant l'analyse des systèmes incorporant :

- un langage de spécification de haut niveau AltaRica [Griffault et al 98]
- un model-checker MecV [Arnold et al 94]
- des outils de sûreté de fonctionnement (Aralia [Aralia 96])

Aujourd'hui, le projet AltaRica s'est élargi à un niveau national et est le fruit de plusieurs collaborations : industrielles (la société d'ingénierie GFI, Dassault Aviation, Total, Schneider Electric, France Telecom, Thales systèmes aéroportés) et universitaires (LaBRI, IML, IRCCyn et ONERA).

Dans l'atelier AltariRica, un modèle AltaRica est transformé en des formalismes de plus bas niveau, comme les systèmes de transitions, et peut ainsi être vérifié selon la méthode du model checking.

Le model-checker MecV dédié aux modèles AltaRica prend en entrée des modèles AltaRica et permet de vérifier des propriétés logiques sur leurs comportements. En particulier, il est possible

de vérifier si deux modèles AltaRica ont les mêmes comportements. Un autre atout important du langage est la possibilité d'exprimer des propriétés entre événements.

Discussion : l'atelier AltaRica est une réponse à certains aspects du contrôle de processus. Des limites ont été rencontrées dans l'expressivité du modèle et la nécessité d'ajouter des opérateurs de modélisation temps réel est apparue. On peut citer Timed AltaRica, Hybrid AltaRica, et les travaux de [Pagetti 04] sur l'extension temps réel d'AltaRica. On peut souligner également qu'on peut générer des arbres de défaillance dans certains cas avec AltaRica.

1.6.7. Travaux de Nicolas Rivière

L'objectif des travaux de [Rivière 03] réalisés au LAAS-CNRS est de contribuer à l'élaboration de méthodes d'aide à la conception de systèmes coopératifs en prenant en compte les contraintes temporelles de manière quantitative. Son approche est fondée sur les réseaux de Petri, la logique linéaire et les graphes de contraintes temporelles. C'est une approche orientée « événements » et non orientée « états » comme c'est souvent le cas dans les approches fondées sur les réseaux de Petri. Elle est décomposée en deux étapes : une étape d'analyse « qualitative » et une étape d'analyse « quantitative ».

La première consiste à obtenir les relations de causalité entre les événements appartenant à un scénario donné. L'équivalence entre un arbre de preuve en logique linéaire et le processus fini obtenu par dépliage d'un réseau de Petri à partir du même marquage initial montre que ces relations sont des relations de précédence. L'introduction de la notion de séquent caractéristique permet de mettre en oeuvre une approche compositionnelle des processus à partir des règles du calcul des séquents.

La deuxième étape consiste à passer du graphe décrivant les relations de précédence à un graphe de contraintes temporelles exprimant de façon linéaire l'ensemble des contraintes temporelles quantitatives que doivent vérifier les dates des franchissements des transitions dans un scénario. Cette démarche est complètement cohérente avec les réseaux de Petri p-temporels mais difficilement compatible avec les t-temporels, car ils engendrent des ensembles de contraintes qui sont plus complexes.

Discussion : les méthodes orientées événement consistent à observer les événements et analyser comment, quand et dans quel ordre ils apparaissent (ou dans combien de temps on atteint un certain état dans le cadre d'un scénario). Ces méthodes permettent de vérifier le respect des contraintes logiques de synchronisation et les échéances temporelles quantitatives. Les contraintes temporelles quantitatives sont des contraintes supplémentaires que doivent vérifier les scénarios de franchissement de transitions. Pour pouvoir vérifier les contraintes temporelles entre les événements, il faut d'abord générer les scénarios de franchissement de transitions.

Ce que nous retenons de ces travaux c'est qu'un outil comme la logique linéaire peut à partir d'un réseau de Petri générer des comportements, sous la forme d'ordres partiels entre événements. On peut ainsi s'affranchir de la représentation classique basée sur des séquences (ordre total) et lutter contre l'explosion combinatoire.

1.6.8. Travaux de Sarhane Khalfaoui

La thèse de [Khalfaoui 03] réalisée au LAAS en collaboration avec PSA (Pegeot Citroën) se focalise sur l'analyse qualitative de la sécurité des systèmes mécatroniques en vue de l'obtention des scénarios redoutés. La connaissance de ces scénarios permet d'évaluer leur probabilités d'occurrence et de valider les lois de reconfiguration pour orienter le choix des concepteurs. Ces travaux ont permis de développer une méthode de recherche de scénarios redoutés basée sur la logique linéaire et le formalisme de réseau de Petri et d'un ensemble d'équations différentielles pour modéliser le système. Les scénarios sont ainsi extraits directement de modèle Réseau de Petri (RdP) sans générer le graphe d'accessibilité associé. Cette méthode ne se réduit pas à l'utilisation d'un joueur de réseau de Petri, elle est équivalente à une recherche systématique des causalités qui serait bien sur des processus finis obtenus par dépliage de réseaux de Petri. Mais dans le cas de l'analyse de la sécurité des systèmes mécatroniques, seul l'état final du scénario (l'état redouté) est connu. L'état initial (dernier état de fonctionnement normal) et la liste des événements conduisant à l'état redouté doivent être découverts. La méthode mise au point permet, en partant de l'état redouté, de revenir en arrière à travers la chaîne des relations de cause à effet et d'extraire tous les scénarios possibles menant vers cet état. Ainsi, les événements constitutifs des scénarios sont découverts progressivement et seuls les événements nécessaires pour arriver à l'état redouté sont pris en considération. La démarche permet ainsi de focaliser la recherche sur les parties intéressantes du modèle pour l'analyse de la sûreté de fonctionnement. De plus comme la démarche est fondée sur une sémantique opérationnelle du parallélisme vrai comme [Rivière 03], chaque scénario est donné sous la forme d'un ordre partiel entre les évènements. Cela diffère d'un arbre de défaillance qui donne un ensemble de combinaisons statiques des états partiels nécessaires pour l'obtention de l'état redouté. Cette approche a conduit au développement d'un algorithme pour générer automatiquement des scénarios critiques à partir d'un modèle RdP.

Une caractéristique importante est qu'au niveau de la modélisation, la nature hybride et dynamique des systèmes mécatroniques est respectée par le choix d'une modélisation associant RdP et équations différentielles : les RdP Prédicats Transitions Différentielles Stochastiques (RdP PTDS). Le modèle RdP décrit le fonctionnement nominal, les défaillances et les mécanismes de reconfiguration. Les équations différentielles représentent l'évolution des variables continues de la partie énergétique du système. L'avantage est de séparer clairement l'aspect continu de l'aspect discret.

Discussion : le choix des RdP PTDS pour la modélisation de l'aspect discret et continu des systèmes mécatroniques est en bonne adéquation avec la nature hybride de ces systèmes. Le problème d'explosion combinatoire est évité par le fait d'extraire les scénarios directement à partir d'un modèle RdP du système sans passer par le graphe d'accessibilité et en s'appuyant sur les arbres de preuves en Logique Linéaire qui permettent de gérer les ordres partiels.

Ce travail était une première approche et l'algorithme de construction des scénarios ne prend en compte que la vue discrète du système. Or il s'avère, même sur des exemples assez simples, que de nombreux scénarios produits ainsi ne vérifient pas les contraintes provenant de la vue continue du système hybride. Ils sont contradictoires avec la dynamique de la partie énergétique du système. Il semble plus efficace d'en éliminer, au moins un certain nombre, dès la construction des scénarios. Cette amélioration a été l'une des préoccupations majeures de notre travail de thèse.

Il faut souligner qu'il existe de nombreux autres travaux qui se sont intéressés de très près à la recherche de scénarios dans les systèmes hybrides. Notamment le cas de ce qui touche aux études probabilistes de sûreté (EPS ou PSA pour Probabilistic Safety Assessment), principalement dans les domaines nucléaire et aéronautique.

1.7. Conclusion et Analyse

Nous avons vu dans ce chapitre que dans le domaine de la sûreté de fonctionnement on trouvait trois volets interdépendants : la vérification de modèle, la preuve de propriété et le test. Si l'on s'intéresse à des systèmes hybrides ayant une forte composante discrète, c'est souvent le cas à cause des procédures de reconfiguration, nous avons vu qu'il y avait un besoin : celui de développer des outils et des techniques fondés sur la notion d'ordre partiel plutôt que sur celle de séquence. Une raison importante est de diminuer l'explosion combinatoire due au parallélisme. Une autre raison est liée à la compréhension. S'il y a une relation d'ordre, elle ne provient pas d'un hasard temporel ("il se trouve que e_1 apparaît après e_2"), mais elle découle d'une relation de causalité ("e_2 est un effet de e_1"). La relation d'ordre entre les événements est donc une *explication* du comportement et va au-delà d'une simple observation.

Les divers travaux que nous avons rapidement introduits nous ont confortés dans l'idée que les réseaux de Petri étaient un outil naturel dans le cadre de ce que nous voulions faire. Comme l'a montré par exemple [Betous 02], ils ne sont pas du tout contradictoires avec la prise en compte de l'architecture du système et peuvent s'intégrer dans une démarche très structurée. Enfin [Rivière 03] nous a montré qu'ils pouvaient être associés à une démarche logique pour extraire des relations d'ordre partiel entre événements.

C'est pourquoi nous avons choisi de poursuivre la voie ouverte par [Khalfaoui 03]. Le formalisme RdP PTDS proposé par [Khalfoui 03] et choisi dans ses travaux modélise au mieux l'aspect discret et continu des systèmes pilotés par calculateurs. Son approche basée sur l'extraction des scénarios directement à partir d'un modèle RdP du système est intéressante et semble bien adaptée pour une analyse qualitative de la sûreté de fonctionnement des systèmes pilotés par calculateurs d'un point de vue fiabilité dynamique.

Chapitre 2 : Modélisation des systèmes hybrides et notion de scénario

2.1. Introduction

Dans le chapitre précédent, nous avons montré la bonne adéquation des réseaux de Petri à la modélisation des systèmes dynamiques hybrides. Cet outil retenu dans nos travaux permet de modéliser l'aspect discret de nos systèmes pilotés par calculateurs. Dans ce chapitre, nous étudierons différents formalismes, à base de réseaux de Petri, qui ont été étendus afin de prendre en compte l'aspect temporel et hybride des systèmes complexes. Nous introduirons également la notion de scénario redouté, sa caractérisation et ses différentes propriétés (Minimalité).

2.2. Modélisation des systèmes temporisés et hybrides

Les études de Sûreté de Fonctionnement des systèmes pilotés par calculateurs nécessitent la prise en compte de l'aspect discret et de l'aspect continu. Le chapitre précédent nous a montré que les réseaux de Petri étaient bien adaptés pour la représentation de la partie discrète. Nous allons ici discuter un peu plus en détail la prise en compte du temps et celle d'une dynamique continue. En effet, le temps peut être vu comme un premier pas pour, à partir d'un modèle à événements discrets, introduire l'aspect continu.

Nous présentons, dans ce paragraphe, les réseaux de Petri temporisés, les réseaux de Petri temporels, les réseaux de Petri hybrides (les aspects discrets et continus sont pris en compte par un seul formalisme) et les réseaux de Petri couplés avec des équations différentielles (les aspects discrets et continus sont pris en compte par deux formalismes interagissant l'un sur l'autre).

2.2.1. Modélisation avec les RdP temporisés et les RdP temporels.

Pour prendre en compte les contraintes temporelles quantitatives, il existe plusieurs extensions temporelles des réseaux de Petri : le temps est associé aux transitions, aux places ou aux arcs. Le temps est associé de deux manières : soit c'est une durée associée à un type de nœud (transition, place ou arc) et nous parlons de réseau « temporisé », soit c'est un intervalle temporel et nous parlons alors de réseau « temporel ».

Nous présentons les cas où le temps est associé aux transitions ou aux places.

2.2.1.1 Les RdP t-temporisés

Définition 1 (Réseau de Petri t-temporisé) *Un réseau de Petri t-temporisé est une paire N_{tl}=<N, D> où N est un réseau de Petri <P, T, M> marqué et D est une fonction qui à chaque transition t_i du réseau fait correspondre une durée de temps d_{ti}. Dans le triplet <P, T, M>, P est l'ensemble des places du réseau, T l'ensemble des transitions et M le marquage.*

La durée de temps est associée aux transitions. Quand une transition t_i est franchie, un temps d_{ti} s'écoule entre la consommation des jetons en entrée de la transition et la production des jetons en sortie. Pendant la durée d_{ti} le jeton est réservé et ne peut plus être consommé par le franchissement d'une autre transition. Si la représentation de la durée d'activités de façon quantitative ne pose pas de problème, il n'est par contre pas possible de représenter une défaillance qui interrompe une activité déjà commencée. Ce modèle est donc inadapté aux études de sûreté de fonctionnement.

2.2.1.2 RdP P-temporisés

Définition 2 (Réseau de Petri p-temporisé) *Un réseau de Petri p-temporisé est une paire N_{tl}=<N, D> où N est un réseau de Petri <P, T, M> marqué et D est une fonction qui à chaque place p_i du réseau fait correspondre une durée de temps d_{pi}.*

La durée de temps est associée aux places. Chaque jeton doit rester au moins d_{pi} unités de temps dans un état non disponible quand il arrive dans la place p_i avant d'être consommé par un franchissement de transition. Cette technique de modélisation est en fait équivalente à la précédente. Elle présente le même inconvénient : un jeton non disponible ne peut être consommé et il est donc impossible de représenter une activité interrompue par une défaillance.

2.2.1.3 RdP t-temporels

Définition 3 (Réseau de Petri t-temporel) *Un réseau de Petri t-temporel est une paire N_{tl}=<N, D> où N est un réseau de Petri <P, T, M> marqué et D est une fonction qui à chaque transition t_i du réseau fait correspondre un intervalle statique de temps $d(t_i) = [d_{imin}(t_i), d_{imax}(t_i)]$ qui décrit une durée de sensibilisation.*

L'intervalle de temps est associé aux transitions. La transition t_i doit rester sensibilisée durant au moins d_{imin} unités de temps et au plus d_{imax} unités de temps avant d'être franchie. Par contre les jetons peuvent à tout moment être consommés par une autre transition et l'on peut donc représenter des défaillances.

2.2.1.4 RdP p-temporels

Définition 4 (Réseau de Petri p-temporel) *Un réseau de Petri p-temporel est une paire N_{tl} = <N, D> où N est un réseau de Petri <P, T, M> marqué et D est une fonction qui à chaque place p_i du réseau fait correspondre un intervalle statique de temps $d(p_i) = [d_{imin}(p_i), d_{imax}(p_i)]$ qui décrit une durée de séjour des jetons dans p_i.*

L'intervalle de temps est associé aux places. Chaque jeton doit rester au moins d_{imin} unités de temps dans la place p_i et doit être consommé par un franchissement de transition avant d_{imax} au plus tard. La durée pendant laquelle le jeton restera dans la place peut prendre n'importe quelle valeur du domaine temporel $[d_{imin}(p_i), d_{imax}(p_i)]$. Avant d_{imin}, le jeton est indisponible pour toutes les transitions de sorties de la place p_i. Une transition de sortie de la place p_i peut être franchie en consommant ce jeton pendant sa fenêtre de visibilité. Elle ne sera plus franchissable en consommant ce jeton lorsque la durée écoulée est supérieure à d_{imax} : le jeton est dit *mort* et cela symbolise un viol *de contraintes temporelles*. Il y a en fait une horloge par jeton du marquage courant.

A l'intérieur d'une fenêtre de visibilité, un jeton peut être consommé pour représenter une défaillance, mais pas en dehors. Par contre, dans le cadre de la modélisation il est souvent plus naturel d'associer une activité (un processus continu par exemple) à une place qu'à une transition.

Les RdP temporels sont plus généraux que les RdP temporisés. Ils permettent de décrire certains mécanismes que les RdP temporisés ne permettent pas comme les chiens de garde. Pour passer d'un RdP temporisé à un RdP temporel, il suffit de remplacer chaque transition t (dans le cas de RdP t-temporisé) du premier par une séquence $[t_1, t2]$ avec :

$$d_{1min}(t_1) = d_{1max}(t_1) = 0$$
$$d_{2min}(t_2) = d_{2max}(t_2) = d_{ti}$$

Avec le même principe, on peut passer d'un RdP p-temporisé à un RdP p-temporel. D'autre part les réseaux de Petri ordinaires (sans implication du temps) correspondent simplement au cas où tous les intervalles de sensibilisation sont égaux à $[0, \infty [$. Comme les systèmes pilotés par calculateurs peuvent être défaillants à tout instant et peuvent être réparés, l'intervalle $[0, \infty [$ sera associé aux transitions modélisant les défaillances et les réparations. Nous aurons donc une approche où le système est décrit par des réseaux p-temporels mais avec des transitions t-temporelles pour les défaillances.

2.2.1.5 RdP arc-pt-temporels

Définition 5 (Réseau de Petri arc-pt-temporel) *Un réseau de Petri arc-pt-temporel est une paire* $N_{tl}=<N,D>$ *où N est un réseau de Petri* $<P,T,M>$ *marqué et D est une fonction qui à chaque arc* (p_i, t_i) *reliant la place* p_i *à la transition* t_i *du réseau fait correspondre un intervalle statique de temps* $d(p_i, t_i)=[d_{imin}(p_i, t_i), d_{imax}(p_i, t_i)]$ *qui décrit une durée de séjour des jetons entre la place* p_i *et la transition* t_i.

L'intervalle de temps est associé aux arcs. Chaque jeton doit rester après sa consommation de la place p_i au moins d_{imin} unités de temps dans l'arc reliant cette place à la transition de franchissement t_i et doit être consommé par un franchissement de transition avant d_{imax} au plus tard. La durée pendant laquelle le jeton restera dans l'arc (p_i, t_i) peut prendre n'importe quelle valeur du domaine temporel $[d_{imin}(p_i), d_{imax}(p_i)]$. Avant d_{imin}, le jeton est indisponible pour toutes les transitions de sortie de la place p_i. Le jeton est invisible pendant l'intervalle temporel $[d_{imin}(p_i), d_{imax}(p_i)]$. Ce modèle correspond mieux à ce que nous voulons modéliser, en effet l'évolution de la dynamique continue commence à d_{imin} le temps nécessaire pour atteindre le seuil inférieur et qui correspond à la date de déplacement de jeton de la place p_i et de son séjour dans

l'arc (p_i, t_i). La dynamique continue atteint son seuil supérieur à d_{imax} la date de franchissement au plus tard de la transition sensibilisée par ce jeton.

Dans les réseaux de Petri arc-pt-temporels, une transition t_i est franchissable s'il existe une date pour laquelle tous ses arcs entrants permettent sa sensibilisation. Si une transition possède plusieurs arcs d'entrée, son intervalle temporel de sensibilisation correspond à l'intersection des intervalles temporels de l'ensemble de ses arcs d'entrée.

Les jetons peuvent à tout moment être consommés par une autre transition et l'on peut donc représenter des défaillances. Un intervalle de temps $[0, \infty[$ est associé aux arc reliant une place avec une transition modélisant une défaillance.

2.2.2. Modélisation avec les RdP hybrides

2.2.2.1 RdP hybride

Les réseaux de Petri hybrides [David et Alla 89] permettent de représenter, d'une façon unifiée, la dynamique discrète par des places et des transitions ordinaires, et la dynamique continue par des places (dites continues) dont le marquage est un nombre réel (et non plus un nombre entier) positif ou nul, et des transitions (continues) qui correspondent à des écoulements continus. Une transition continue t_i est franchie avec une vitesse $V(t)$. Cela signifie qu'entre t et $t + dt$ une quantité $v(t).dt$ de marques est enlevée de sa place amont et est ajoutée à la quantité de marques de sa place avale. On peut ainsi décrire des variables qui évoluent de façon continue et linéaire en fonction du temps.

L'influence de la partie discrète sur la partie continue se fait par l'intermédiaire de boucles élémentaires reliant des places discrètes à des transitions continues. L'influence de la partie continue sur la partie discrète (franchissement de seuils par des variables continues) se fait par des boucles élémentaires reliant des places continues (représentant les variables testées) à des transitions discrètes.

Les RdP hybrides permettent une représentation homogène des aspects continus (flux continus) et discrets dans le même formalisme, ce qui leur permet de représenter facilement des systèmes de transfert de liquides ou des systèmes de production manufacturiers où le flux des pièces est approximé par un flux continu [Allam 98]. Cependant lorsqu'on introduit des contraintes algébriques, il faut rajouter des équations au modèle et effectuer une mise à jour régulière des variables. Nous perdons ainsi l'intérêt principal des RdP hybrides qui, de ce fait, ne peuvent représenter dans un unique formalisme l'ensemble de la partie discrète et de la partie continue. De plus, il est impossible de prendre en compte des équations différentielles générales.

2.2.2.2 Les RdP Prédicats-Transitions Différentiels Stochastiques

Alors que dans l'approche précédente les aspects discrets et continus sont pris en compte de façon intégrée, dans l'approche que nous allons présenter maintenant ils sont pris en compte par deux formalismes différents, chacun étant bien adapté à l'aspect visé [Andreu 96].

En effet, le comportement hybride est pris en compte en associant les variables représentant l'état continu à des jetons et en associant à chaque place un ensemble d'équations algébro-différentielles (algébriques et/ou différentielles), on parle d'un *réseau de Petri Prédicats-Transitions Différentiels* (RdP PTD) développée par [Champagnat 98]. Un jeton mis dans une place déclenche l'intégration des équations correspondantes. Parallèlement à l'intégration, plusieurs seuils sont surveillés. Chaque seuil surveillé est associé à une transition aval d'une place marquée. Quand le seuil est franchi, cela signifie que l'événement attendu correspondant est apparu, et la transition associée à ce seuil est franchie. Un nouveau marquage est calculé et l'intégration du nouveau système démarre. Si deux jetons sont dans la même place, les variables de chacun d'eux évoluent, indépendamment les unes des autres, selon les mêmes équations associées à la place. Le mécanisme de défaillance et de reconfiguration est pris en compte en ajoutant l'aspect stochastique aux RdP PTD. Cette extension réalisée par [Khalfoui 03] donne naissance aux *RdP Prédicats-Transitions Différentiels Stochastiques* (RdP PTDS) qui est le formalisme que nous avons retenu pour la modélisation des systèmes pilotés par calculateurs.

Le modèle RdP PTD associe à toute transition une fonction de sensibilisation déterministe : la transition est franchie exactement à la date à laquelle le seuil est atteint. Dans le cas des RdP PTDS, nous distinguons deux types de transitions : les *transitions déterministes* et les *transitions stochastiques*. Aux transitions déterministes sont associées des fonctions de sensibilisation et aux transitions stochastiques sont associées des fonctions stochastiques. Ces fonctions stochastiques sont associées à toute transition modélisant l'occurrence d'une défaillance d'un composant ou sa réparation.

Définition 6 (RdP Prédicats-Transitions Différentiels Stochastiques) *le modèle RdP Prédicats-Transitions Différentiels Stochastiques est défini, par le triplet <P, T, M>. Les places de ce modèle sont décrites par la paire <P, F> où F est l'ensemble des fonctions associées aux places.*

F est défini comme suit :

$$F = \begin{bmatrix} F_i(\dot{X}_i, X_i, t) \\ \vdots \\ F_l(\dot{X}_{l,} X_l, t) \end{bmatrix}, \quad i = 1...l$$

l étant le nombre de places, X_i étant l'ensemble des variables manipulées par les fonctions F_i associées à la place P_i. Nous avons :

$$F_i(\dot{X}_{i,} X_i, t) = \begin{bmatrix} f_{ij}(\dot{X}_{i,} X_i, t) \\ \vdots \\ f_{ik}(\dot{X}_{i,} X_i, t) \end{bmatrix}, \quad j = 1...k$$

Où $f_{ij}...fik$ sont des équations représentant l'évolution des variables continues. Ainsi chaque place du réseau (qui représente un état discret) peut décrire l'évolution de l'ensemble des variables continues X_i (grâce au système d'équations F_i).

Les transitions de ce réseau sont réparties en deux sous-ensembles disjoints : T_s et T_d tel que $T = T_s \cup T_d$ et $T_s \cap T = \emptyset$. T_s est l'ensemble des transitions stochastiques. T_d est l'ensemble des transitions déterministes.

L'ensemble des transitions déterministes est caractérisé par le triplet $<T_d, E, J>$ où E est l'ensemble des fonctions de **sensibilisation** et J l'ensemble des fonctions de **jonction** associées aux transitions. E est défini comme suit :

$$E = \begin{bmatrix} E_1 \\ \vdots \\ E_m \end{bmatrix}$$

m étant le nombre de transitions. Le seuil E_j associé à la transition T_j est défini comme étant la première solution de :

$$E_j(\dot{X}_j, X_j, t) = 0$$

X_j étant l'ensemble des variables continues utilisées par E_j (X_j est l'union des variables associées aux jetons nécessaires au franchissement de T_j). Cette fonction permet de détecter les événements d'état (une variable atteint une valeur prédéterminée), aussi bien que les événements de temps (t atteint une valeur prédéfinie). Les fonctions de sensibilisation sont activées lorsque leurs transitions correspondantes sont sensibilisées. Lors du franchissement d'une transition, les fonctions de jonction associées sont activées. Elles sont définies par :

$$J = \begin{bmatrix} J_1 \\ \vdots \\ J_m \end{bmatrix}$$

La fonction J_j associée à la transition T_j, à la date t^+ (juste après t), lors du franchissement de T_j à la date t calcule les valeurs des variables ainsi que leurs dérivées.

Quant aux fonctions stochastiques, elles sont de deux types : des fonctions stochastiques temporisées ou immédiates. Les fonctions stochastiques temporisées sont décrites par le couple $< I, D >$ où I est l'ensemble des intervalles de tirs associés aux transitions stochastiques temporisées et D est l'ensemble des densités de probabilités associées à ces mêmes transitions. A une transition stochastique temporisée t_k, nous associons la fonction densité de probabilité $D(t_k) = d_k(x)$ définie sur un intervalle de tir statique $I(t_k) = [a, b]$ (intervalle de réels). Il n'y a pas d'effet mémoire, c'est-à-dire que chaque fois que t_k est sensibilisée par un nouveau n-uplet de jetons, un nouvel intervalle $I(t_k)$ est associé au franchissement de t_k par ce n-uplet. Cet intervalle

est effacé si t_k cesse d'être sensibilisée par ce n-uplet, par exemple si l'un des jetons de ce dernier est consommé par un franchissement de transition.

En ce qui concerne les fonctions stochastiques immédiates, elles sont telles qu'une probabilité fixe est associée à chaque transition concernée. Ces transitions sont utiles pour la prise en compte des défaillances à la sollicitation et sont immédiates. En général, un composant qui peut être défaillant à la sollicitation est représenté par une place représentant l'état « attente » du composant possédant deux transitions en sortie. A ces deux transitions sont associées des probabilités p_1 (probabilité de panne à la sollicitation) et p_2 (probabilité de bon fonctionnement après sollicitation) et tel que $p_1 + p_2 = 1$.

2.3. Abstraction temporelle de fonctionnements hybrides

Un outil comme les réseaux de Petri prédicats-transitions différentiels stochastiques est très bien adapté à une description fine de systèmes hybrides soumis à des reconfigurations. Cela est nécessaire pour, par exemple, simuler le plus fidèlement possible des scénarios redoutés dont on connaît déjà les événements constitutifs. Par contre, une telle description parait trop détaillée pour trouver ces scénarios car la complexité de la dynamique continue va, en général, rendre le modèle trop complexe pour pouvoir analyser les relations de causalité existant entre certains événements. C'est pour cela que [Khalfaoui 03] s'était basé sur les réseaux de Petri ordinaires pour son algorithme de construction des scénarios. Mais en procédant ainsi les aspects continus sont complètement perdus lors de cette construction et nous verrons, dans le chapitre suivant, que cela n'est pas sans inconvénient.

Une démarche classique dans le domaine du diagnostic est de rechercher un modèle *qualitatif*, plus abstrait du système pour mieux analyser les causes d'un dysfonctionnement et un certain nombre de chercheurs se sont tournés vers les automates temporisés ou les réseaux de Petri temporels comme outil d'expression. Ainsi [Lunze 99] ou [Hélias 04] ont-ils proposé de construire des automates temporisés en définissant des états qualitatifs pour les variables et en définissant des événements pour chaque changement d'état qualitatif. Une série de simulations permet d'évaluer la durée minimale et maximale séparant deux événements pour un type de trajectoire donné. Une démarche analogue est utilisée par [Loures et Pascal 04] pour construire des réseaux de Petri temporels flous. Ce type de démarche est intéressant pour nous puisque notre objectif principal est de dégager des scénarios pour des systèmes hybrides relativement complexes.

Cependant, dans notre cas, nous avons déjà un modèle discret, le réseau de Petri ordinaire représentant la vue discrète du système. Nous n'avons pas à partir d'un ensemble de simulations ou d'observations à discrétiser les variables pour construire notre modèle qualitatif. Il est déjà présent, c'est le réseau de Petri, mais il n'y a pas d'information temporelle. Nous pouvons alors utiliser le système d'équations algébro-différentielles associé à une place p et le seuil associé à une transition t_i de sortie de p pour obtenir une valeur minimale et une valeur maximale de la durée de séjour des jetons dans p lorsqu'ils sont consommés par t_i. En procédant ainsi, l'abstraction temporelle obtenue va avoir la forme d'un réseau de Petri p-temporel. Par contre les défaillances sont toujours prises en compte sous la forme de transitions stochastiques. Nous

verrons tout ceci plus en détail dans le chapitre 3. Cette section a simplement pour but de préciser nos choix. Le système sera donc représenté par un réseau de Petri prédicats-transitions différentiel stochastique, mais l'algorithme de construction des scénarios utilisera une abstraction temporelle composée d'un réseau de Petri p-temporel associé à des transitions stochastiques. Ce réseau peut être considéré comme une abstraction qualitative du modèle détaillé précédent. Il nous faut maintenant préciser ce que nous entendons par "scénario".

2.4. Notion de scénario redouté

2.4.1. Introduction

Un scénario sous-entend un début, une fin et une histoire qui décrit l'évolution d'un système. Dans le contexte de la sûreté de fonctionnement, un scénario redouté mène à un état catastrophique ou dangereux : c'est l'état final (dit état redouté). L'état initial est un état de bon fonctionnement du système. Le scénario redouté décrit de manière précise (ce qui est nécessaire pour la compréhension) et concise (le juste nécessaire : causalité) comment le système quitte le bon fonctionnement pour évoluer vers un fonctionnement jugé dangereux. C'est en effet une description du système sous la forme de changements d'états et de suites d'événements qui mènent vers l'état redouté. C'est une explication claire des raisons pour lesquelles le système s'est trouvé ou risque de se trouver dans un état redouté donné. Ce n'est donc pas simplement une suite d'événements sans liens de causalité entre eux mais, un ensemble de relations d'ordre qui peuvent être interprétées comme des relations de causalité entre certains événements, ceux qui ne sont pas reliés par une relation d'ordre étant considérés comme des événements non causalement liés.

En résumé, un scénario redouté est une description de l'évolution de certains composants du système global à partir d'un état de bon fonctionnement jusqu'à l'occurrence de l'événement redouté. Ce scénario doit donc ne faire intervenir que les composants ayant un lien de causalité avec l'occurrence de l'événement redouté.

Les combinaisons de défaillances des composants élémentaires d'un système forment la majeure partie des causes possibles pouvant provoquer les scénarios redoutés. Ces combinaisons peuvent être identifiées par un raisonnement en parcourant l'ensemble des combinaisons possibles des défaillances élémentaires. Dans certains cas, ces défaillances élémentaires sont bénignes pour le fonctionnement du système mais, combinées avec des interactions entre certains sous-systèmes (partage de ressource ou partage de variable continue), ces défaillances peuvent être à l'origine des scénarios redoutés.

L'objectif de nos travaux est d'identifier ces scénarios redoutés de façon exhaustive. Pour cela, nous utilisons une approche formelle (la logique Linéaire) pour caractériser et extraire ces scénarios à partir du modèle du système en réseau de Petri. Dans cette partie nous présentons quelques méthodes de représentation et de caractérisation d'un scénario ainsi que les différents aspects : aspect test et aspect vérification, mais nous présentons d'abord l'exemple retenu pour illustrer ces différentes méthodes.

2.4.2. Exemple de constellation de satellites

L'exemple choisi pour illustrer les différentes caractérisations de scénario est un système spatial. Il s'agit d'une constellation composée de deux satellites (S1, S2) et d'une station sol (SS). Nous avons choisi ce système à la fois pour sa simplicité et parce qu'il contient des scénarios qui permettent de voir comment dans une même séquence d'événements avec des ordres d'occurrence différents on peut oui ou non atteindre un état redouté. Le système est défaillant lorsque les deux satellites sont défaillants ou lorsque la station sol est en panne. Dans la figure 2.1 qui représente la modélisation de ce système en réseau de Petri, la défaillance des satellites S1 et S2 est représentée par le franchissement des transitions f_1 et f_2 (marquage des place Ko_1 et Ko_2). La défaillance de la station sol est représentée par le franchissement la transition f_s (marquage de la place Ko_s).

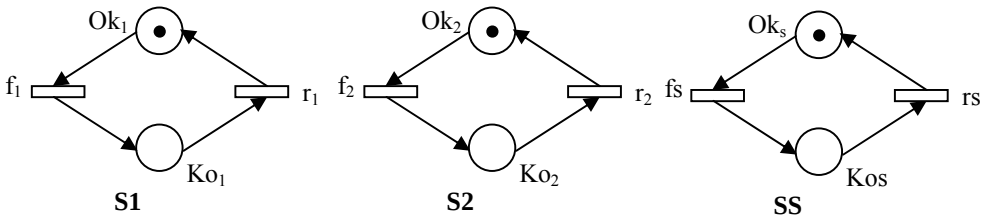


Figure 2.1. Modèle de réseau de Petri de la constellation

Chaque composant du système (un des deux satellites ou la station sol) peut être dans un état nominal (bon fonctionnement "Ok") ou en panne ("Ko"). Tous les composants sont en silence sur défaillance. Quand ils sont dans un état nominal, les satellites et la station sol ont des comportements légèrement différents. Un satellite actif "*i*" envoie sans interruption une télémesure "Out_i" à la station sol ; quand la station sol est dans un état nominal, elle fournit les données prévues "Out_S" si au moins une télémesure est disponible. On suppose que initialement les composants sont dans un état de bon fonctionnement (les place Ok_1, Ok_2 et Ok_s sont marquées) ; ils peuvent alors subir des défaillances et être réparés à tout instant (franchissement des transitions r_1, r_2 et r_s). L'arbre de défaillance de la constellation est donné dans la figure 2.2.

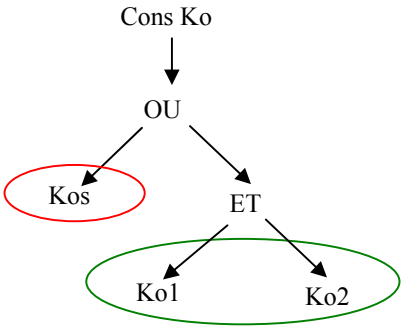


Figure 2.2. Arbre de défaillance de la constellation

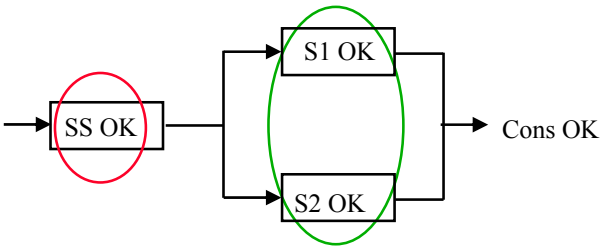


Figure 2.3. Bloc Diagramme de Fiabilité de la constellation

L'analyse de la sûreté basée sur la notion de scénario a été abordée en utilisant différents modèles permettant de représenter des séquences d'événements ainsi que les relations temporelles. Il existe des équivalences entre ces modèles et nous travaillons actuellement sur les notions de minimalité et complétude en se basant sur des modèles de type Logique Temporelle Linéaire (LTL) ou RdP Logique Linéaire.

Dans les paragraphes suivants, nous présentons un scénario sous forme d'une séquence dans un automate, d'un graphe de précédence (un ordre partiel) ou de formule en LTL.

2.4.3. L'approche classique

Dans l'approche classique, on dispose de deux outils de base pour décrire les conditions qui font que le système est dans un état redouté (ou état défaillant). Le bloc Diagramme de fiabilité décrit la structure du système et fait apparaître les configurations de composants nécessaires au bon fonctionnement du système. L'arbre de défaillance décrit les conditions pour que le système soit défaillant. Les deux sont équivalentes. Elles ne prennent pas en compte l'aspect dynamique et la notion de scénario est donc totalement absente.

Prenons le cas de notre exemple. L'arbre de défaillance (figure 2.2) va donner par deux coupes minimales les deux conditions pour que le système soit défaillant : soit la station sol est défaillante, soit les deux satellites sont défaillants. L'information donnée par le bloc diagramme de fiabilité est équivalente (figure 2.3). On caractérise bien un état défaillant, une structure défaillante, mais on ne spécifie pas comment on y arrive. Pour cela il faut clairement se fonder sur une représentation explicite de la dynamique à travers des états et des changements d'état, et l'approche la plus classique est celle des automates finis.

2.4.4. Scénario défini à partir d'un automate fini

Un scénario peut être vu comme une séquence dans un automate qui est une suite complètement ordonnée d'événements (chemin) menant d'un état initial à un état final. Reprenons l'exemple de la constellation de satellites, Si l'on considère le réseau de Petri de la figure 2.1 comme un ensemble d'automates et si l'on construit l'automate produit (graphe des états) (ou, de façon équivalente, le graphe des marquages accessibles du réseau de Petri) on obtient l'automate de la figure 2.4 (chaque arc représente en fait deux arcs, chacun orienté dans un sens). Les états S1, S2 et S4 sont des états de bon fonctionnement, alors que les états S3, S5, S6, S7, S8 sont des états

défaillants. Parmi les séquences menant de l'état initial à l'état redouté on trouve donc f_s, mais aussi $f_s;r_s;f_s$, ou bien f_1; f_s ou encore $f_1;f_2$ et $f_2;f_1$.

Comme le graphe représentant l'automate produit comporte des boucles, le premier problème qui apparaît est celui de l'existence d'une infinité de séquences partant de l'état initial et aboutissant à un état défaillant ou redouté. Il se pose donc la question de savoir si l'on peut, avec un nombre fini de séquences, représenter l'ensemble de ces comportements. Dans ce travail nous donnons par la suite beaucoup d'éléments de réponse, mais la preuve de la complétude de notre approche reste encore à faire.

La seconde question est dans un scénario comme f_1; f_s, on part bien de l'état initial et on s'arrête au premier état redouté rencontré. Pourtant l'événement f_1 n'est pas nécessaire (f_s est suffisant) et la séquence n'est pas minimale. Donc même en l'absence de cycle, le problème de la minimalité se pose.

La troisième question est que si dans un scénario comme fs;rs;fs le fait que r_s suive f_s provient bien d'une relation de causalité (on ne va pas réparer la station sol si elle n'est pas défaillante), il n'en est pas de même pour $f_1;f_s$ ou $f_1;f_2$. La deuxième défaillance n'est pas corrélée à la première et la relation de précédence ne peut pas être interprétée comme une relation de cause à effet.

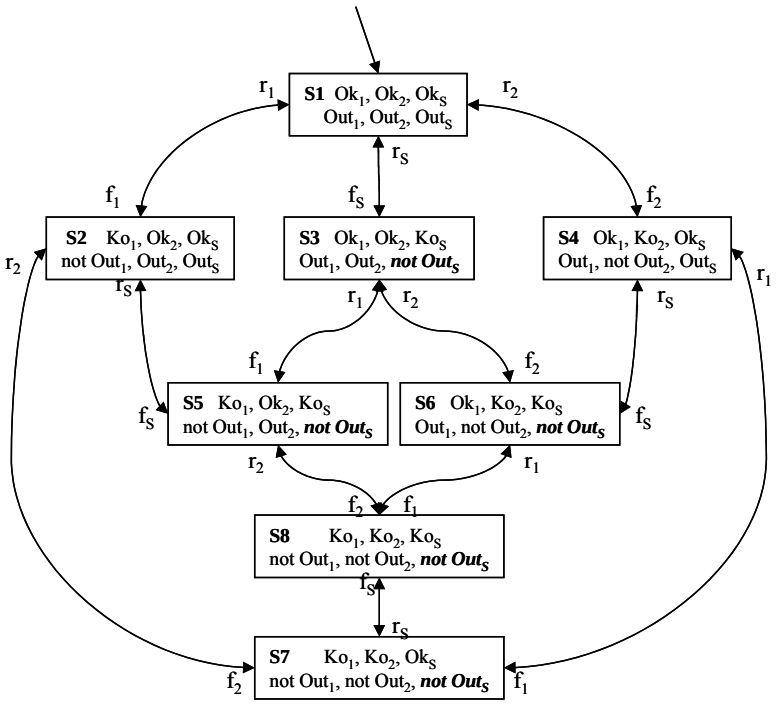


Figure 2.4. Automate du système avec défaillance

Enfin la quatrième question est de savoir s'il n'est pas possible d'avoir une représentation unique pour les deux séquences f_1; f_2 et f_2; f_1. En effet, ce qui importe réellement c'est un comportement où les deux satellites sont devenus défaillants (f_1 et f_2), l'ordre n'ayant pas d'importance. Le problème avec l'utilisation de la notion de séquence pour caractériser un comportement, c'est

qu'elle impose un ordre total entre tous les événements. C'est pourquoi nous allons introduire la notion de graphe de précédence et d'ordre partiel.

2.4.5. Graphe de précédence et ordre partiel

Tout d'abord, il nous semble plus simple de nous placer dans le cas où le système est modélisé par un réseau de Petri qui est capable de prendre en compte explicitement le parallélisme. Définissons d'abord ce que nous appelons un événement dans ce contexte.

Définition 7 (Evénement et ensemble d'événements)*: Soit un réseau de Petri* $P = \{P, T, Pre, Post\}$ *et soit* M_0 *son marquage initial. Un événement est un franchissement d'une transition* $t \in T$. *L'ensemble des événements est noté E. Pre, étant pré-condition et Post, post-condition de transitiont.*

Nous pouvons donc remarquer que l'ensemble des événements E est infini (une même transition peut être franchie une infinité de fois), mais qu'il est construit à partir de l'ensemble fini T des transitions. Dans le reste de notre travail, les événements sont notés e_i^j. Un événement e_i^j correspond au $j^{ième}$ franchissement de la transition t_i.

L'ensemble E doit être complété d'un événement initial I (production du marquage initial) et d'un événement final F (blocage du système dans le marquage correspondant à l'état redouté).

Définition 8 : (Graphe de précédence) *Un graphe de précédence est un graphe orienté acyclique défini par un ensemble fini d'événements E et une relation de précédence qui est un sous ensemble A de E x E. C'est un graphe dont les sommets sont les événements de E et les arcs les éléments de A. Si un arc relie l'événement* e_i *à l'événement* $e_j \left(e_i \rightarrow e_j\right)$ *alors* e_i *précède* e_j *et le couple* $\left(e_i, e_j\right) \in A$.

Etant donné que le graphe est acyclique, et comme la cause précède toujours l'effet, une relation de précédence peut être utilisée pour décrire une relation de causalité entre deux événements. Si l'on part d'un sous-ensemble d'événements permettant de passer de l'état initial à l'état redouté et si l'on est capable d'exprimer les relations de causalité entre ces événements, alors le graphe de précédence sera une représentation du scénario.

Revenons à l'exemple des satellites et considérons les deux séquences $f_1;f_2$ et $f_2;f_1$. L'ensemble E est formé des événements f_1, f_2, I et F. L'événement I est la cause de f_1 et de f_2 (si les satellites ne sont pas mis en fonctionnement ils ne peuvent tomber en panne), et f_1 et f_2 sont tous les deux des causes de F. Nous obtenons le graphe de précédence de la figure 2.5. Nous voyons que de cette façon nous avons un seul graphe de précédence pour représenter globalement les deux séquences.

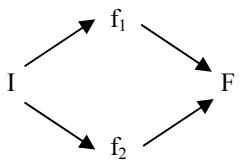


Figure 2.5. Graphe de précédence = ordre partiel

Enfin pour terminer, notons que comme le graphe de précédence est acyclique (il ne peut y avoir de cycle de causalité) sa fermeture transitive définit une relation d'ordre partiel stricte entre les événements.

Nous venons de voir que la notion logique de causalité était fondamentale pour définir les scénarios redoutés. Le strict cadre de la logique classique n'est pas propice pour analyser les causalités entre événements car les propositions logiques y sont des vérités éternelles. Avant de détailler le cadre logique de notre approche qui est fondé sur la logique linéaire de J.Y. Girard, montrons qu'il est également possible de s'appuyer sur la logique temporelle linéaire LTL, au moins pour spécifier les scénarios.

2.4.6. Scénarios et logique temporelle linéaire

Cette logique permet de décrire qu'un événement doit précéder un autre. La notion de graphe de précédence peut donc très facilement être transposée dans ce cadre. Les formules de LTL décrivant un scénario (critique) [Bieber et al 05] sont la conjonction des formules temporelles de base suivantes :

- *F a* : l'événement a se produit au moins une fois dans le présent ou le futur,
- *not F a* : l'événement ne se produira jamais maintenant ou à l'avenir, similaire à G (not a),
- *F (a ∧ F b)* : l'événement b se produit après que l'événement a se soit produit,
- *F (a ∧ not F b)* : l'événement b ne se produira jamais après que l'événement a soit produit.

Dans l'exemple des satellites, la condition de défaillance « perte permanente de la station sol » sera présenté par les formules : $F\ (G\ (not\ Out_s))$.

Le premier scénario critique est $F\ (f_s \wedge not\ F\ r_s)$. Il correspond à la perte permanente de la station sol.

Le deuxième scénario est $F\ (f_1 \wedge not\ F\ r_1) \wedge F\ (f_2 \wedge not\ F\ r_2)$. Il correspond à la perte permanente des deux satellites.

Il est alors possible par *model-checking* de vérifier qu'un scénario est, ou n'est pas cohérent avec un système physique décrit sous la forme d'un automate fini. On peut utiliser par exemple SMV [Kehren et al 04]. Toutefois une recherche directe des relations de causalité à partir de l'automate décrivant le système paraît plus difficile à mettre en œuvre. Le fait qu'un automate fini n'explicite

pas le parallélisme vrai et n'autorise que l'entrelacement des événements rend une telle démarche délicate.

Nous allons maintenant introduire brièvement la logique linéaire et le calcul des séquents pour montrer qu'ils permettent, à partir d'une représentation du système donnée sous la forme d'un réseau de Petri, de déduire des scénarios sous la forme d'un ensemble de relations de précédence.

2.5. Scénario et logique linéaire

2.5.1. Modélisation logique

Dans le cas de la construction d'un scénario, ce que l'on souhaite obtenir c'est un ensemble d'événements muni d'un ordre partiel. On ne souhaite pas énumérer tous les états (les marquages d'un réseau de Petri), mais on souhaite avoir une approche orientée événements. La logique linéaire proposée par J.Y. Girard [Girard 87] prend en compte la notion de *ressource* ce que ne peut pas faire la logique propositionnelle classique. Cela veut dire que les propositions logiques sont considérées, non plus comme des vérités éternelles, mais comme des ressources qui sont consommées et produites pendant les preuves. Une déduction en logique linéaire *consomme* les propositions qu'elle prend pour prémisses, et *produit* les propositions qui forment sa conclusion. Cela signifie que pour réutiliser une prémisse qui vient d'être consommée, soit il aurait fallu initialement poser deux fois cette prémisse, soit il faut préalablement la produire à nouveau, par le biais d'une autre déduction.

Plusieurs connecteurs ont été introduits en Logique Linéaire (LL) mais pour traduire un réseau de Petri en Logique Linéaire, le fragment MILL (Multiplicative Intuitionnist Linear Logic) contient les connecteurs nécessaires.

2.5.1.1 Introduction à la logique linéaire : fragment MILL

Ce fragment comprend le connecteur multiplicatif « *Fois* » et le connecteur « *implication linéaire* ». Il n'y a pas de négation et le méta-connecteur «**,** » est commutatif. Le lecteur intéressé pourra trouver une présentation détaillée des autres connecteurs dans [Girault 97].

Le connecteur « *Fois* », noté $\otimes$, est la conjonction multiplicative. Il correspond au connecteur « *et* » de la logique classique auquel on a enlevé la propriété d'idempotence. Ce connecteur exprime l'accumulation de ressources : ainsi la proposition $A \otimes A$ représente la présence de deux exemplaires de A. Cette proposition n'est pas équivalente à A.

Le connecteur « *implication linéaire* », notée $\multimap$, exprime la *causalité* entre la production et la consommation de ressources. Par exemple, $A \multimap B$ traduit le fait qu'en consommant la proposition A, la proposition B est produite. C'est donc le résultat d'un changement d'état qui est ainsi modélisé.

Considérons deux exemples de déduction utilisant ces deux connecteurs. A partir de la proposition $A \otimes A$ et de $A \multimap B$ il est possible de déduire $A \otimes B$ mais pas B : le second

exemplaire de A ne peut être « oublié ». Si on part de A et de $A \otimes A \multimap B$, il n'est pas possible de déduire B car un seul exemplaire de A n'est pas suffisant pour la déduction. Dans les deux cas, l'usage des connecteurs classiques (« et » et « *implique* ») aurait permis ces déductions.

Un séquent en logique Linéaire

Un *séquent* est une formule de la forme : $\Gamma, F \vdash \Delta, G$ où le symbole $\vdash$, dit le tourniquet, est un méta-symbole qui sépare la partie gauche (ici, Γ, F) de la partie droite (ici, Δ, G). Ces deux parties sont constituées de suites finies de formules. La virgule est aussi un méta-symbole dont le sens est fonction de sa position par rapport au tourniquet : la conjonction dans la partie gauche et la disjonction dans la partie droite. Le séquent peut se lire de la façon suivante : la conjonction « Γ *et* F » permet de déduire la disjonction « Δ *ou* G ». Dans le cadre de fragment MILL, le membre droit sera toujours réduit à une seule formule (qui peut contenir plusieurs atomes).

2.5.1.2 Traduction des RdP en logique linéaire

L'une des bases de la logique linéaire est sa capacité à raisonner sur les ressources. Les réseaux de Petri sont également un outil qui manipule des ressources. Du fait de cette similitude, plusieurs travaux ont traité du lien entre ces deux formalismes. Nous ne présenterons, par la suite, que ceux qui traitent de la logique linéaire comme un outil permettant d'interpréter les modèles RdP et d'en extraire les relations d'ordre en prenant en compte la notion de marquage. Cela permet, une meilleure caractérisation des relations d'ordre en tenant compte de l'influence du marquage sur les relations d'ordre structurelles dans le réseau de Petri.

Afin de permettre une meilleure caractérisation des relations d'ordre dans un réseau de Petri, des travaux [Pradin et al 99], [Girault 97] ont été menés et ont abouti à une approche, dite avec marquage. Cette approche permet de prendre en compte le parallélisme dynamique d'un réseau de Petri. C'est celle que nous avons retenue pour notre travail car une prise en compte exacte du parallélisme est essentielle dans les scénarios.

Dans le cadre de cette approche, les transitions d'un réseau de Petri sont représentées par une proposition implicative. Nous représentons ainsi des instances de franchissement de transitions et non plus des transitions, et les propositions implicatives pourront être consommées au cours de la preuve, ce qui indiquera que la transition est effectivement franchie. Si la proposition implicative représentant la transition est consommée deux fois au cours de la preuve, cela signifie que cette transition a été franchie deux fois. En ce qui concerne le marquage, il est explicitement pris en compte.

Dans cette approche, une formule logique est associée à chaque marquage et à chaque transition. Un marquage correspond à la présence simultanée de jetons dans un ensemble de places. A chaque place correspond une proposition atomique. Un marquage est alors décrit par une formule conjonctive (connecteur $\otimes$) de propositions atomiques. Ainsi, un marquage constitué d'un jeton dans la place A et d'un jeton dans la place B se traduira par la formule $A \otimes B$. Alors que le marquage est constitué d'un jeton dans la place A et aucun jeton dans la place B se traduira par la formule A. Par contre, un marquage constitué de deux jetons dans la place A et d'un jeton dans la place B se traduira par la formule $A \otimes A \otimes B$.

Une transition exprime une *relation de causalité* (dite aussi une *relation de cause à effet*) entre deux formules de marquage. En effet, le marquage final est causalement lié au marquage initial car, quand ce dernier est consommé, ceci produit le marquage final. En conséquence une transition est traduite par une formule implicative (connecteur $\multimap$). Le côté gauche de la formule établit le marquage minimal pour franchir la transition, tandis que le côté droit représente le marquage atteint après le franchissement de cette transition à partir du marquage minimal. Pour un réseau de Petri donné, cette traduction peut être formalisée de la façon suivante :

- un atome propositionnel P est associé à toute place p du réseau,
- un monôme en $\otimes$ (*Fois* : la conjonction multiplicative) est associé à tout marquage ainsi qu'à toute pré-condition (*Pre*) ou post-condition (*Post*) de transition,
- une formule implicative est définie pour chaque transition t du réseau de Petri :

$$t : \bigotimes_{i \in \Pr e(pi,t)} p_i \multimap \bigotimes_{o \in Post(po,t)} p_o$$

Chaque franchissement de transition est représenté par une instance de la forme implicative associée à la transition franchie.

L'accessibilité entre deux marquages M_0 et M_f est représentée par un séquent. La partie gauche de ce séquent contient la liste de tous les franchissements de transition permettant d'atteindre le marquage M_f à partir du marquage M_0. Cette partie du séquent contient également la formule représentant le marquage initial. Quant à la partie droite du séquent, elle contient la formule représentant le marquage final. Le séquent exprimant l'accessibilité entre les marquages M_0 et M_f s'écrit sous la forme : $M_0, t_1, ..., t_n \vdash M_f$

Le séquent spécifie quelles sont les transitions franchies (t_i représente la formule implicative correspondant à la transition t_i et doit être répétée autant de fois qu'elle est franchie pour atteindre M_f).

2.5.1.3 Equivalence entre accessibilité et preuve

[Girault 97] a montré qu'il y a équivalence entre la prouvabilité de certains séquents du fragment MILL de la logique linéaire et l'accessibilité dans un réseau de Petri. Afin d'illustrer cette équivalence entre accessibilité et prouvabilité, considérons un réseau de Petri muni d'un marquage initial M_0 et d'un marquage final M_f. L'équation fondamentale étant $C = Post - \Pr e$.

Soit $\bar{s}$ une liste non ordonnée de transitions ($\bar{s}$ est le vecteur caractéristique, solution de l'équation fondamentale $M_f = M_0 + C.\bar{s}$. Il y a équivalence entre le fait de prouver le séquent $M_0, \bar{s} \vdash M_f$ et celui de trouver une séquence σ de franchissements de transitions menant de M_0 à M_f dans le réseau de Petri telle que les transitions de σ soient celles de s avec le même nombre

de franchissement de transitions (on franchit autant de fois la transition t_i qu'elle se trouve dans la liste $\bar{s} : \bar{\sigma} = \bar{s}$).

Puisqu'il y a équivalence entre l'accessibilité dans un réseau de Petri et la prouvabilité de certains séquents en logique linéaire, nous allons nous intéresser à comment conduire la preuve du séquent exprimant cette accessibilité à travers la construction d'un arbre de preuve canonique.

2.5.2. Preuve d'un séquent caractéristique

Prouver un séquent $(\Gamma, F \vdash \Delta, G)$ consiste à montrer qu'il est syntaxiquement correct : entièrement construit à partir d'un ensemble de règles introduisant les atomes (propositions) et les connecteurs. Les règles représentées ci-dessous sont celles du fragment multiplicatif MILL. Ces règles sont réparties en 3 groupes : le groupe identité, le groupe structurel et le groupe logique. Le groupe identité et le groupe structurel expriment des propriétés intrinsèques à la logique tandis que le groupe logique définit une règle d'introduction à gauche et une règle d'introduction à droite pour chacun des connecteurs spécifiques à la logique considérée (à savoir ici $\otimes$ et $\multimap$). Le meta-symbole «, » et le connecteur $\otimes$ sont commutatifs. Nous ne présentons ici que les règles que nous utilisons.

$$\frac{\Gamma \vdash F \quad \Delta, G \vdash H}{\Gamma, \Delta, F \multimap G \vdash H} \multimap L \qquad \frac{}{A \vdash A}\ identité$$

$$\frac{\Gamma, F, G \vdash H}{\Gamma, F \otimes G \vdash H} \otimes L \qquad \frac{\Gamma \vdash F \quad \Delta \vdash G}{\Gamma, \Delta \vdash F \otimes G} \otimes R$$

Figure 2.6. Règles du calcul des séquents du fragment MILL

A est un atome, *F*, *G* et *H* sont des formules, pas nécessairement atomiques. Γ et Δ sont des blocs de formules séparées par des virgules. Pour chaque règle, le séquent à prouver est écrit en dessous de la barre tandis que le (ou les) séquent(s) utilisé(s) pour cette preuve est (sont) écrit(s) au dessus.

L'attribut indique que l'on applique la règle à gauche (indice *L* comme *Left*) ou à droite (indice *R* comme *Right*) du tourniquet du séquent concerné.

Une preuve est conduite de bas en haut en posant comme racine de l'arbre de preuve le séquent à démontrer. On applique ensuite une à une les règles pour éliminer successivement les connecteurs du séquent. Chaque noeud de l'arborescence est alors un séquent prémisse plus simple que celui dont il est conclusion. Ce séquent est prouvé si chaque feuille de l'arbre se termine par un séquent axiome (dans le sous-ensemble MILL, le seul axiome est le séquent identité). On dit qu'un séquent est prouvable si et seulement si il existe une preuve dont il est la racine.

2.5.3. Formalisation du scénario en RdP et en Logique Linéaire

Nous avons vu au paragraphe 2.4 que la notion de scénario la plus intéressante était celle qui reposait sur un graphe de précédence exprimant, en fait, les causalités entre les événements. Si nous nous appuyons sur l'expression d'un séquent $M_i, \bar{s} \vdash M_f$ pour caractériser un scénario entre un marquage initial M_i et un marquage final M_f, nous ne disposons, a priori, que de la liste $\bar{s}$ des franchissements de transition. La question qui se pose immédiatement est donc de savoir si, à partir du séquent, la logique linéaire nous permet d'obtenir l'ordre partiel entre les franchissements de transition de $\bar{s}$. Pour cela nous allons introduire l'étiquetage de l'arbre de preuve. Mais tout d'abord, revenons sur notre exemple de constellation de satellites afin de donner un exemple d'arbre de preuve de séquent avant étiquetage.

2.5.3.1 Arbre de preuve canonique de l'exemple

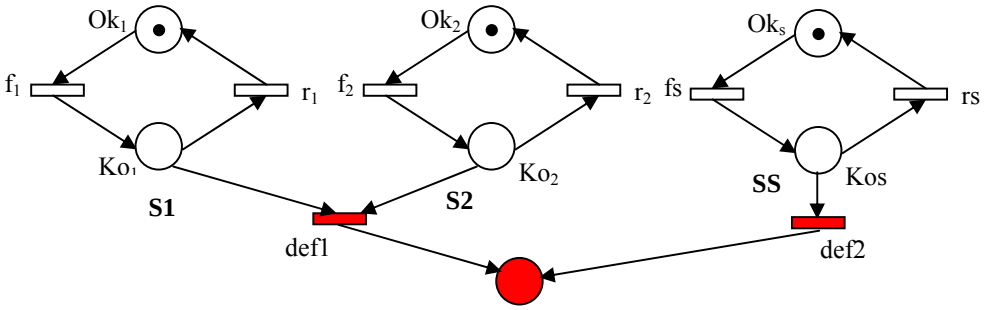


Figure 2.7. Modèle réseau de Petri de la constellation avec défaillance

$$\dfrac{\dfrac{}{Ok_s \vdash Ok_s}\,id \qquad \dfrac{\dfrac{}{Ko_s \vdash Ko_s}\,id \qquad \dfrac{}{D \vdash D}\,id}{Ko_s, def_2^1 \vdash D} \multimap L}{Ok_s, f_s, def_2 \vdash D} \multimap L$$

Figure 2.8. Arbre de preuve du scénario sc1

Pour que le scénario soit plus explicite, nous complétons d'abord le modèle de la figure 2.1 en explicitant l'état redouté. Ce nouveau réseau de Petri est représenté sur la figure 2.7. Les transitions def_1 et def_2 correspondent aux deux coupes minimales de l'arbre de défaillance du système. Nous allons prendre comme exemple le scénario sc_1 associé à la coupe def_2 c'est-à-dire à la défaillance de la station sol. Ce scénario est associé au séquent de logique linéaire $Ok_s, f_s, def_2 \vdash D$ puisqu'il correspond aux franchissements des transitions f_s et def_2.Il faut donc appliquer deux fois la règle $\multimap L$ pour éliminer les implications linéaires des deux formules $Ok_s \multimap Ko_s$ et $Ko_s \multimap D$ associées à ces deux franchissements. On obtient alors directement trois séquents identités.

2.5.3.2 Etiquetage de l'arbre de preuve canonique

Dans la preuve ci-dessus, nous devons commencer par éliminer l'implication linéaire associée au franchissement de f_S avant d'éliminer celle associée à celui de def_2. En effet, pour pouvoir produire un séquent identité, il faut pouvoir associer la pré-condition associée au franchissement avec un élément du marquage courant. Donc l'ordre partiel entre les franchissements est implicitement présent dans l'arbre de preuve. Pour l'expliciter Nicolas Rivière [Rivière 03] a introduit le processus d'étiquetage suivant.

Annotation des règles : A chaque fois que la règle $—\circ L$ (pour éliminer une instance de formule décrivant un franchissement de transition) est appliquée, une étiquette avec le nom de la transition correspondant à la formule implicative éliminée est associé. Lorsqu'une transition est franchie plusieurs fois, nous mettons en exposant un indice égal au nombre de franchissements effectués. Ainsi l'étiquette t_i^j signifie que c'est la j^{ème} élimination d'une formule associée à la transition t_i. Cela permet de différencier les tirs d'une même transition.

Annotations des atomes : Chaque atome d'une étape courante est étiqueté différemment selon qu'il est à gauche ou à droite du tourniquet du séquent à prouver. Lorsque cet atome est à gauche du tourniquet, cette étiquette est celle de la règle qui l'a produit. Quand il est à droite, il prend l'étiquette de la règle qui l'a consommé. Pour que tous les atomes aient une étiquette, les atomes associés au marquage initial du scénario sont étiquetés par des événements I^i (représentant des franchissements hypothétiques ayant produit les jetons correspondants) et les atomes du marquage final par des événements F^j (représentant des franchissements hypothétiques ayant consommé les jetons correspondants). Ces étiquettes rajoutées ont l'avantage de permettre la composition des scénarios [Rivière 03].

L'arbre de preuve annoté du scénario correspondant à la défaillance de la station sol est donné en figure 2.9. L'arbre de preuve annoté de scénario correspondant à la défaillance des deux satellites peut être obtenu d'une façon similaire.

$$\dfrac{\dfrac{Ok_s\left(I^1\right)\vert- Ok_s\left(f_s^{\,1}\right)^{id} \quad \dfrac{Ko_s\left(f_s^{\,1}\right)\vert- Ko_s\left(def_2^{\,1}\right)^{id} \quad D\left(def_2^{\,1}\right)\vert- D\left(F^1\right)^{id}}{Ko_s\left(f_s^{\,1}\right), def_2^{\,1}\vert- D\left(F^1\right)}\ —\circ L(def_2^{\,1})}{Ok_s\left(I^1\right), f_s, def_2\vert- D\left(F^1\right)}\ —\circ L(f_s^{\,1})$$

Figure 2.9. Arbre de preuve étiqueté du scénario sc1

$$\dfrac{\dfrac{Ok_1\left(I^1\right)\vert- Ok_1(f_1) \quad \dfrac{\dfrac{Ko_1(f_1)\vert- Ko_1(def_1) \quad Ko_2(f_2)\vert- Ko_2(def_1)}{Ko_1(f_1), Ko_2(f_2)\vert- Ko_1(def_1)\otimes Ko_2(def_1)} \quad D(def_1)\vert- D\left(F^1\right)}{Ko_1(f_1), Ko_2(f_2), def_1\vert- D\left(F^1\right)} \quad Ok_2\left(I^2\right)\vert- Ok_2(f_2)}{Ko_1(f_1), Ok_2\left(I^2\right), f_2, def_1\vert- D\left(F^1\right)}}{Ok_1\left(I^1\right)\otimes Ok_2\left(I^2\right), f_1, f_2, def_1\vert- D\left(F^1\right)}$$

Figure 2.10. Arbre de preuve étiqueté du scénario sc2

2.5.3.3 Obtention du graphe de précédence

Les feuilles de l'ardre de preuve sont des séquents identités qui expriment le fait que les jetons consommés avaient bien été produits au préalable. La présence des étiquettes explicite les événements de production et de consommation. Donc chaque feuille de l'arbre de preuve explicite une relation de précédence associée à un atome logique (un jeton dans une place du réseau de Petri). Le graphe de précédence caractérisant le scénario peut donc être construit directement à partir de l'arbre de preuve étiqueté.

Les graphes de précédence des scénarios sc_1 et sc_2 sont donnés par les figures 2.11 et 2.12.

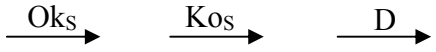


Figure 2.11. Graphe de précédence étiqueté du scénario sc1

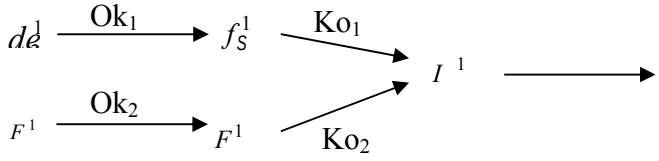


Figure 2.12. Graphe de précédence étiqueté du scénario sc2

2.6. Définition et propriétés d'un scénario critique

Maintenant que nous avons introduit informellement ce que nous appelons un scénario et que nous avons donné les fondements de la logique linéaire nécessaires à notre approche, nous allons définir formellement un scénario et certaines de ses propriétés.

2.6.1. Définition d'un scénario

Définition 9 : *Soit un réseau de Petri, un scénario sc pour ce réseau est défini par un marquage initial M_i, un marquage final M_f, une liste de franchissements de transitions l et un ordre partiel strict $\prec_{sc}$ défini sur l'ensemble $l \cup I \cup F$ où I est l'ensemble des événements ayant produit les jetons de M_i et F l'ensemble de ceux ayant consommé les jetons de M_f.*

Les figures 2.11 et 2.12 donnent deux exemples de scénarios définis sur le réseau de Petri de la figure 2.7.

2.6.2. Conditions suffisantes

Définition 10 : *Soit un réseau de Petri et soit E l'ensemble des événements correspondant à des franchissements de transitions de ce réseau.*

L'ensemble d'événements $l \subset E$ *est suffisant pour passer du marquage initial* M_i *au marquage final* M_f *si le séquent* $M_i, l \vdash M_f$ *est prouvable.*

Si on prend l'exemple de la figure 2.7, l'ensemble d'événements $\left\{f_1^1, f_2^1, def_1^1\right\}$ est suffisant entre $M_i = Ok_1 \otimes Ok_2$ et $M_f = D$.

Ce même ensemble d'événements n'est pas suffisant entre $M_i = Ok_1 \otimes Ok_2 \otimes Ok_s$ et $M_f = D$. En effet, le jeton présent dans la place Ok_S doit être consommé par un événement de l'ensemble *l*. Sinon il doit être encore présent au marquage final et l'on doit avoir alors $M_f = D \otimes Ok_s$.

Il est important de souligner que l'équation fondamentale ($C = Post - \Pr e$) :

$$C\,\overline{l} = M_f - M_i$$

est une condition *nécessaire* mais non suffisante pour que l'ensemble d'événements l soit un scénario entre M_i et M_f. Ceci est dû en particulier aux boucles élémentaires qui font qu'il peut être impossible de trouver une séquence de franchissements de transitions effectivement franchissables entre M_i et M_f même si l'équation fondamentale est effectivement vérifiée.

Définition 11 (scénario strictement suffisant) : *Soit un réseau de Petri et un ensemble d'événements* $l' \subset (\mathrm{E} \cup \mathrm{I} \cup \mathrm{F})$ *(défini comme ci-dessus) et soit* $l = l' \cap E$ *la restriction de* l' *à* E. *Le scénario* $sc = (l', \prec_{sc})$ *est strictement suffisant pour passer de* M_i *à* M_f *si le séquent* $M_i, l \vdash M_f$ *est prouvable et s'il existe un ordre partiel* $\prec_j$ *résultant d'un arbre de preuve du séquent tel que* $\prec_j = \prec_{sc}$.

Le scénario sc_1 (figure 2.11) est strictement suffisant car les relations de précédence sont obtenues par l'étiquetage d'un arbre de preuve du séquent (figure 2.9). Le scénario sc_2 (figure 2.12) est aussi strictement suffisant (arbre de preuve étiqueté de la figure 2.10).

2.6.3. Scénario minimal

Définition 12 (ensemble minimal) : *Soit* $l_k \subset E$ *un ensemble d'événements suffisant pour passer de* M_i *à* M_f. *L'ensemble d'événements* l_k *est minimal s'il n'existe aucun ensemble* $l_i \subset E$ *qui soit un sous-ensemble de* l_k *et qui soit suffisant.*

Par exemple, considérons le réseau de Petri de la figure 2.7 et les marquages $M_i = Ok_1 \otimes Ok_2$ et $M_f = D$, l'ensemble d'événements $l_1 = \left\{f_1^1, r_1^1, f_2^1, f_1^2, def_1^1\right\}$ est suffisant. Mais il n'est pas minimal car il existe aussi $l_2 = \left\{f_1^1, f_2^1, def_1^1\right\}$ qui est inclus dans le premier et qui est également suffisant. Cette notion permet d'éliminer les boucles.

Définition 13 (scénario minimal) : *Soit un ensemble d'événements* $I \subset (E \cup I \cup F)$ *et soit* $I' = I \cap E$ *la restriction de* I *à* E. *Le scénario* $sc = (I, \prec_{sc})$ *est minimal pour passer de* M_0 *à* M_f *s'il est strictement suffisant et si l'ensemble d'événements* I' *est minimal.*

Si maintenant nous considérons la séquence (ordre total entre les événements) $s_2 = f_1^1; f_2^1; def_1^1$, elle correspond à la liste d'événements $l_2 = \{f_1^1, f_2^1, def_1^1\}$ et est donc minimale. Par contre elle ne correspond pas à un scénario minimal. En effet, l'arbre de preuve du séquent $Ok_1 \otimes Ok_2, f_1^1, f_2^1, def_1^1 | - D$ ne produit qu'un seul ordre partiel, celui de la figure 2.12 et cet ordre partiel est moins restrictif que celui de la séquence complètement ordonnée s_2. La notion de scénario minimal permet de sélectionner la relation d'ordre la moins contrainte possible entre les événements. En effet un tel scénario englobera en une seule spécification toutes les séquences admissibles. Il sera donc un meilleur représentant des comportements redoutés.

Maintenant que nous avons donné ces définitions théoriques, nous allons présenter notre démarche qui permet la construction des scénarios redoutés.

2.7. Méthode d'extraction des scénarios redoutés

Dans cette partie nous présentons la méthode proposée par [Khalfaoui 03] permettant de trouver les scénarios redoutés dans un système mécatronique, qui est un cas d'un système piloté par calculateurs, et de les caractériser par des relations de causalité entre événements. Ces relations induisent l'ordre partiel entre les événements définissant le scénario. Cette méthode de recherche de scénarios redoutés est basée sur la logique linéaire et permet d'extraire les scénarios directement du modèle réseau de Petri retenu comme nous venons de le présenter précédemment. La recherche des causalités est basée uniquement sur le modèle réseau de Petri du système et non sur le modèle continu (les équations algébro-différentielles)

2.7.1. Principe

Soit un modèle en réseau de Petri PTDS d'un système mécatronique. On cherche les scénarios menant à un état redouté caractérisés par un marquage partiel c'est-à-dire, un état tel qu'une place particulière ou un sous-ensemble de places du réseau contenant un jeton. Cette place, notée E.P.R (pour Etat Partiel Redouté) peut correspondre au déclenchement d'une alarme suite au franchissement d'un seuil de sécurité par une variable continue du système. La figure 2.13 montre différents scénarios menant à un état partiel redouté. Les étoiles représentent des états partiels du système et les flèches représentent des enchaînements de relations de cause à effet menant de l'état partiel, origine de la flèche, à l'état partiel destination. Nous expliquons ci-dessous comment s'applique cette méthode de recherche de scénarios.

Partant de l'état partiel redouté E.P.R, on commence *un raisonnement arrière* permettant de construire les prédécesseurs immédiats de cet état partiel. Dans l'exemple de la figure 2.15, on trouve deux états partiels : l'état partiel normal E.P.N.1 et l'état partiel dégradé E.P.D. L'état E.P.N.1 étant considéré comme état de fonctionnement normal, on ne cherche plus ses prédécesseurs car cela n'apportera pas plus d'informations du point de vue sûreté de

fonctionnement. Quant à l'état partiel dégradé E.P.D, on poursuit le raisonnement arrière et la construction de ces prédécesseurs immédiats car il s'agit d'un état dégradé et non d'un état normal. Ceci donne les états partiels normaux E.P.N.2 et E.P.N.3. Etant donné que ce sont des états de fonctionnement normal, construire leurs prédécesseurs n'apportera aucune information pertinente du point de vue sûreté de fonctionnement. Le raisonnement arrière s'achève ici.

A ce stade, on a construit trois scénarios menant à l'état partiel redouté E.P.R : celui partant de l'état partiel E.P.N.1, celui reliant les états partiels E.P.N.2 et E.P.R en passant par l'état partiel dégradé E.P.D et enfin celui passant par ce même état mais reliant E.P.N.3 et l'état partiel redouté. Pour résumer, on a donc trois scénarios menant vers l'état partiel redouté en question.

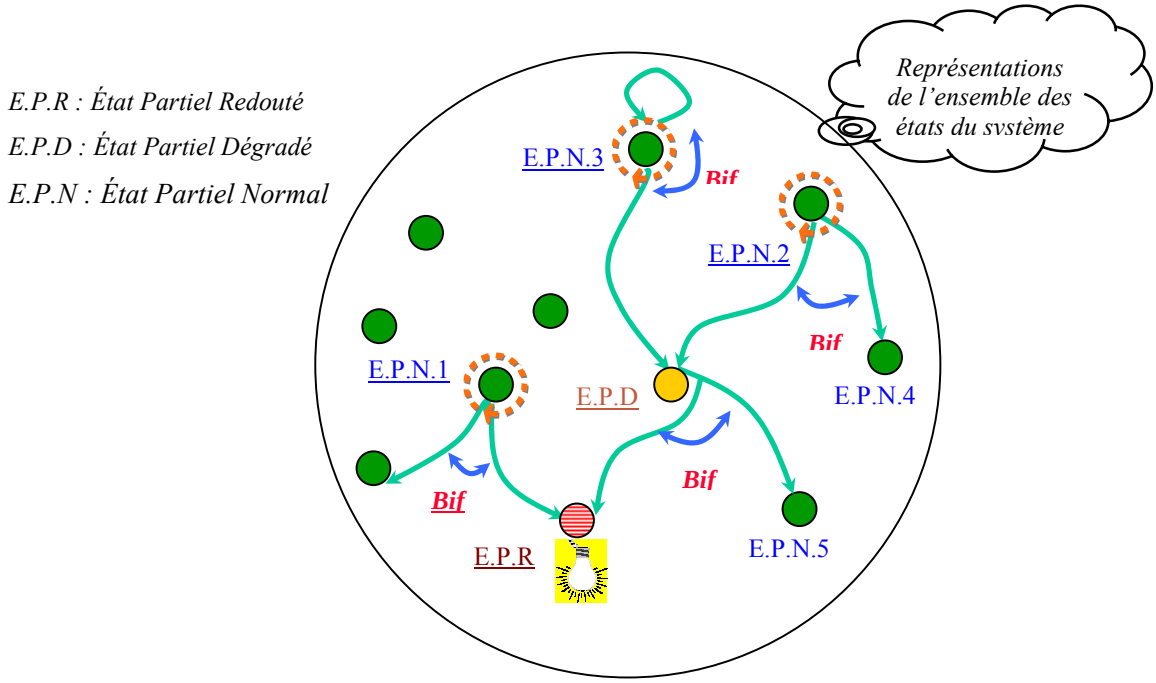


Figure 2.13. Principe de la méthode de recherche de scénarios

A ce stade, on connaît les événements qui ont accompagné l'évolution du système vers l'état redouté, toutefois, on n'a obtenu aucune information sur les causes à l'origine de ces scénarios redoutés. Dans le cas des systèmes mécatroniques, l'évolution du système vers un état redouté s'explique généralement par l'échec d'une stratégie de reconfiguration. Cette dernière a pour but d'empêcher le système d'évoluer vers l'état redouté en assurant un fonctionnement dégradé ou en compensant complètement ce dysfonctionnement. Le système se trouverait ainsi dans un nouvel état de bon fonctionnement. La reconfiguration consiste donc à détourner l'évolution du système vers un nouvel état non redouté, c'est ce qu'on a appelé une *bifurcation*. Son échec rend l'évolution vers l'état redouté inévitable si aucune autre reconfiguration n'a été prévue.

Une façon de connaître les causes de l'occurrence du scénario redouté est de suivre l'évolution du système à partir des états partiels normaux obtenus à la suite du raisonnement arrière, à savoir ici E.P.N.1, E.P.N.2 et E.P.N.3. Ces états sont appelés *états partiels conditionneurs*. Ces états représentent les derniers états de bon fonctionnement du système pendant son évolution jusqu'à

l'état redouté. Le passage par ces états conditionne donc l'évolution vers l'état redouté. Afin de suivre cette évolution, on commence un raisonnement avant à partir de ces états.

Un raisonnement avant consiste à chercher les conséquences logiques possibles à partir d'un état partiel donné. A partir de chacun de ces états partiels conditionneurs (par exemple ici E.P.N.2), on construit ses successeurs immédiats (E.P.D ou E.P.N.4). On arrête le raisonnement avant pour E.P.N.4 car il s'agit d'un état de bon fonctionnement pour le système, poursuivre le raisonnement n'aura plus de sens du point de vue SdF. Quant à l'état partiel dégradé E.P.D, la construction des successeurs se poursuit, ce qui donne soit l'état partiel redouté (E.P.R) soit un état partiel de bon fonctionnement (E.P.N.5). Le raisonnement avant s'achève dans les deux cas.

Récapitulons. A l'issue de ce raisonnement avant à partir de l'état partiel normal E.P.N.2, on a identifié deux scénarios en plus du scénario redouté (le passage de E.P.N.2 vers E.P.R à travers E.P.D) : le premier est celui partant de E.P.N.2 et aboutissant à E.P.N.4 et le deuxième relie E.P.N.2 à E.P.N.5 en passant par E.P.D. Ces deux scénarios peuvent être considérés comme des tentatives de reconfiguration réussies du système lui évitant d'évoluer jusqu'à l'état redouté. Si le système ne peut évoluer selon ces deux scénarios, il ne peut qu'évoluer selon le scénario redouté. On considère que l'impossibilité du système d'évoluer selon ces scénarios est une cause possible du scénario redouté. L'étude approfondie des raisons de cette impossibilité, impliquant la dynamique des variables continues, pourrait donner des renseignements à propos des causes possibles de l'évolution dangereuse. Les causes d'occurrence du scénario redouté s'enrichissent petit à petit en étudiant les évolutions en conflit avec celle du scénario en question.

2.7.2. Les différentes étapes

Cette méthode permet de déterminer, de manière systématique et formelle, comment marquer et démarquer un ensemble de places correspondant à l'état cible. Elle est composée de 4 étapes. Les deux premières permettent de définir l'état cible ainsi que les états conditionneurs. Les deux dernières ont pour rôle de déterminer soit les composants ou sous-modules impliqués dans les scénarios redoutés, pour la troisième étape, soit l'évolution du système à partir de l'état de ces composants trouvé à l'étape précédente, en ce qui concerne la dernière étape. Il s'agit respectivement de l'étape de raisonnement arrière et de l'étape de raisonnement avant. Nous décrivons ci-dessous ces différentes étapes.

2.7.2.1 Détermination des états nominaux

La première étape consiste à déterminer les places dont le marquage représente un état partiel de fonctionnement normal. Ces places nominales seront utilisées comme critère d'arrêt du raisonnement. Cette étape peut être réalisée de deux manières : soit en utilisant une connaissance a priori des états de bon fonctionnement du système, soit en effectuant une simulation de Monte Carlo du modèle sur une courte fenêtre temporelle pour déterminer la probabilité de marquage des places du réseau. Celles qui auront une probabilité de marquage non négligeable seront assimilées à des places normales.

2.7.2.2 Détermination des états cibles

La deuxième étape détermine l'état cible à étudier. Cet état cible peut être soit un état partiel redouté, soit un autre état partiel ayant un lien de causalité direct ou indirect avec cet état redouté (par exemple une place qui représente la disponibilité d'une ressource pour assurer un fonctionnement dégradé évitant l'occurrence de l'événement redouté). La détermination des états partiels redoutés (déclenchement d'une alarme suite à un dépassement d'un seuil de sécurité par exemple) peut se faire grâce à une Analyse Préliminaire des Risques.

2.7.2.3 Raisonnement arrière

Définition 14 (RdP inversé) : Soit *R* un réseau de Petri. On appelle réseau de Petri inversé le réseau de Petri obtenu une fois que l'on a inversé tous les arcs du réseau *R*. Notons *R -1* ce nouveau réseau de Petri.

L'étape de raisonnement arrière est réalisée sur le RdP inversé. Elle consiste à générer l'ensemble des chemins qui mènent vers l'état cible et identifie les différents sous-modules impliqués dans le scénario. Dans le RdP inversé, on prend comme marquage initial le seul état cible et l'on cherche de façon exhaustive tous les scénarios permettant de consommer le marquage initial et aboutissant à un marquage final uniquement formé de places associées au fonctionnement normal. Au cours de cette étape, on est en général amené à enrichir le marquage initial (ajouter des jetons dans certaines places comme cela sera expliqué dans le paragraphe 1.6.3). Cela se fera chaque fois que, pour consommer un jeton dans une place non associée à un fonctionnement normal il faut franchir une transition non sensibilisée par un marquage accessible à partir du marquage initial non enrichi.

Les jetons ajoutés lors du processus d'enrichissement du marquage correspondent à des états partiels qui sont des conséquences logiques des scénarios redoutés. Ils seront donc nécessairement observés lors de l'évolution du système vers l'état redouté. En inversant les scénarios obtenus lors de cette étape, nous aurons les actions menant d'un état normal à l'état partiel redouté avec l'ordre partiel que l'on doit respecter.

2.7.2.4 Raisonnement avant

La dernière étape de la méthode consiste à construire un raisonnement à partir du modèle RdP initial en partant de chaque état conditionneur déterminé à l'étape précédente. Cette étape de raisonnement avant a pour objectif de localiser les bifurcations entre le comportement redouté et le fonctionnement normal du système ainsi que les conditions (de marquage de certaines places du réseau) impliquées dans ces bifurcations.

2.7.3. Enrichissement de marquage

L'enrichissement de marquage permet de compléter l'information sur le scénario redouté en ajoutant des hypothèses sur les états des composants (des jetons dans des places non marquées) ayant un lien avec ce scénario. L'enrichissement de marquage aura lieu au cours du raisonnement avant (ou arrière) pendant la construction des successeurs (ou des prédécesseurs). Parfois, pour construire un successeur immédiat d'un état partiel donné (ce qui revient à franchir une transition

donnée dans le réseau de Petri), il est indispensable d'ajouter des jetons dans des places non marquées pour franchir cette transition. L'ajout de ces jetons peut parfois causer des problèmes de cohérence. En effet, partant du principe que l'enrichissement de marquage consiste à supposer qu'un composant est dans un état donné, l'enrichissement de marquage peut amener à supposer qu'un composant est dans deux états différents en même temps, ce qui est incohérent. Ainsi, il faut se doter des mécanismes nécessaires pour s'assurer de la cohérence de l'enrichissement, et ce, à chaque pas du raisonnement.

Avant de présenter les deux mécanismes de contrôle de la cohérence pour l'enrichissement de marquage, on introduit tout d'abord la notion de transition potentiellement franchissable.

2.7.3.1 Définition

Soit *R* un réseau de Petri marqué. Une *transition potentiellement franchissable* est une transition dont au moins une place amont est marquée (elle possède au moins un jeton) et au moins une place amont ne l'est pas (il lui manque au moins un jeton).

Le processus d'enrichissement de marquage consiste à ajouter les jetons manquants à la transition potentiellement franchissable de sorte qu'elle devienne franchissable. Mais il faudra vérifier qu'un unique composant du système ne se trouve pas dans deux configurations différentes en même temps avec cet enrichissement ce qui serait contradictoire avec la réalité du système physique. Il faut donc, interdire les enrichissements de marquage contradictoires avec la structure du système.

2.7.3.2 Mécanisme de contrôle de la cohérence

Le mécanisme de contrôle de la cohérence de l'enrichissement du marquage se fait par le calcul des invariants de places [Valette 92] du réseau. Le but est de vérifier que l'ajout d'un ou plusieurs jetons dans une ou plusieurs places du réseau ne viole pas un invariant de place. Si c'est le cas, l'enrichissement de marquage correspondant n'est pas autorisé.

Les invariants sont calculés une fois pour toutes, en prenant pour marquage initial celui qui correspond à l'état initial de tous les composants du système. La valeur obtenue pour le marquage correspondant à l'étape courante du raisonnement doit donc toujours être inférieure ou égale aux invariants. Elle sera inférieure si certains composants du système impliqués dans l'invariant ne sont pas nécessaires au scénario étudié. Si la valeur est supérieure, cela veut dire qu'il y a incohérence. Il serait en effet impossible d'associer un état du système complet à l'étape courante du raisonnement en ajoutant des jetons.

La figure 2.14 représente un système composé de deux ressources R1, R2 et une place partagée R3. R1 et R2 peuvent être en état de bon fonctionnement (marquage respectif des places R1_Ok et R2_Ok) ou en état de défaillance (marquage respectif des places R1_Ko et R2_Ko). Une ressource ne peut pas être dans deux états (bon fonctionnement et défaillance) en même temps d'où les invariants de places suivant :

$$R1\_Ok + R1\_Ko = 1$$
$$R2\_Ok + R2\_Ko = 1$$

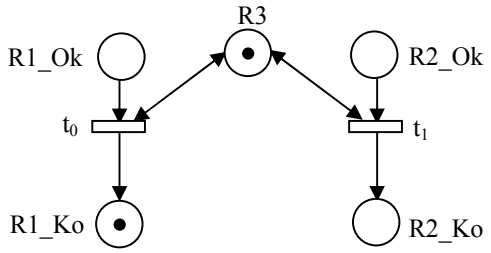


Figure 2.14. Contrôle de la cohérence du marquage

L'enrichissement de marquage nous indique la possibilité de mettre un jeton dans R1_Ok et un jeton dans R2_Ok. Après vérification des invariants de places, on n'autorise qu'un jeton dans la place R2_OK car la présence d'un jeton dans R1_Ko ne permet pas un autre jeton dans R1_OK (violation du premier invariant de places ce qui signifie que le système va se trouver dans deux états en même temps).

2.7.4. Limites

Le point fort de cette méthode est que le problème d'explosion combinatoire est évité par le choix d'extraire les scénarios directement à partir d'un modèle RdP du système sans passer par le graphe d'accessibilité. Mais le fait que l'approche proposée opère uniquement sur l'aspect discret du modèle a pour conséquence les limites suivantes:

- De nombreux scénarios incohérents vis-à-vis de la dynamique continue du système sont générés, d'où la nécessité de prendre en compte la partie continue du système.

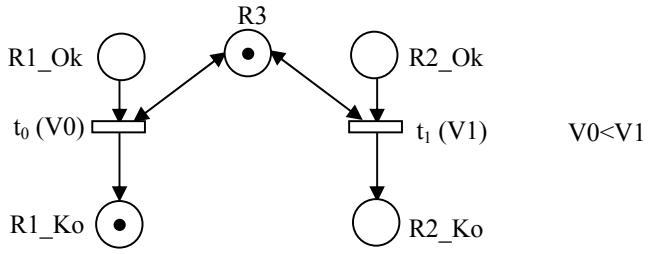


Figure 2.15. Incohérents vis-à-vis la dynamique continue

On reprend l'exemple de la figure 2.14 et on rajoute respectivement les seuils V0 et V1 aux transitions t_0 et t_1 correspondant à l'évolution des variables continues comme le présente la figure 2.15. L'application de la méthode présentée dans cette section donne les deux scénarios :

$$sc1 = \{t_0\}$$
$$sc2 = \{t_1\}$$

Alors que la dynamique continue du système ne permet pas une évolution vers V1 une fois V0 atteint et t_0 franchie. Ce qu'on appellera dans le chapitre suivant une transition interdite à cause de la dynamique continue du système.

- L'ordre d'occurrence des événements n'est pas pris en compte.
- Problème de redondance des scénarios.
- Problème de critère d'arrêt : qu'est ce réellement qu'un état normal ?
- Les scénarios générés ne sont pas minimaux.

2.8. Conclusion

Dans ce chapitre, nous avons présenté différents modèles des systèmes temporisés et hybrides. Nous avons introduit la notion de scénario, sa caractérisation et ses propriétés dans différents aspects : fiabilité et vérification. La représentation logique de ces scénarios était inévitable pour mieux caractériser les relations de causalité entre les événements menant vers l'état redouté.

Nous avons présenté une méthode d'extraction des scénarios redoutés proposée par [Khalfaoui 03], cette méthode basée sur l'extraction des scénarios redoutés directement à partir du modèle réseau de Petri est intéressante car elle permet d'éviter le problème d'explosion du nombre d'états. Mais cette méthode ne tient pas compte de l'aspect continu des systèmes ce qui donne des scénarios incohérents et non minimaux.

Mon travail de thèse est la suite de ces travaux. Nous proposons une approche [Medjoudj 04], [Medjoudj et al 04a], [Medjoudj et al 04b] pour une analyse qualitative de la sûreté de fonctionnement des systèmes pilotés par calculateurs d'un point de vue fiabilité dynamique. Cette approche a conduit à l'élaboration d'une nouvelle version de l'algorithme qui tient partiellement compte de l'aspect continu et plus particulièrement des seuils associés à certaines transitions dans le modèle RdP. Cette nouvelle approche permet d'éliminer les scénarios incohérents vis-à-vis de la dynamique continue du système.

Dans le chapitre 3, nous allons présenter cette nouvelle méthode qui tient compte de l'aspect hybride des systèmes pilotés par calculateurs et nous allons l'appliquer sur un exemple classique dont le fonctionnement est similaire au fonctionnement d'un système mécatronique. Cette méthode sera ensuite orientée, dans les chapitres 5 et 6, vers la vérification de certaines propriétés temporelles des systèmes hybrides [Rivière et al 05].

Chapitre 3 : Recherche de scénarios dans les systèmes hybrides

3.1. Introduction

L'aspect hybride des systèmes pilotés par calculateurs nous a conduit au choix d'une modélisation associant Réseaux de Petri (RdP) et équations différentielles. Le modèle RdP décrit le fonctionnement nominal, les défaillances et les mécanismes de reconfiguration. Les équations différentielles représentent l'évolution des variables continues de la partie énergétique du système. Nous allons nous appuyer lorsque c'est nécessaire sur une abstraction temporelle du système hybride sous la forme d'un réseau de Petri p-temporel.

Dans ce chapitre, nous proposons une nouvelle version de l'algorithme développé par [Khalfaoui03] qui permet la construction des scénarios critiques à partir d'un modèle Réseau de Petri mais opère uniquement sur l'aspect discret du modèle et qui génère par conséquent des scénarios incohérents avec la dynamique continue du système. Cette nouvelle version est plus précise car elle tient compte partiellement de l'aspect continu du système hybride étudié. Ces scénarios caractérisent comment le système quitte le fonctionnement normal pour évoluer vers l'état redouté en déterminant la suite d'actions et de changements d'états conduisant à l'état critique. Nous allons montrer après une illustration de l'algorithme sur un cas d'étude, comment un grand nombre de scénarios cohérents vis-à-vis de la partie discrète du modèle hybride mais incohérents vis-à-vis de sa partie continue ne sont plus générés grâce à l'abstraction temporelle.

3.2. Méthode d'extraction des scénarios dans un système hybride

3.2.1. Principe

Cette méthode est composée de deux étapes : un raisonnement arrière et un raisonnement avant. Le raisonnement arrière prend comme marquage initial dans le réseau de Petri inversé le seul état cible (redouté) et cherche de façon exhaustive tous les scénarios permettant de consommer le marquage initial (état redouté puisque l'on raisonne en arrière) et aboutissant à un marquage final formé uniquement de places associées au fonctionnement normal. Le raisonnement avant prend comme état initial ces places de fonctionnement normal dans le réseau de Petri initial. Son objectif est de localiser les bifurcations entre le comportement redouté et le fonctionnement normal du système ainsi que les conditions impliquées dans ces bifurcations. Ainsi on dispose non seulement de l'explication du comportement dangereux ("comment peut-on arriver dans l'état redouté) mais aussi de stratégies permettant de l'éviter. Un point important de la méthode est que le contexte dans lequel s'est produit l'évènement redouté est enrichi progressivement. Chaque scénario est donné sous la forme d'un ordre partiel entre les évènements nécessaires à l'apparition

de l'évènement redouté ce qui diffère d'un arbre de défaillance qui donne un ensemble de combinaisons statiques des états partiels nécessaires pour l'obtention de l'état redouté.

3.2.2. Prise en compte du continu par des abstractions temporelles

3.2.2.1 Principe

Nous allons, dans cette section, mettre en évidence les modifications que nous avons effectuées par rapport à l'algorithme précédent pour diminuer le nombre de scénarios qui étaient générés car ils étaient cohérents avec la seule vue discrète, mais qu'il fallait rejeter dans une seconde étape car ils étaient incohérents vis-à-vis de la dynamique continue.

La méthode que nous proposons prend en compte les conditions associées au franchissement de certaines transitions. Ces conditions sont des seuils impliquant des variables continues. Par approximation temporelle de la dynamique hybride, ces seuils sont transformés en des durées qui correspondent au temps que met le système pour les atteindre quand les transitions sont sensibilisées. Comme, pour l'instant nous travaillons uniquement d'un point de vue qualitatif et non quantitatif, ce que nous retenons c'est, lorsque c'est possible, l'ordre suivant lequel ces transitions pourront être franchies. Ainsi nous pourrons nous trouver dans une situation où deux transitions t1 et t2 sont franchissables si l'on considère le réseau de Petri ordinaire, mais qui sont telles que t1 sera toujours franchie avant t2 si l'on considère l'abstraction temporelle. Dans la génération des scénarios seul le franchissement de t1 sera considéré puisque celui de t2 avant t1 serait en fait incohérent avec la dynamique continue. Dans la nouvelle version de l'algorithme cela apparaît sous la forme d'une sorte de priorité : si t1 et t2 sont franchissables, seul celui de t2, plus prioritaire, est examiné.

Pour prendre en compte ce mécanisme, cette version de l'algorithme introduit une nouvelle liste de données (Lint2) qui est une liste de transitions franchissables ou potentiellement franchissables à ne pas franchir car elles sont en conflit avec des transitions qui doivent être franchies avant elles (à cause de l'aspect continu). La prise en compte de ces relations de précédence provenant de la dynamique continue et non spécifiées par le réseau de Petri ordinaire permet de réduire le nombre de scénarios générés en éliminant un certain nombre de scénarios incohérents vis-à-vis de la dynamique continue

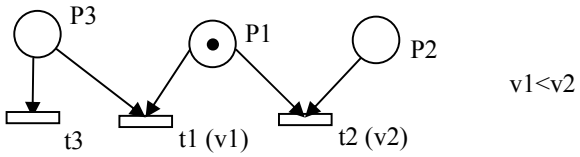


Figure 3.1. La priorité due aux seuils associés aux transitions

Considérons un exemple. Dans la figure 3.1 on suppose que le système algébro-différentiel associé à la place P1 garantit que la variable x est croissante. A la transition t1 on associe le seuil x=v1 et à la transition t2 le seuil x=v2 avec v1<v2. On suppose enfin que lorsque le jeton arrive dans la place p1 nous avons toujours x< v1. Alors, si la place P3 est marquée, la transition t1 sera franchie avant t2 puisque le seuil associé à t1 est inférieur à celui de t2. Dans ce cas on ne

considère pas le scénario associé au franchissement de t2. Par contre, si t3 est déjà franchie (place P3 vide) et si la place P2 est marquée, on ne peut plus franchir t1 et t2 sera alors franchie.

3.2.2.2 Précédence et causalité directe et indirecte

Dans l'exemple ci-dessus, nous n'examinerons finalement qu'un seul type de scénarios, ceux pour lesquels la transition t2 est franchie après t3. Nous avons donc une relation de précédence entre le franchissement de t3 qui vide la place P3 et celui de t2. Pourtant aucune place ne relie t3 à t2.

Cette relation de précédence est donc une conséquence de la dynamique continue et des seuils associés aux transitions t1 et t2. Nous parlerons dans ce cas de relation de précédence indirecte et de causalité indirecte.

Les relations de précédence et de causalité directes sont celles qui sont mises en évidence par le seul réseau de Petri, c'est-à-dire par le seul aspect discret. Par exemple dans la figure 3.2, la place P2 induit une relation de causalité directe entre le franchissement de t1 et celui de t2. Il faut avoir produit un jeton dans la place P2 par le franchissement de t1 pour pouvoir franchir t2.

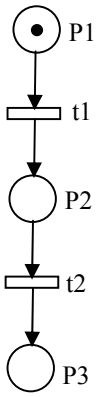


Figure 3.2. Lien de causalité direct

3.2.2.3 Cas de priorités entre les franchissements des transitions

Nous venons de voir que la dynamique continue pouvait induire des relations de précédence entre des franchissements de transitions. Un phénomène tout à fait analogue peut se produire si l'on introduit des règles de priorité entre les franchissements des transitions. Bien que nous n'ayons pas introduit explicitement des règles de priorité entre les franchissements des transitions, elles peuvent s'avérer nécessaires pour représenter des stratégies de commande et de reconfiguration de façon simple. De plus, à cause de ce qui précède, elles seront prises en compte de façon simple dans notre algorithme.

Considérons par exemple le réseau de Petri de la figure 3.3 a. Il représente une fenêtre temporelle (la place p1 contient un jeton entre les franchissements de t1 et de t2) pendant laquelle le traitement t4 non seulement peut être effectué, mais *doit* être effectué sans attente s'il y a une

demande (présence d'un jeton dans P2 suite au franchissement de t3). De façon classique, cela peut être exprimé en rendant le franchissement de t4 prioritaire par rapport à celui de t2.

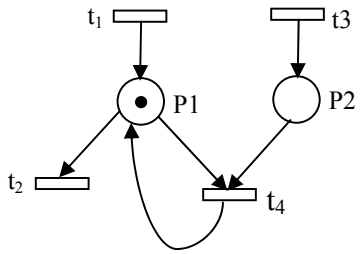


Figure 3.3 a. Lien de causalité indirecte

En ce qui concerne les relations de précédence, seuls deux types de scénarios pourront avoir lieu. Celui de la figure 3.3.b pour lequel t3 est franchie après t2 et celui de la figure 3.3.c pour lequel t4 est franchie avant t2. La relation de précédence et de causalité entre t2 et t3 dans la figure 3.3.b est une relation indirecte comme précédemment car elle ne correspond pas à une place reliant les deux transitions.

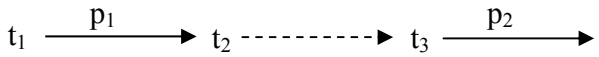


Figure 3.3 b. Lien de causalité indirecte entre t_2 et t_3

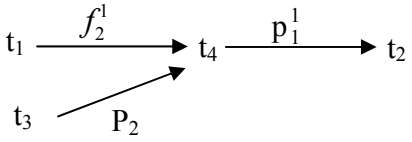


Figure 3.3 c. Lien de causalité direct entre t4 et t2

3.2.2.4 Retour sur le principe de l'amélioration de l'algorithme

Ce que nous venons de présenter dans cette section c'est le principe de l'amélioration de l'algorithme de recherche des scénarios. Grâce à une abstraction temporelle de la dynamique continue, nous mettons en évidence des relations de précédence et de causalité indirectes entre certains franchissements de transition. Dans le cœur de l'algorithme cela est exprimé sous la forme de règles de priorité (une certaine transition n'est pas franchie si une autre est franchissable). Dans l'expression des résultats, c'est-à-dire des scénarios, cela apparaît sous la forme de relations de précédence/causalité indirectes entre des transitions qui ne sont pas reliées par une place. Ainsi nous allons restreindre, parfois fortement, le nombre de scénarios générés et pour chaque scénario l'ensemble des séquences de franchissements cohérentes avec la relation

d'ordre partiel strict associée au scénario. La phase de vérification du fait que ces scénarios sont effectivement cohérents vis-à-vis de la dynamique continue en sera simplifiée d'autant.

Dans le paragraphe suivant nous donnons une description de l'algorithme de génération de scénarios. Nous présentons la structure de données, les procédures et les différentes étapes. Il est à noter qu'une seule exécution de l'algorithme génère automatiquement plusieurs scénarios.

En effet partant d'un état redouté, on remonte dans la chaîne des causalités dans le réseau de Petri inverse jusqu'à un état de fonctionnement normal, on arrête alors le raisonnement arrière et on entame une démarche avant. Dans cette dernière appelée « raisonnement avant » on commence par l'état de fonctionnement normal trouvé précédemment, et on parcourt le réseau de Petri pour atteindre l'état redouté en enrichissant le marquage quand c'est nécessaire. Tous les scénarios possibles et cohérents vis-à-vis de la dynamique continue du système sont ainsi générés.

3.2.3. Algorithme de recherche de scénarios dans un système hybride

L'algorithme est constitué de différentes procédures et structures de données manipulant des ensembles de transitions qui peuvent être en conflit pour pouvoir générer tous les scénarios possibles.

3.2.3.1 Structure des données

Trois sortes de structures de données sont nécessaires pour le déroulement de l'algorithme : des données d'entrée, des données de sortie et des données internes.

Données d'entrée : elles sont composées de la liste des jetons initiaux (notée Li) et de la liste des jetons normaux (notée Ln) qui sera utilisée comme l'un des deux critères d'arrêt. Quand tous les jetons présents dans la liste courante des jetons sont normaux non initiaux (liste LjnInit), on s'arrête. Une liste de transitions interdites LintEntree sert à bloquer certaines transitions du réseau dès le début de raisonnement.

Données de sortie : l'algorithme génère un ensemble d'ordres partiels ; certains correspondent au bon fonctionnement du système, d'autres aux reconfiguration et d'autre aux différents scénarios menant à l'état redouté. Chaque ordre partiel est défini par un triplet de la forme (E, A, B) avec E la liste des instances de tir des transitions, A et B représentent l'ensemble des relations de précédence générant l'ordre partiel. A est la liste des arcs reliant des éléments de E possédant des liens de causalité directe, B la liste des arcs reliant des éléments de E possédant des liens de causalité indirecte. L'algorithme génère également une liste des jetons créés à la suite d'un enrichissement de marquage. Cette liste est nommée Le.

Données internes :

- La liste courante, notée Lc contient la liste des jetons courants qui est mise à jour après chaque franchissement de transition. Les jetons consommés sont enlevés et les jetons produits sont ajoutés. Pour générer la liste Lc, chaque jeton est représenté sous la forme d'un couple (e, p) ; e est l'évènement qui a produit ce jeton et p est la place le contenant.

- La liste des jetons d'enrichissement, noté Le contient les jetons d'enrichissement. A chaque enrichissement cohérent de marquage, on rajoute le couple (e, p) à la liste Le.

- La liste des jetons interdits à cause de conflit d'enrichissement, notée LeInt.

- La liste de jetons normaux non initiaux, notée LjnInit contient les jetons de Ln qui ne sont pas initiaux. A chaque franchissement d'une transition tk, on rajoute le couple (tk, p) à Lc et on vérifie si la place $p \in Ln$, on la rajoute à LjnInit qui va nous servir comme critère d'arrêt.

- La liste des transitions interdites de premier niveau, notée Lint1 contient la liste des transitions à ne pas franchir à partir d'une étape courante donnée. Elle permet de gérer les conflits de transitions. Dès qu'une transition, en conflit avec une autre, est franchie, elle est ajoutée à cette liste pour ne pas la franchir une deuxième fois ; ce qui empêche la génération d'un même ordre partiel plusieurs fois.

- La liste des transitions interdites de deuxième niveau Lint2 contient la liste des transitions franchissables ou potentiellement franchissables à ne pas franchir. C'est-à-dire parce que la transition en question est en conflit avec une transition franchissable qui a un seuil de franchissement inférieur.

- La liste des transitions interdites de troisième niveau Lint3 contient la liste des transitions qui ne peuvent être enrichies pour éviter les boucles en essayant de les enrichir. C'est-à-dire une incohérence d'enrichissement est détectée et que ces transitions ne peuvent donc être enrichies et franchies.

- Le contexte, noté Ci : contient la liste de jetons courants (Lc), les listes de transitions interdites (Lint1 et Lint2, Lint3), l'ordre partiel construit (E, A et B), et la liste des jetons enrichis (Le). Ci est de la forme (Lc, Lint1, Lint2, Lint3, E, A, B, Le, LeInt, LjnInit). La liste C contient les contextes Ci, elle est de la forme C= {C0, C1, …Ci….Cn}.

D'autres listes de données internes sont générées à partir de la liste courante Lc. Ces données concernent l'ensemble des transitions franchissables et potentiellement franchissables sachant que les transitions franchissables sont prioritaires par rapport à celles qui sont potentiellement franchissables. La propriété dans le traitement des transitions de ces listes est la suivant :

- La liste des transitions franchissables en conflit ou sans conflit, notée tfcEsc.
- La liste des transitions potentiellement franchissables en conflit soit avec des transitions franchissables, soit avec des transitions potentiellement franchissables, notée Tpfc.

- La liste des transitions potentiellement franchissables sans conflit, noté Tpfsc.

3.2.3.2 Procédures

Tirer Transition : dans cette procédure, la liste courante est mise à jour suite au tir de la transition en question en enlevant les jetons consommés et en ajoutant les jetons produits. Les événements sont mémorisés dans E et tous les arcs correspondant à une relation de précédence entre deux événements dans A. Chaque jeton étant associé à l'événement l'ayant produit, la

relation de précédence correspondante est directement obtenue lorsqu'il est consommé. La procédure est identique à l'étiquetage de l'arbre de preuve.

Tirer Transition (tk)

Ajouter Tk dans E,

Pour chaque jeton (ti, P) nécessaire pour franchir tk enlever (ti, p) de la liste Lc et ajouter (ti, tk) dans A,

Pour chaque place de sortie Ps de tk, ajouter un jeton de la forme (tk, Ps) dans Lc

Si Ps est une place normale, ajouter Ps à `LjnInit`

Enrichir Marquage : l'enrichissement du marquage consiste à ajouter des jetons dans la liste des jetons courante de façon à rendre franchissables des transitions qui ne sont que potentiellement franchissables (certaines de leurs places d'entrée contiennent des jetons, mais d'autres sont vides). Cela correspond à la prise en compte de l'état de nouveaux composants du système étudié car leur interaction avec les composants déjà pris en compte peut provoquer des changements d'état. Il faut toutefois vérifier que l'ajout de ces jetons n'est pas incohérent vis-à-vis du modèle du système étudié.

La procédure "`Enrichir Marquage`" permet donc de vérifier la cohérence de l'enrichissement de marquage. Elle ne l'autorise que lorsque cela est cohérent par rapport aux invariants de places. On distingue deux types de procédure pour l'enrichissement. La première, appelée `Enrichir Marquage1` permet l'enrichissement du marquage vis-à-vis des transitions potentiellement franchissables en conflit avec une transition appartenant à l'ensemble Tfcpf. La deuxième appelée `Enrichir Marquage2` permet l'enrichissement des autres transitions potentiellement franchissables.

Enrichir Marquage1 (tk)

L, variable interne de la procédure, est une liste de jetons initialement vide.

Pour chaque transition tj, potentiellement franchissable en conflit avec tk, et pour chaque place pl en amont de tj, ajouter un jeton (ek, pl) dans la liste L,

Vérifier la cohérence du marquage

Enrichir Marquage2 (tk)

L, variable interne de la procédure, est une liste de jetons initialement vide.

Pour chaque place pl en amont de tk, ajouter un jeton (ek, pl) dans la liste L,

Vérifier la cohérence du marquage

Cohérence Marquage : cette méthode vérifie la cohérence du marquage. Elle fait appel à une liste interne L qui contient des éléments de type (ti, P) qui sont les jetons d'enrichissement; la liste des invariants et la liste de la composante conservative de chaque invariant qui sont des données d'entrée.

Cohérence Marquage (L, tk)

Pour chaque jeton pl de L,

Vérifier pour chaque invariant s'il contient pl

Vérifier si le nombre de jetons dans les places constituant cet invariant est supérieur à la composante conservative de cet invariant alors on supprime pl de L.

Si le nombre d'éléments dans L est supérieur à 0 alors l'enrichissement est cohérent, on rajoute les atomes de L dans les listes Lc et Le.

Sinon on rajoute tk à la liste des transitions qui ne peuvent pas être enrichies Lint3 pour éviter les boucles.

Conflit enrichissement : Dans l'enrichissement de marquage, on est confronté des fois au cas ou on ne peut mettre les jetons dans toutes les places à cause de l'incohérence de marquage. On met les jetons graduellement dans les places qui nécessitent un enrichissement et on vérifie à chaque fois que l'enrichissement est cohérent. En cas d'incohérence à cause d'un jeton d'enrichissement dans la phase courante, on mémorise le contexte et on continue l'enrichissement. Ce procédé nous permet de ne pas oublier des contextes du système étudié.

Conflit enrichissement (pk)

Mettre un jeton d'enrichissement dans la place pk,

Vérifier la cohérence de marquage

Mettre graduellement des jetons dans les autres places qui nécessite un enrichissement

En cas d'incohérence à cause de pk, mémoriser_contexte enrichissement pk.

Mémoriser contexte Conflit enrichissement (pk)

Ajouter le jeton de la place pk à la liste des jetons interdits LeInt,

Ajouter un nouvel élément Ci (Lc, Lint1, Lint2, Lint3, E, A, B, Le, LeInt LjnInit) au contexte C. Lint2 et Lin3 étant initialement vides dans ce nouveau contexte.

Pour eviter la mémorisation d'un même contexte d'enrichissement, à chaque mémorisation le jeton est rajouté à la liste des jetons IntEnrich qui est une variable globale.

Mémoriser Contexte conflit transitions : comme on l'a mentionné avant, à chaque fois qu'un conflit est rencontré pendant la construction d'un ordre partiel, ce dernier est scindé en autant d'ordres partiels différents que de transitions impliquées dans le conflit. Cette procédure permet de mémoriser toute l'information nécessaire pour construire un autre ordre partiel correspondant au tir d'une autre transition en conflit avec cette dernière.

Mémoriser Contexte (tk)

Ajouter la transition tk à la liste des transitions interdites Lint1,

Ajouter un nouvel élément Ci (Lc, Lint1, Lint2, Lint3, E, A, B, Le, LeInt LjnInit) au contexte C. Lint2 et Lin3 étant initialement vides dans ce nouveau contexte.

Conflit transitions tk et tj : cette méthode vérifie si tk est en conflit avec tj. S'il y a au moins une place qui est amont aux deux transitions alors les deux transitions sont en conflit

D'autres procédures sont utilisées comme, trier la liste des transitions selon le seuil croissant, la construction des liste des transition interdite Lint2, vérifier si deux transitions en conflit ont des seuil de franchissement disjoint pour choisir la transition prioritaire,etc.

3.2.3.3 Différentes étapes du l'algorithme

(0). Pas initial

1) initialise le contexte C avec un élément (Lc, Lint1, Lint2, Lint3, E, A, B, Le, LeInt, LjnInit) pour générer le premier ordre partiel :
Lint1, Lint2, Lint3, LjnInit, Le et A sont vides, Lc=Li (ensemble des jetons initiaux), E= {i} et l'entier **Inc** est égal à 1.
C← (Lc = Li, Lint1 = Lint2= Lint3 = {}, E = {i}, A = B = {}, Le= LeInt = {}, LjnInit = {})
2) Définir la priorité de tir de transitions due à l'aspect continu du système en associant à chaque transition tk un seuil temporel (un intervalle) de franchissement D = [dmin, dmax]. La transition tk sera franchie à l'instant $\tau k \in$ D.

(1). Nouvel ordre partiel

// Construire un nouvel ordre partiel //
Si C = {} **alors** faire **(8). Pas final** *;*
Sinon

- Mémoriser le premier élément de C dans (Lc, Lint1, Lint2, Lint3, A, B, E, Le, LeInt, LjnInit) ;
- Effacer cet élément de C ;
- Faire **(2). Différentes listes transitions;**

(2). Différentes listes transitions

Générer à partir de Lc toutes les transitions franchissables (tfcEsc) et potentiellement franchissables (Ltpf). Effacer de ces listes:

- Les transitions de E. En effet comme la recherche des boucles est complexe, nous avons dans un premier temps choisi de ne franchir chaque transition qu'au plus une fois.
- Les transitions de Lint1 qui sont en conflit avec les transitions frnachissables pour éviter de générer plus d'une fois un même ordre partiel.
- Les transitions de Lint3 qui sont des transitions qui ne peuvent pas être enrichies et franchies dans le contexte pour éviter les boucles infinies.
- Les transitions de LintEntree interdites à l'entrée
- Générer les listes suivantes : Tpfsc et Tpfc.
- Faire **(3). Critère d'arrêt**

(3). Critère d'arrêt

//Critère d'arrêt de la construction d'un ordre partiel//

Si Lc contient uniquement des jetons de Ln et qui ne sont pas initiaux (LjnInit) **ou** les listes tfcEsc, Tpfsc, et Tpfc sont toutes vides **alors** faire **(7).Générer scénario ;**
Sinon faire **(4). tfcEsc ;**

(4). tfcEsc
//Les transitions franchissables sont tirées en priorité. En cas de conflit de transition, cette étape les résout en en tirant une et mémorise l'information nécessaire pour la construction des autres ordres partiels relatifs aux tirs des autres transitions impliquées dans le conflit. Dans cette étape on effectue un enrichissement de marquage si c'est nécessaire.
Si TfcEsc = {} alors faire **(*5*).Tpfc**;
Sinon

- Trier tfcEsc selon le seuil croissant ;
- Soit tk la première transition de tfcEsc ;
- Rajouter à Lint2 toute transition franchissable tj de seuil supérieur en conflit avec tk (intervalles temporels de tir disjoints)
- Enrichir Marquage1 (tk) s'il y a lieu ; Mémoriser contexte conflit d'enrichissement s'il y a lieu ;
- **Si** l'enrichissement n'est pas cohérent ;
  Si tk n'est pas en conflit avec une transition franchissable de seuil inférieur;
  Si tk est en conflit avec une transition qui n'appartient pas à Lint1, Lint2, Lint3, E ou LintEntree (intersection d'intervalle temporel de tire) ;
  Alors Mémoriser Contexte (tk) ;
  Alors Rajouter s'il y a lieu la relation de causalité indirecte (tj, tk) à B ;
  Tirer Transition (tk) ;
- faire **(2).Différentes listes transitions ;**

(5).Tpfc
Si Tpfc = {} **alors** faire **(6).Tpfsc**;
Sinon

- Ordonner la liste Tpfsc selon leur seuil croissant ;
- Choisir la première transition tk dans Tpfc ;
- Mettre les autres transitions de seuil supérieur en conflit avec tk dans Lint2 ;
- Enrichir Marquage2 (tk) ;
- **Si** l'enrichissement est cohérent ;
  Si tk n'est pas en conflit avec une transition franchissable de seuil inférieur ;
  Si tk est en conflit avec une transition qui n'appartient pas à Lint1, Lint2, Lint3, E ou LintEntree (intersection d'intervalle temporel de tire) ;
  Alors Mémoriser Contexte (tk) ;
  Alors Rajouter s'il y a lieu la relation de causalité indirecte (tj, tk) à B ;
  Tirer Transition (tk) ;
- faire **(2).Différentes listes transitions ;**

(6). Tpfsc
//On enrichit le marquage d'une transition potentiellement franchissable sans conflit et on la franchit ensuite//
Si Tpfsc = {} **alors (7).Générer scénario** ;
Sinon

- Ordonner la liste Tpfsc selon leur seuil croissant

- Choisir la première transition tk dans Tpfsc ;
- Enrichir Marquage 2(tk) ;
- **Si** l'enrichissement est cohérent ;
  Tirer Transition (tk) ;
- faire **(2).Différentes listes transitions ;**

(7).Générer scénario

//On mémorise l'ordre partiel construit et on revient au (1). Nouvel ordre partiel//

Pour chaque atome (ti, p) de Lc ajouter (ti, f) dans A ; f étant l'événement fin ;

- Mémoriser l'ordre partiel dérivé numéro ***Inc*** tel que : E (***Inc***) = E, A (***Inc***) = A, B (***Inc***) = B, Le (***Inc***) = Le, LjnInit (***Inc***) = LjnInit ;
- Incrémenter ***Inc*** ;
- Faire **(1). Nouvel ordre partiel ;**

(8). Pas final

- Afficher tous les scénarios générés

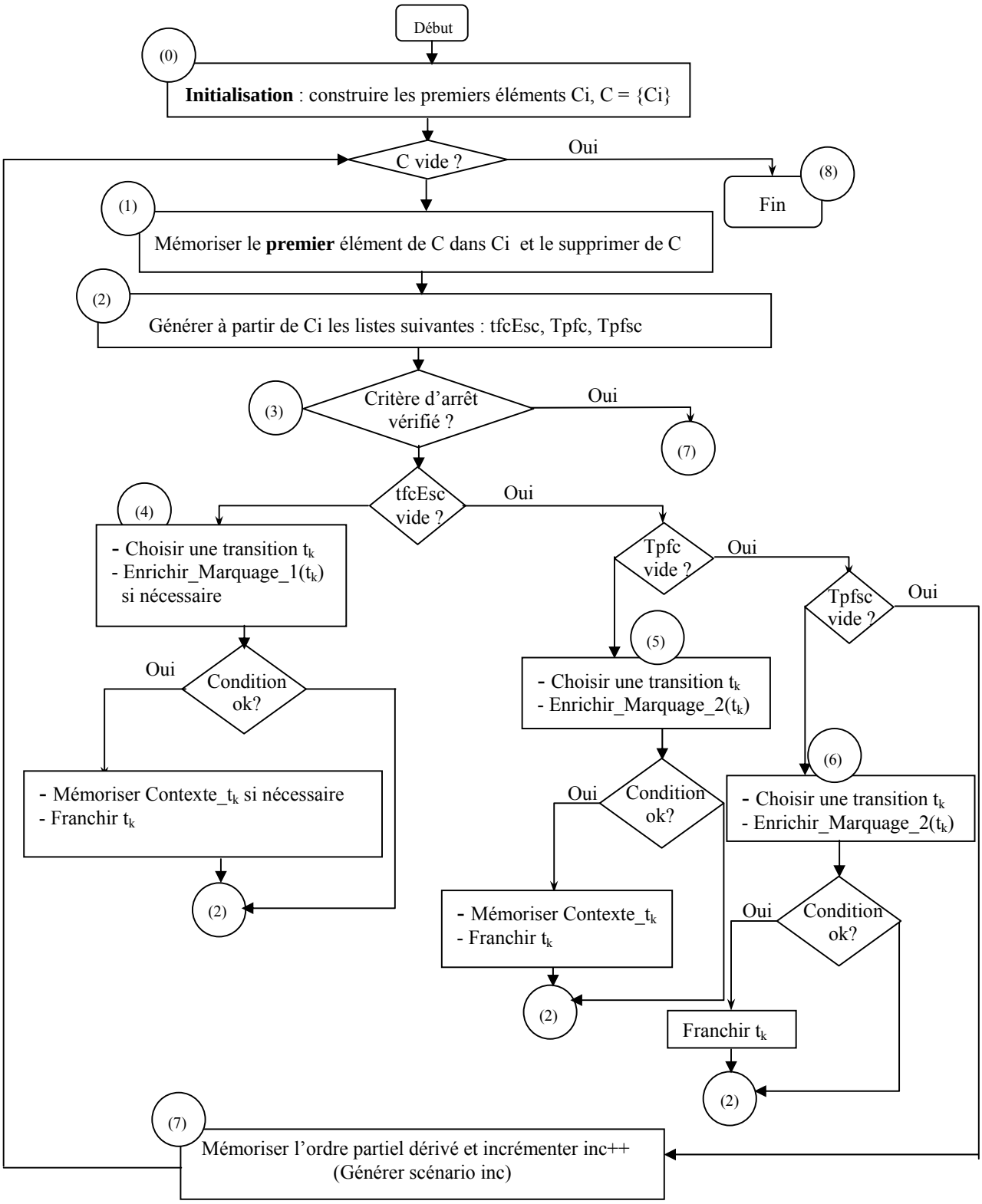


Figure 3.4. Algorithme de génération des scénarios critiques avec l'aspect Continu

La figure 3.4 représente l'organigramme de l'algorithme. Toutes les transitions sont choisies selon le seuil croissant dans leurs listes.

3.3. Application de l'algorithme sur un cas d'étude

Nous allons d'abord présenter le cas d'étude pour appliquer ensuite l'algorithme de recherche de scénarios redoutés. Nous allons montrer comment le fait de tenir compte de la dynamique continue du système permet le respect d'ordre d'occurrence des événements et l'élimination des scénarios incohérents vis-à-vis de la dynamique continue du système.

3.3.1. Présentation du cas d'étude

Le réseau de Petri de la figure 3.5 représente un système qui peut être dans 3 états : en état d'arrêt (quand la place *OFF* est marquée), en état de fonctionnement nominal (quand la place *N* est marquée) ou en état de fonctionnement dangereux (quand la place *E-red* est marquée). Le démarrage du système est modélisé par le tir de la transition t_0, son fonctionnement nominal est modélisé par le tir de la transition *t1* et son évolution vers l'état redouté (dangereux) est modélisée par le tir de *red*. La ressource *Rs-ok* qui permet de maintenir le système dans un état de fonctionnement nominal peut être dans deux états : un état de fonctionnement nominal (*Rs-ok* marquée) ou dans un état défaillant (place *Rs-ko* marquée).

La loi de commande de ce système est telle que lorsque la variable continue dépasse une limite supérieur *V2* (seuil associé à la transition *t1*), le système demande l'utilisation de la ressource *Rs-ok* pour ramener la valeur de la variable continue au seuil de bon fonctionnement. Maintenant si la ressource est hors service (la place *Rs-ko* est marquée), la valeur de la variable continue augmente jusqu'à une limite dite dangereuse (*V1* associé à la transition *red*) ce qui provoque l'événement redouté. Nous avons les deux invariants de place suivants :

$$M(OFF) + M(N) + M(E\text{-}red) = 1 \qquad (1)$$

$$M(Rs\text{-}ok) + M(Rs\text{-}ko) = 1 \qquad (2)$$

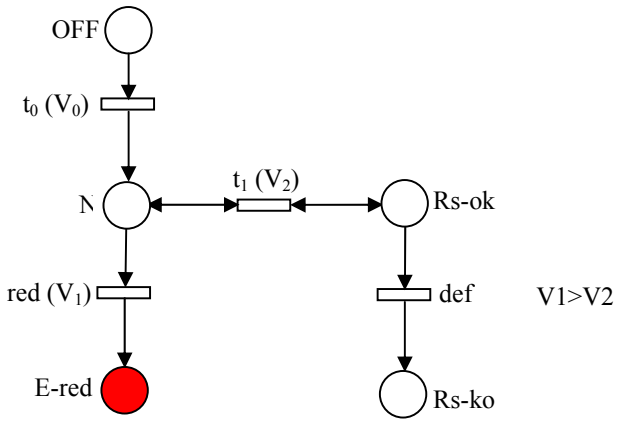


Figure 3.5. Modèle réseau de Petri du cas d'étude

3.3.2. Application de l'algorithme

Avant l'application d'algorithme de recherche de scénarios, il est nécessaire de déterminer l'état cible et les états nominaux. Dans notre cas d'étude, l'état cible (redouté) correspond au marquage de la place *E-red*, les états nominaux sont les places *OFF*, *N* et *Rs-ok.* Nous avons également besoin d'associer des intervalles temporels aux transitions correspondant au temps que met la variable pour atteindre les seuils. Dans cet exemple, *t0*, *t1*, et *red* sont des transitions déterministe : *t0* est immédiate tandis que *t1* peut être franchie à l'intervalle de temps [2, 2] et *red* à l'intervalle de temps [3, 3], *def* est une transition stochastique, elle représente la défaillance de la ressource sollicitée par le système elle peut être défaillante à tout moment.

3.3.2.1 Raisonnement arrière

Comme nous l'avons déjà vu, Le raisonnement arrière opère sur le réseau de Petri inversé du cas d'étude comme présenté dans la figure 3.6. L'abstraction temporelle est difficile à utiliser en raisonnement arrière.

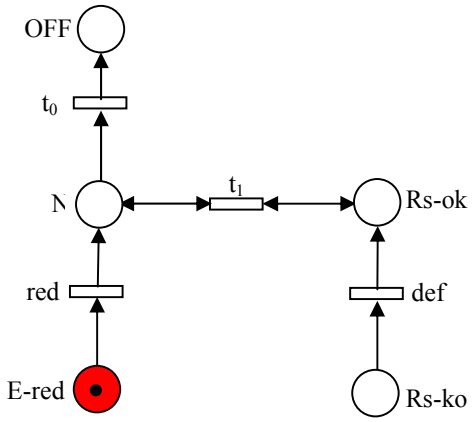


Figure 3.6. Modèle réseau de Petri inverse du cas d'étude : raisonnement arrière

(0). Pas initial
Lc = (I1, E_red), A = {}, E = {I1}, Le = LeInt ={}, Lint 1 = Lint2 = Lint3= {}, LjnInit = {}.
C0 (Lc, A, B, E, Lc, Le, LeInt, Lint1, Lint2, Lint3, LjnInit), C={C0}

(1). Nouvel ordre partiel

C n'est pas vide d'où Ci(Lc, A, E, Lc, Le, LeInt, Lint1, Lint2, Lint3, LjnInit) reçoit C0(Lc = (i, E_red), A = {}, B={}, E = {I1}, Le = {}, LeInt = {}, Lint 1 = Lint2 = Lint3= {}, LjnInit = {})

C devient vide ;
Faire **(2). Différentes listes transitions;**

(2) . Différentes listes transitions

La seule transition franchissable est *red* qui n'est en conflit avec aucune autre transition. Aucune transition n'est potentiellement franchissable d'où : tfcEsc ={red}, Tpfsc = {} Tpfc = {}.
Faire **(3). Critère d'arrêt**

(3). Critère d'arrêt

Lc ne contient pas uniquement des jetons appartenant à Ln (E_red n'est pas un jeton normal).
Faire **(4). tfcEsc ;**

(4). tfcEsc

tfcEsc ={red} n'est pas vide donc :

Soit tk= red;

Tirer transition (red);

E ={I1, red }, A={(I1, red)}, Lc ={(red, N)}, LjnInit = {N};
Revenir au **(2). Différentes listes transitions;**

(2). Différentes listes transitions

La liste courante contient un jeton dans la place N. A partir de ce marquage, il y a une transition franchissable t0 et une transition potentiellement franchissable t1 (t0 est en conflit avec t1) d'où :

tfcEsc ={t0}, Tpf={t1} ;

On élimine de ces ensembles la liste des transitions de Lint1, Lint3 et E on obtient :

tfcEsc ={t0}, Tpf={t1} d'où :
tfcEsc ={}, Tpfsc = {}, {t0}, Tpfc = {t1} ;

(3). Critère d'arrêt

Lc ={(red, N)} contient uniquement des jetons normaux qui ne sont pas initiaux (de LjnInit), le critère d'arrêt est donc satisfait.
Faire **(7).Générer scénario** ;

(7).Générer scénario

L'ordre partiel construit par ce raisonnement arrière est E1={I1, red}, A1={(I1, red), (red,F1)},
Revenir au **(1). Nouvel ordre partiel**

(1). Nouvel ordre partiel
Le contexte C est vide c'est donc la fin de l'algorithme de raisonnement arrière. L'ordre partiel obtenu nous a conduit au marquage de la place normale N qui sera le point de départ du raisonnement avant.

Faire **(8). Pas final ;**

(8). Pas final

On obtien un contexte qui sera utilisé pour commencer le raisonnement avant : c0(Lc={N}).

3.3.2.2 Raisonnement avant

Ce raisonnement avant s'opère sur le réseau de Petri initial (figure 3.7) avec comme marquage initial celui obtenu par le raisonnement arrière : un jeton dans la place N.

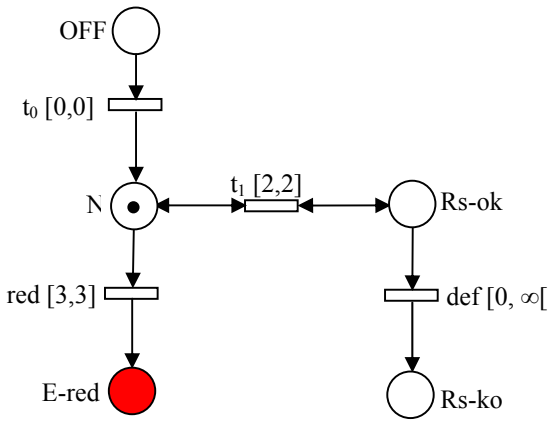


Figure 3.7. Modèle réseau de Petri du cas d'étude : raisonnement avant

(0). Pas initial
Lc = (I1, N)}, A = {}, B={}, E = {I1}, Le = LeInt ={}, Lint 1 = Lint2 = Lint3= {} LjnInit = {}.
LjnInit est initialisé à 0 pour chaque ordre partiel.
C0 (Lc, A, B, E, Lc, Le, LeInt, Lint1, Lint2, Lint3, LjnInit), C={C0}

(1). Nouvel ordre partiel
C n'est pas vide d'où Ci(Lc, A, B, E, Lc, Le, LeInt, Lint1, Lint2, Lint3, LjnInit) reçoit C0(Lc = (I1, N), A = {}, B = {}, E = {I1}, Le = LeInt ={}, Lint 1 = Lint2 = Lint3= {}, LjnInit = {}) ;
C devient vide ;
Faire **(2). Différentes listes transitions;**

(2). Différentes listes transitions
La liste courante Lc contient un jeton dans la place N qui est initial. A partir de ce marquage, il y a une transition franchissable red et une transition potentiellement franchissable t1 (red est en conflit avec t1) d'où :
tfcEsc ={red}, Tpf={t1} ;
On élimine de ces ensembles la liste des transitions de Lint1, Lint3 et celles de E on obtient :
tfcEsc ={red}, Tpf ={t1} d'où :
tfcEsc ={ red }, Tpfsc = {}, Tpfc = {t1}.

(4). tfcEsc
tfcEsc= {red}n'est pas vide. red est une transition franchissable en conflit avec une transition potentielement franchissable, elle nécéssite donc un enrichissement). Red est la seule dans sa liste s'il y avait plusieurs transitions, on aurait choisi celle de seuil inférieur.
Soit tk=red
Enrichir Marquage 1
Initialement L ={}
la seule transition en conflit avec red est t1. Rs-ok est la seule place en amont de t1 non marquée. Nons ajoutons un jeton (e1, Rs-ok) à cette place, ainsi L ={(e1, Rs-ok)}
Cet enrichissement de marquage est cohérent car il respecte l'invariant de place (2) (la place Rs-ko n'est pas marquée) ce qui veut dire que la ressource ne peut pas être en bon état et défaillante à la fois.
Lc ={(I1, N), (e1, Rs-ok)}, Le = {(e1, Rs-ok)}
Revenir au **(2). Différentes listes transitions;**

(2). Différentes listes transitions
La liste courante Lc contient un jeton dans la place N et un jeton dans la place Rs-ok qui sont initiaux. A partir de ce marquage, il y a trois transitions franchissables en conflit (t1 en conflit avec red et def) d'où :
tfcEsc ={t1, red, def}, Tpf={};
On élimine de ces ensembles la liste des transitions de Lint1, Lint3 et celles de E on obtient :
tfcEsc ={t1, red, def}, Tpf={} d'où:
tfcEsc ={t1, red, def}, Tpfsc = {}, Tpfc ={};

(4). tfcEsc
tfcEsc = {t1, red, def} n'est pas vide, on choisit une transition selon le seuil croissant : une transition stochastique a le même ordre de priorité qu'une transition de seuil inférieur. Dans notre cas *t1* a un seuil inférieur au seuil de *red* tandis que *def* est une transition stochastique donc on peut choisir soit *t1*, soit *def*. On choisit dans un premier temps *t1*. Lint2 ={*red*}, c'est une transition interdite car elle a un seuil supérieur à celui de *t1* et on sait que si le système atteint en premier le seuil *t1* et que celle-ci est franchissable, elle sera franchie et *red* ne sera jamais franchie : Lint2 est réinitialisée pour chaque nouveau contexte, c'est-à-dire les transitions mises dans cette liste ne concérnent que la construction du scénario en cours.
Mémoriser contexte (t2) ;
Lint1 ={t1}, pour ne pas générer à nouveau le même scénario ;
C1(Lc ={(I1, N), (e1, Rs-ok)}, A={}, E={I1}, Le = {(e1, Rs-ok)}, LeInt ={}, Lint1={t1}, Lint2{}, Lint3{}, LjnInit = {}) ;
C={C1} ;
Tirer transition (t1) ;
E={I1, t1}, A={(I1, t1)}, Lc={(t1, N), (t1, Rs-ok)}, LjnInit = {N, Rs-ok} ;
Revenir au **(2). Différentes listes transitions;**

(2). Différentes listes transitions
La liste courante Lc contient un jeton dans la place N, et Rs-ok qui sont normaux mais non initiaux. A partir de ce marquage, il y a trois transitions franchissables t1, red et def d'où :
tfcEsc ={t1, red, def}, Tpf={};
On élimine de ces ensembles la liste des transitions de Lint1, Lint3 et celles de E on obtient :
tfcEsc ={ red, def }, Tpf={} d'où :
tfcEsc ={ red, def }, Tpfsc = {}, Tpfc ={}

(3). Critère d'arrêt
Lc contient contient uniquement des jetons normaux non initiaux ;
Faire **(7).Générer scénario**;

(7).Générer scénario
Le scénario généré est défini par : E={I1, t1}, A={(I1, t1), (t1, F1)}, B = {} ; Lc={(t1, N), (t1, Rs-ok)}, Le={(e1, Rs-ok)}, LjnInit = {N, Rs-ok}.
Faire **(1). Nouvel ordre partiel ;**

Dans la figure 3.8 qui représente le graphe de précédence du scénario 1, I1 et e1 représentent des événements initiaux et F1 est un événement final. Pour être précis e1 est un événement initial d'enrichissement de marquage.

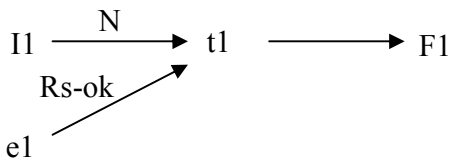


Figure 3.8. Graphe de précédence du scénario1 (reconfiguration)

(1). Nouvel ordre partiel

C n'est pas vide d'où Ci(Lc, A, B, E, Lc, Le, Lint1, Lint2, Lint3, LjnInit) reçoit C1(Lc ={(I1, N), (e1, Rs-ok)}, A={}, B= {}, E={I1}, Le = {(e1, Rs-ok)}, Lint1={t1}, Lint2{}, Lint3{}, LjnInit ={}) ;
C devient vide ;
Faire **(2). Différentes listes transitions;**

(2). Différentes listes transitions

La liste courante Lc contient un jeton dans la place N et un jeton dans la place Rs-ok qui sont des jetons initiaux. A partir de ce marquage, il y a trois transitions franchissables en conflit : t1 en conflit avec red et def d'où :
tfcEsc ={t1, red, def}, Tpf={};
On élimine de ces ensembles la liste des transitions de Lint1, Lint3 et celles de E on obtient :
tfcEsc ={red, def}, Tpf={} d'où :
tfcEsc ={red, def}, Tpfsc = {}, Tpfc = {}.
Faire **(4). tfcEsc ;**

(4). tfcEsc

On choisit une transition selon un seuil croissant, on n'a que deux transitions : une transition stochastique et une transition déterministe donc on peut choisir l'une ou l'autre.
Soit tk = red ;
red dans le réseau est en conflit avec *t1* qui appartient à Lint1 de seuil inférieur et qui est franchissable bien qu'elle soit interdite. Donc on ne peut pas franchir *red*, on prend alors une autre transition qui est *def* d'où :
tk=def ;
def est une transition stochastique, on peut la franchir sans mémoriser le contexte puisque *t1* est interdite.
Tirer transition (def)
E={I1, def}, A={(I1, def)}, Lc={(I1, N), (def, Rs-ko)};
Faire **(2). Différentes listes transitions;**

(2). Différentes listes transitions

La liste courante *Lc* contient un jeton dans la place *N* qui est initial et un jeton non initial dans la place Rs-ko. A partir de ce marquage, il y a une transition franchissable (*red*) en conflit avec une transition potentielement franchissable (*t1*) d'où :
tfcEsc ={red}, Tpf={t1} ;
On élimine de ces ensembles la liste des transitions de Lint1, Lint3 et celles de E on obtient :
tfcEsc ={red}, Tpf={} d'où :
tfcEsc ={red}, Tpfsc = {}, Tpfc = {} ;

Faire **(4). tfcEsc ;**

(4). tfcEsc
Il y a une seule transition dans tfcEsc ={t1}.
Soit tk=red ;
red dans le réseau n'est pas en conflit avec une transition franchissable de seuil inférieur (y compris les transitions de Lint1 et Lint3}donc on peut franchir *red*.
Red peut être franchie car t1 ne peut être franchie à cause de franchissement de def donc on a une relation de causalité indirecte entre def et red : B = {(def, red)} ;
Tirer transition (red);
E={I1, def, red}, A={(e1, def), (I1, red)}, Lc={(def, Rs-ko), (red, E_red), };
Faire **(2). Différentes listes transitions;**

(2). Différentes listes transitions
La liste courante Lc contient un jeton dans la place E_red et un dans Rs-ok qui ne sont pas initiaux. Il n'y a aucune transition franchissable.
Tf ={}, Tpf={} d'où
tfcEsc ={}, Tpfsc = {}, Tpfc = {} ;
Faire **(3). Critère d'arrêt** ;

(3). Critère d'arrêt
Le critère d'arrêt est doublement vérifié : tous les ensembles sont vides et Lc ne contient que des jetons de Ln qui ne sont pas initiaux.
Faire **(7).Générer scénario** ;

(7).Générer scénario
le deuxième scénario généré est défini par : E={I1, def, red}, A={(e1, def), (I1, red), (def, F2), (red, F3)}, B{(def, red)}, Lc={(def, Rs-ko), (red, E_red)}, Le={(e1, Rs-ok), LjnInit ={}} ;
Faire **(1). Nouvel ordre partiel** ;

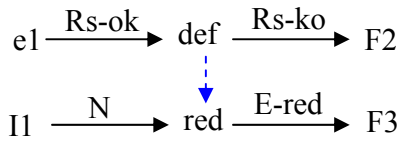


Figure 3.9. Graphe de précédence du scénario2 (redouté)

Dans ce scénario, l'arc en pointiés entre la transition *def* et la transition *red* correspond à une relation de causalité indirecte entre ces deux transitions.

(1). Nouvel ordre partiel
C est pas vide d'où :
Faire le **(8). Pas final**;

(8). Pas final
E1={I1, t1} ;
E2={I1, def, red} ;
Scenario1 :{(I1, t1), (e1, t1), (t1, F1)} ;
Scenario2: {(e1, def), (I1, red), (def, F2), (red, F3), (def, red)}.

3.3.3. Discussion

Le premier graphe de précédence (figure 3.8) représente un scénario de fonctionnement nominal du système en utilisant la ressource Rs-ok qui est en bon état. Le deuxième graphe de précédence (figure 3.9) représente le scénario menant vers l'état redouté. Dans ce dernier on peut voir la prise en compte de l'ordre d'apparition des événements, la transition *red* est franchie car *t1* de seuil inférieur n'est pas franchissable à cause de la défaillance de la ressource (Rs-ko marqué).

La transition red n'est franchie que parce que t1 n'est pas franchissable, c'est-à-dire que def a dû être franchie auparavant. D'où la relation de causalité indirecte entre def et red.

Cet exemple est repris de la thèse de [Khalfaoui 03]. En comparant les résultats obtenus, on voit bien que la prise en compte de la dynamique continue du système permet de prendre en compte l'ordre d'occurence des événements ce qui est indispensable pour la conception des systèmes sûrs de fonctionnement.

3.4. Conclusion

Nous avons présenté dans ce chapitre une méthode d'extraction des scénarios redoutés dans les systèmes hybrides. Cette approche basée sur la logique linéaire et les Réseaux de Petri Prédicats Transitions Différentielles Stochastiques (RdP PTDS) tient compte de la dynamique continue du système représenté par des abstractions temporelles. L'avantage de cette méthode est que l'ordre d'occurrence des événements est pris en compte et les scénarios incohérents vis-à-vis de la dynamique continue sont éliminés. Nous avons illustré les étapes de cette méthode sur un exemple d'un système hybride de petite taille. Pour valider notre méthode, nous l'avons appliqué sur une série d'exemple de taille moyenne comme le système de régulation de niveau d'eau dans les réservoirs [Medjoudj et al 04b] et sur un système industriel (réel) plus complexe de taille importante. Il s'agit d'un système de contrôle des trains d'atterrissage dans un avion qui sera présenté dans le chapitre 5. Nous avons comparé à chaque fois les résultats obtenus avec les résultats de l'ancienne version de l'algorithme qui ne tient pas compte de l'aspect continu du système et le résultat est satisfaisant. Notre méthode génère moins de scénarios en éliminant les scénarios incohérents. Nous avons développé un outil automatisant cette méthode qui sera présenté dans le chapitre suivant.

Chapitre 4 : Mise en œuvre d'un prototype d'outil : ESA_PetriNet

4.1. Introduction

Dans le cadre de l'analyse de systèmes critiques, il est bien sûr nécessaire d'automatiser toutes les étapes de la méthode. Nous présentons dans ce paragraphe l'outil *ESA_PetriNet* (Extraction Scenarios & Analyzer by PetriNet model) qui permet à partir d'un modèle Réseau de Petri d'extraire les scénarios critiques qui mènent vers l'état redouté.

Nous avons choisi Java pour le développement d'*ESA_PetriNet* pour avoir une meilleure portabilité de l'outil. Ainsi, on peut utiliser ce logiciel sur différentes plateformes matérielles et sous divers systèmes d'exploitation.

D'autre part nous avons retenu l'idée d'établir des passerelles entre notre outil et l'outil TINA (Time Petri Net Analyzer) [Berthomieu et al 03]. Bien que cet outil soit dédié aux réseaux de Petri ordinaires et aux réseaux de Petri t-temporels et non à des réseaux de Petri associés à des équations algébro-différentielles, cela présente en effet plusieurs avantages. Tout d'abord il est possible d'utiliser son éditeur graphique pour décrire nos réseaux de Petri. Ensuite, bien que l'abstraction temporelle la plus directe des réseaux de Petri Prédicats Transitions Différentiels Stochastiques soit obtenue sous la forme d'un Réseau de Petri arc-pt-temporel, il est parfois possible, comme nous le verrons dans le premier exemple traité, de le transformer dans un réseau de Petri t-temporel.

4.2. Description générale d'ESA_PetriNet

Dans ce paragraphe, nous utilisons le formalisme UML pour illustrer la mise en ouvre de l'outil. La figure 4.1 représente le diagramme de cas d'utilisation qui modélise les services rendus par le système. Il exprime les interactions utilisateur/système.

Le système offre cinq fonctions principales :

Charger les fichiers d'entrée : les fichiers d'entrée de l'outil correspondent à une description textuelle du réseau de Petri temporel modélisant le système étudié. Dans une première étape nous avons défini notre propre format de description qui nécessite un travail manuel fastidieux pour la génération des fichiers texte du réseau de Petri correspondant. Etant donné que cette étape est importante pour la suite du traitement et afin de réduire les risques d'erreur, il fallait automatiser la génération de ces fichiers. La version actuelle d'ESA_PetriNet considère les fichiers de sortie de TINA (Time Petri Net Analyzer) version 2.7.4 qui grâce à son interface graphique facilite considérablement la modélisation de systèmes représentés par des réseaux de Petri. Il faut noter qu'il est facile de maintenir l'outil pour l'interfacer avec d'autres logiciels permettant de générer des descriptions textuelles de réseaux de Petri. En effet il suffit de changer une seule fonction du

système. Lors de la vérification des contraintes temporelles un fichier supplémentaire contenant les contraintes temporelles est introduit.

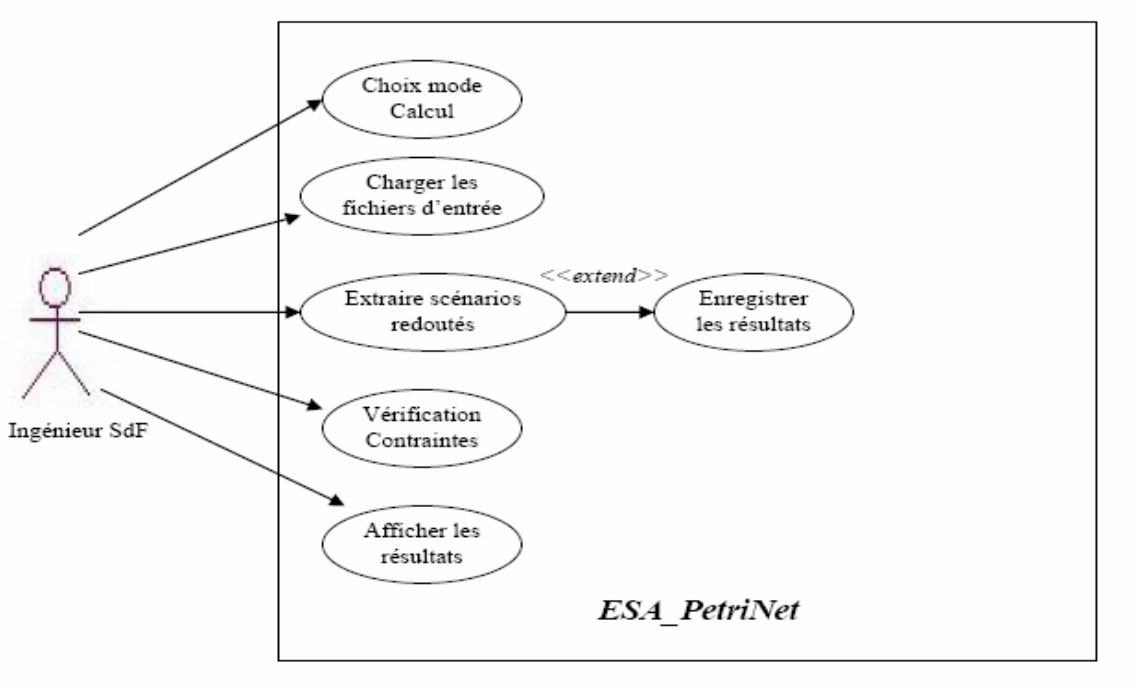


Figure 4.1. Diagramme de fonctionnement d'ESA_PetriNet

Extraire scénarios redoutés : c'est la fonction principale du système. Après analyse des fichiers d'entrée, l'outil extrait : le nombre de transitions, nombre de places, nombre de jetons normaux, nombre d'invariants, matrice d'invariants, matrice des places constituant chaque invariant, matrice des jetons normaux, la matrice d'incidence avant, la matrice d'incidence arrière, matrice de marquage et matrice de seuil et dans certains exemples la liste des transitions correspondant à des commandes ou interdites dès le début après une première analyse du système. Il est à noter que les matrices d'incidence correspondant au réseau de Petri inverse pour le raisonnement arrière sont construites ultérieurement. Pour la génération des scénarios redoutés nous avons besoin de classes correspondant aux structures de données décrites dans le paragraphe 3.2.3 et nécessaires au déroulement de notre algorithme à savoir : Ci (Lc, Lint1, Lint2, Lint3, E, A, B, Le, LeInt, LjnInit) et C= {C0, C1, …Ci….Cn}. Dans la suite nous présentons le diagramme de classes d'ESA_PetriNet ainsi qu'une description des méthodes principales de chaque classe. Selon le but de l'étude (vérification ou fiabilité) la génération des scénarios va se faire par un raisonnement arrière uniquement si le mode de calcul choisi est 1 (pour la vérification), ou par un raisonnement arrière et avant si le mode choisi est 2 (pour la fiabilité).

Enregistrer les résultats : les résultats correspondent à la valeur finale de C contenant autant d'éléments Ci que de scénarios. Nous considérons tous les scénarios (fonctionnement normal, reconfigurations et scénarios redoutés) car le concepteur a besoin d'avoir des informations précises sur la dynamique du système. Le fichier résultats contient la liste des scénarios générés et permet de dessiner le graphe de précédence. La figure 4.2 représente le fichier résultat généré par ESA_PetriNet pour le cas d'étude illustré dans le chapitre 3. Nous obtenons les deux scénarios correspondant aux figures 3.8 et 3.9.

- la première ligne indique le nombre de scénarios, dans cet exemple il y a 2 scénarios;

- la deuxième ligne correspondant au nombre de transitions, dans cet exemple il y a 4 transitions;
- `I2&def I1&red def&F99 red&F100 def#red :` correspond au première scénario redouté : (`I1` et `I2`) sont des événements initiaux, (`F99` et `F100`) sont des événements finaux, (`I2&def`) relation de précédence directe entre l'événement initial `I2` et la transition `def, (def#red)` relation de causalité indirecte entre les transitions `def,` et `red.`
- `0 :` correspond au numéro du scénario redouté dans la liste des scénarios

Afficher les résultats : nous avons choisi d'afficher les scénarios sous forme d'un graphe de précédence. Les relations de causalité indirecte étant affichées dans une couleur différente.

```
2
4
I2&def I1&red def&F99 red&F100 def#red
I1&t1 I2&t1 t1&F99
0
```

Figure 4.2. f-resultat.txt généré par ESA_PetriNet

La figure 4.3 illustre le diagramme de classes correspondant à *ESA_PetriNet*. Dans ce paragraphe, nous décrivons les méthodes et les attributs des classes principales.

`trans_place`: cette classe modélise un couple transition-place `(ti, pi)` permettant de construire des éléments de type Lc et Le.

`trans_trans`: cette classe modélise un couple transition-transition `(ti, tj)` permettant de construire des éléments de type A et B.

`liste_trans`: cette classe implémente la structure de données E, Lint1, Lint2 et Lint3.

et possède des méthodes permettant d'ajouter ou de supprimer un élément de cette liste.

`liste_trans_place`: cette classe implémente la structure de données Lc et Le. Elle possède des méthodes permettant d'ajouter ou de supprimer un élément de cette liste.

`liste_trans_trans`: cette classe implémente la structure de données A et B. Elle possède des méthodes permettant d'ajouter ou de supprimer un élément de cette liste.

`liste_jeton`: cette classe implémente la structure de données LjnInit et possède des méthodes permettant d'ajouter ou de supprimer un élément de cette liste.

`Ci_element`: cette classe implémente la structure de données Ci.

`C_element`: cette classe implémente la structure de données C et possède des méthodes permettant d'ajouter ou de supprimer un élément de cette liste.

`rdpContinu`: cette classe implémente notre algorithme. Elle possède une trentaine de méthodes et correspond au programme principal. Nous avons choisi ici les méthodes principales.

Extraire_rdp: cette méthode récupère les fichiers d'entrée générés par TINA et construit la matrice de marquage, matrice des jetons normaux, matrice d'invariant, matrice de seuil matrice d'incidence arrière, matrice d'incidence avant, matrice d'invariant et la matrice des places constituant les invariants. Ces paramètres correspondent au réseau de Petri initial, un traitement complémentaire permet de déduire les paramètres du réseau de Petri inversé. Il faut noter que ce traitement est spécifique au format généré par TINA, un effort d'adaptation est nécessaire pour l'interfaçage avec d'autres outils.

Pas_initial_avant (arrière) : cette méthode initialise le contexte C avec un élément Ci pour générer le premier ordre partiel.

Ltf_Ltpf : cette méthode construit la liste des transitions franchissables et la liste des transitions potentiellement franchissables.

Différente_liste_trans: dans un contexte Ci, cette méthode permet de construire les ensembles tfcEsc, Tpfsc et Tpfc.

Generer_scenario: cette méthode permet de générer le scénario du contexte Ci et permet de mémoriser le résultat de traitement de contexte Ci.

Pas_final : cette méthode permet de mémoriser les résultats de traitement de tous les contextes.

Tirer_trans_tk : cette méthode franchit une transition `tk` en mettant à jour les éléments E, A et Lc d'un contexte Ci.

Coherence_marquage1, Coherence_marquage2: pour une transition `tk` cette méthode considère la liste courante des jetons et vérifie si l'enrichissement de marquage est cohérent. Dans le premier, si il y a conflit d'enrichissement, on mémorise un contexte d'enrichissement de marquage.

Enrichir_marquage1_tk, Enrichir_marquage2_tk : considérant un contexte Ci et une transition `tk`, ces deux méthodes effectuent l'enrichissement du marquage correspondant. Selon le cas, cet enrichissement peut se faire selon la première méthode si tk est franchissable en conflit avec une (ou plusieurs) transition(s) potentiellement franchissable(s) ou selon la deuxième méthode si `tk` est potentiellement franchissable.

Memoriser_contexte_tk : cette méthode permet de créer un nouveau contexte Ci si `tk` vérifie les conditions stipulées dans les étapes (4) et (5) de l'algorithme (§ 3.2.3.3).

Memoriser_contexte_enrich_pk : cette méthode permet de créer un nouveau contexte Ci si pk vérifie les conditions stipulées dans les étapes (4) et (5) de l'algorithme (§ 3.2.3.3).

Conflit_tk_tj : dans un contexte Ci, cette méthode vérifie si la transition `tk` est en conflit avec la transition `tj`. On fait appel à cette méthode à chaque fois qu'on veut franchir une transition.

Conflit_tk_liste: *v*erifie si tk est n'est pas en conflit avec un élement d'une liste.

Conflit_trans_tk_seuil : verifie si des transitions en conflit avec tk de seuil inférieur sont franchissables, les relations de causalité indirecte sont construites dans cette méthode.

Conflit_trans_tk : cette methode verifie si tk est en conflit avec une transition franchissable

`Seuil_croissant` : selon l'étape courante de l'algorithme et pour un contexte Ci, cette méthode ordonne les éléments de l'une des listes : tfcEsc, Tpfsc ou Tpfc selon le seuil croissant.

`VerificationContraintes`: cette méthode récupère les contraintes temporelles (`[d_min, d_max]`) dans un fichier textuel donné par l'utilisateur. Elle parcourt la liste des scénarios et pour chaque scénario elle calcule les durées minimale (`d_min_cmd`) et maximale (`d_max_cmd`) entre deux commandes successives et vérifie le respect des contraintes temporelles (`d_min_cmd>=d_min`, et `d_max_cmd<=d_max`). Le traitement s'arrête à la violation d'une contrainte dans un scénario c'est à dire quand l'intervalle de temps entre deux commandes n'est pas inclus dans `[d_min, d_max]` ou après la vérification de tous les scénarios. Un contre-exemple est affiché à l'écran dans le cas de la violation d'une contrainte.

`dessinGraphe:` cette classe permet d'implémenter l'interface graphique de l'outil. En effet, elle récupère les résultats enregistrés dans un fichier textuel (`f-result.txt`) et les affiche sous forme d'un graphe de précédence. En ce qui concerne la vérification on affiche le contre-exemple trouvé ou un message confirmant la vérification des contraintes dans tous les scénarios.

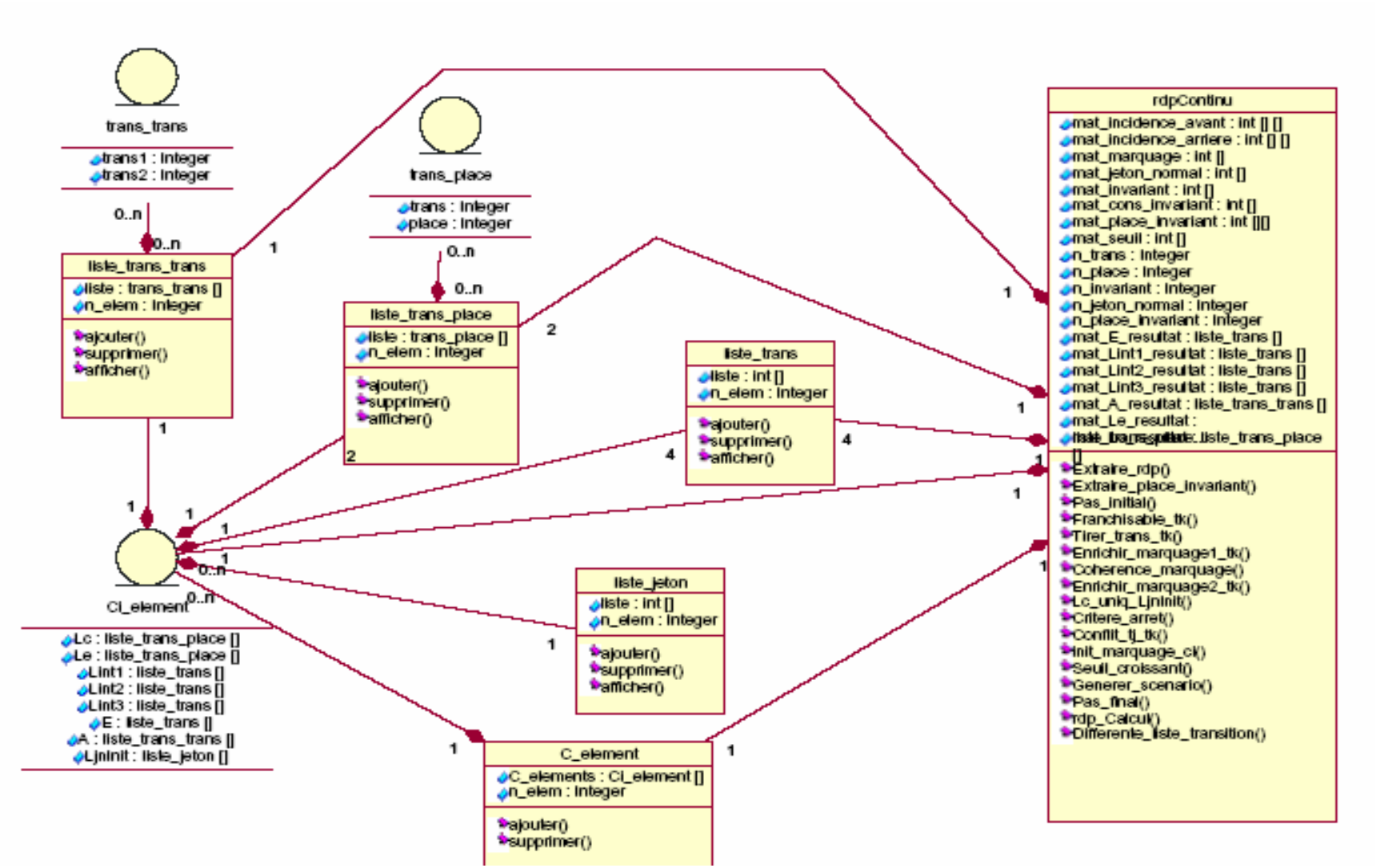


Figure 4.3. Diagramme des classes généré par UML (Rational Rose)

4.3. L'outil TINA

TINA (Time Petri net Analyzer), développé au LAAS-CNRS au sein du groupe OLC, est une boîte à outils pour l'édition et l'analyse des réseaux de Petri et des réseaux de Petri t-temporels.

Il comprend le module "nd" (net draw) qui permet, en particulier, l'édition graphique et textuelle des réseaux de Petri ordinaires et des réseaux de Petri t-temporels. Ce module nous concerne directement. Il est ainsi possible, à partir d'une description graphique d'obtenir un fichier textuel pouvant être utilisé par les autres modules de TINA, mais également par d'autres outils. Nous l'utiliserons donc pour ESA_PetriNet. Une caractéristique très intéressante pour nous est qu'il est possible d'associer des étiquettes aux places et aux transitions dans l'éditeur graphique et de les récupérer dans le fichier textuel.

Le module "tina" permet de construire le graphe d'accessibilité d'un réseau de Petri ordinaire borné, ou de détecter qu'il n'est pas borné. Il contient également des outils permettant d'exploiter le parallélisme pour construire des graphes d'accessibilité réduits lorsque ces graphes sont suffisants pour montrer la propriété recherchée. Ces outils ne concernent pas directement nos travaux et nous ne les détaillerons donc pas. Plus proche de nos préoccupations est l'outil permettant la construction d'un graphe des classes (une classe est une représentation abstraite d'un ensemble infini d'états qui sont tous équivalents vis-à-vis de l'accessibilité) d'un réseau de Petri t-temporel. Dans une phase ultérieure de notre travail, il pourrait être utilisé pour comparer nos résultats concernant l'analyse temporelle des scénarios critiques avec ceux de TINA, en particulier lorsque l'abstraction temporelle du réseaux de Petri Prédicats Transitions Différentiels Stochastiques représentant le système étudié peut se ramener à un réseau de Petri t-temporel.

Enfin le module "struct" permet le calcul des invariants de places et de transitions. Ce module est important pour nous puisque nous utilisons les invariants de places pour vérifier la cohérence des enrichissements de marquage.

4.4. Interfaçage entre TINA et ESA_PetriNet

J'ai choisi d'utiliser TINA pour générer deux fichiers textuels du modèle RdP de notre système à partir d'une description graphique de réseau de Petri éditée sur NetDraw (figure 4.4). Les étiquettes (label) sont utilisées pour le marquage initial du réseau global et pour définir les places normales « N ». Le marquage initial du réseau sous TINA correspond, lui au marquage initial du scénario.

Un fichier `f.net` (figure 4.5) qui contient la liste des places avec leur marquage et leur label, la liste des transitions avec leurs intervalles temporels statiques et la description de la matrice d'invariants. Dans cet exemple on n'a pas associé des labels aux transitions.

`tr t0 [0,0] OFF -> N :` la transition t0 a un intervalle temporel statique de franchissement de `[0,0]`et possède une place d'entrée `OFF` et une place de sortie `N.`

`tr def [0,w[ {Rs-ok} -> {Rs-ko}:` la transition stochastique `def` a un intervalle temporel de franchissement de `[0,w[` qui correspond à `[0,∞[` ce que signifie que cette transition peut être franchie à tout moment et possède une place d'entrée `Rs-ok` et une place de sortie `Rs-ko.`

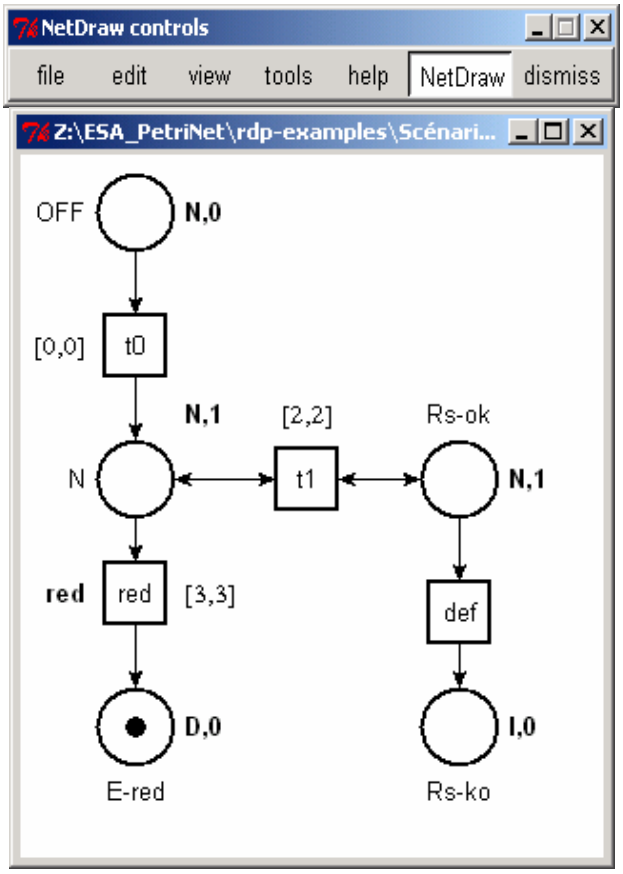


Figure 4.4. f.ndr correspondant au graphe RdP du cas d'étude du chapitre 3

```
tr t0 [0,0] OFF -> N
tr red [3,3] N -> {E-red}
lb red red
tr def [0,w[ {Rs-ok} -> {Rs-ko}
tr t1 [2,2] {Rs-ok} N -> {Rs-ok} N
lb OFF {N,0}
lb N {N,1}
pl {E-red} (1)
lb {E-red} {D,0}
lb {Rs-ok} {N,1}
lb {Rs-ko} {I,0}
net Cas etude
```

Figure 4.5. f.net correspondant au cas d'étude du chapitre 3

`lb OFF {N,0} :` la place `OFF` possède le label `{N, 0},` N pour place normale et 0 pour construire les invariants. En effet pour calculer les invariants TINA nécessite d'avoir un

marquage initial complet de tout le réseau alors que dans notre cas on ne possède qu'un marquage partiel qui sera enrichi ultérieurement. Pour pallier ce problème, nous avons choisi d'utiliser des labels incluant des paramètres permettant le calcul de la composante conservative de chaque invariant.

`lb {E-red} {D,0}:`La place `E-red` possède le label`{D,0},` D pour defaillance et 0 pour construir les invariant. Dans le cas de recherche de scénarios pour la vérification (mode calcul =1), le label de toutes les place est de D pour ne pas arrêter très tôt car le N est utilisé comme critère d'arrêt.

`pl {E-red} (1):` il y a une seule place marquée dans cet exemple qui est `E-red`. Son marquage est de 1.

`lb red red`: Les transitions aussi ont des label, le label red pour differentier la transition menant vers l'état redouté des autres transitions, celà nous permet d'extraire la liste des scénarios redoutés parmi tous les scénarios générés pour les afficher avec une autre couleure. Dans cet exemple c'est la transition `red` qui possède le label `red`. On a également un autre type de label qu'on ne voi pas dans cet exemple, c'est le label `F` qu'on associe à certaines transitions qu'on ne veut pas franchir pour éviter les boucles par exemple. Ces transitions (labelées avec `F`) sont mémorisées dans la liste des transitions interdites `LintEntree.`

```
Struct version 2.7.4 -- 05/02/05 -- LAAS/CNRS

parsed net Cas_etude

5 places, 4 transitions

net Cas_etude
tr def {Rs-ok} -> {Rs-ko}
tr red [3,3] N -> {E-red}
tr t0 [0,0] OFF -> N
tr t1 [2,2] N {Rs-ok} -> N {Rs-ok}
pl {E-red} (1)

0.000s

P-SEMI-FLOWS GENERATING SET ------------------------------------

invariant

{Rs-ko} {Rs-ok}
N OFF {E-red}

0.000s

T-SEMI-FLOWS GENERATING SET ------------------------------------

not consistent

t1

0.000s

ANALYSIS COMPLETED  ----------------------------------------------
```

Figure 4.6. f-struct.txt permettant la récupération des invariants

Un fichier f-struct.txt (figure 4.6) qui est généré en appliquant une analyse structurelle du réseau de Petri. On ne récupère dans ce fichier que les informations relatives aux invariants, par exemple :

`{Rs-ko} {Rs-ok} : C`e réseau de Petri contient deux invariants de place dont le premier est constitué des places `Rs-ko` et `Rs-ok et le deuxième est constitué des places N, OFF et E-red.`

4.5. Récupération des informations nécessaires par ESA_PetriNet

`Extraire_rdp (f.net)` : cette méthode extrait le fichier décrivant textuellement le rdp ;

- Récupérer les lignes commençant par t (pour transition) : construire la liste des transitions, le nombre de transitions, matrice des intervalles temporels, la matrice d'incidence avant et la matrice d'incidence arrière.
- récupérer les lignes commençant par p (pour place) : construire la matrice de marquage.
- récupérer les lignes commençant par l (pour label) : construire la matrice des jetons normaux et les labels de construction de la composante conservative des invariants. Les labels peuvent aussi être associé à des transitions pour identifier les commandes (cmd) dans le cas de l'exemple des calculateurs de vol (chapitre 6) ou les transitions qu'on ne veut pas franchir (F) dans le cas de l'exemple des trains d'atterrissage (chapitre 5).

`Extraire_place_invariant (f-struct.txt)` : cette méthode parcourt tout le fichier pour extraire les listes de places constituant les invariants. Le nombre d'invariants étant le nombre de listes.

- construire la matrice de places d'invariant
- construire la matrice des invariants.

Ces différentes matrices représentent les données d'entrée d'ESA_PetriNet.

4.6. Utilisation d'ESA_PetriNet

Pour utiliser ESA_PetriNet, on commence par l'édition du rdp du système à étudier sur l'interface graphique de TINA. Le fichier est enregistré sous `(f.ndr)` comme le présente la figure 4.4. On génère ensuite le fichier `f.net` (figure 4.5) et le fichier `f-struct.txt` (figure 4.6) et on les enregistre. Ces deux fichiers sont les fichiers d'entrée d'ESA_PetriNet. Rappelons que dans le cas d'une vérification il faut ajouter le fichier contenant les contraintes à vérifier.

ESA_PetriNet est équipé d'une interface graphique qui facilite son utilisation. Ce premier prototype est constitué de cinq champs d'entrée, de trois boutons et d'un champ graphique comme le montre la figure 4.7.

- les cinq champs correspondent respectivement aux fichiers : `f.net`, `f-struct.txt`, `f-result.txt`, `f-contraintes.txt` contenant les contraintes et le choix de mode de calcul.
- Le bouton « `Générer scénario` » permet de générer et d'enregistrer les scénarios dans un format textuel dans le fichier `f-result.txt.`
- Le bouton « `Graphe de précédence` » permet d'afficher les scénarios sous la forme de graphes de précédence.
- Le bouton « `Vérification` » permet de vérifier les contraintes dans la liste des scénarios générés précédemment.

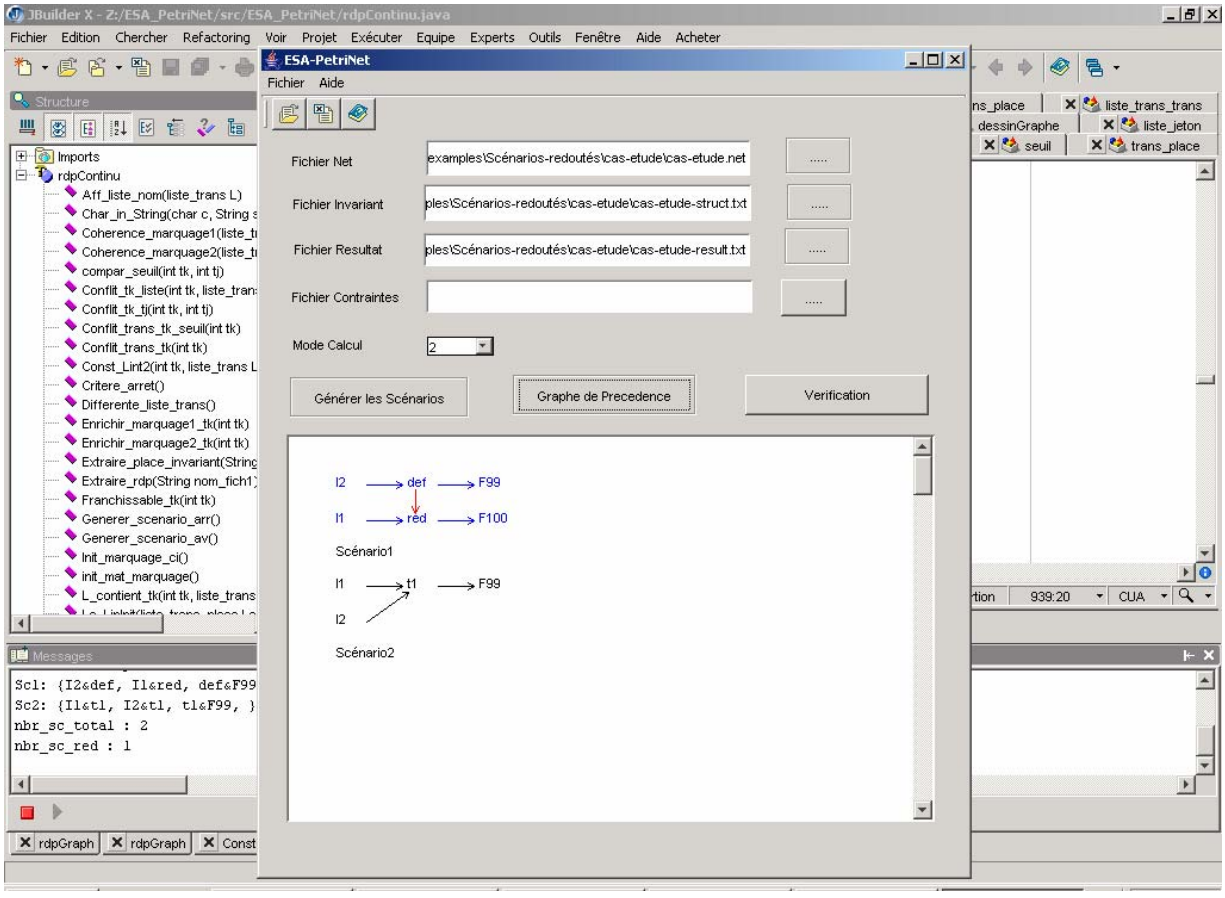


Figure 4.7. Interface graphique d'ESA_PetriNet : cas d'étude 3.5

4.7. Exemple et comparaison avec l'ancienne version de l'algorithme

Si l'on applique l'ancienne version de l'algorithme sur le cas d'étude du chapitre 3 c'est-à-dire sans utiliser l'information concernant les seuils, on obtient trois scénarios (figure 4.8) au lieu de deux scénarios (figure 4.7). Le scénario 1 généré par l'ancienne version de l'algorithme est incohérent vis-à-vis la dynamique continue du système et l'ordre d'occurrence des événements n'est pas pris en compte puisque la transition `def` doit être franchie avant la transition `red`. Le point fort de la nouvelle version est la prise en compte de l'ordre d'occurrence des événements et génère moins de scénarios.

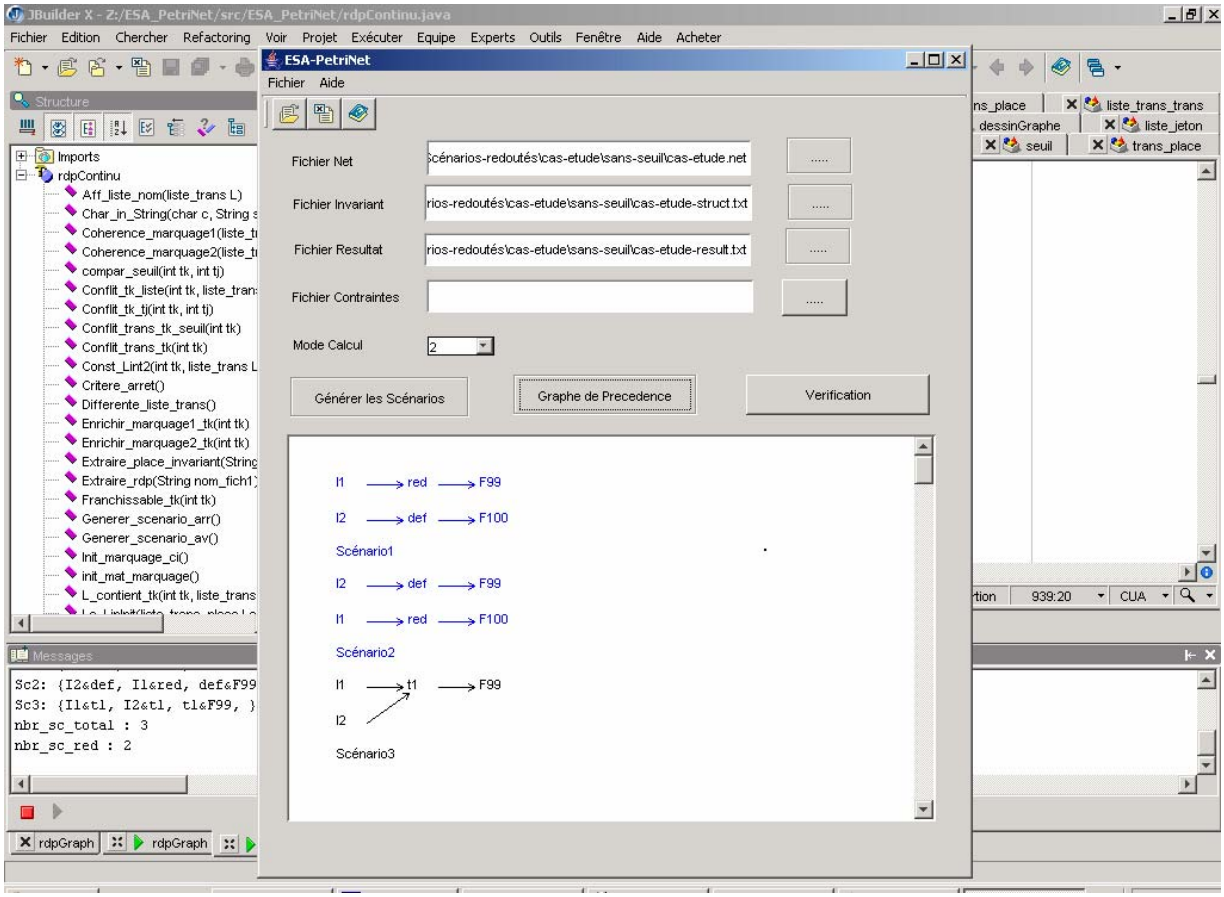


Figure 4.8. Scénarios générés sans tenir compte des seuils : cas d'étude 3.5

4.8. Conclusion

Dans ce chapitre nous avons présenté un premier prototype de notre outil ESA_PetriNet « Extraction & Scenarios Analyzer by PetriNet model » qui permet à partir d'un modèle réseau de Petri l'extraction des scénarios redoutés en tenant compte de l'aspect continu du système. Cet outil permet aussi de vérifier certaines propriétés du système comme la durée d'un scénario ou l'accessibilité entre deux états. Nous avons présenté l'interfaçage de notre outil avec TINA «TImed petri Net Analyser » et comment l'utiliser. Nous l'avons appliqué sur le cas d'étude du

chapitre 3 puis nous avons comparé les résultats obtenus avec les résultats de l'ancienne version de l'algorithme qui ne tient pas compte de l'aspect continu du système. La nouvelle version génère moins de scénarios et prend en compte dans certains cas l'ordre d'occurrence des événements imposé par la dynamique continue. Les scénarios générés sont ainsi cohérents avec la dynamique continue du système. La durée de calcul est de quelques secondes. Dans le chapitre suivant nous allons utiliser notre outil pour la chercher les scénarios redoutés dans un système de contrôle des trains d'atterrissage dans un avion qui est un exemple industriel réel de taille importante. Notre outil sera également appliqué pour la vérification de certaines propriétés temporelles dans le chapitre 6.

Chapitre 5 : Un exemple de recherche de scénarios redoutés

5.1. Introduction

Nous avons choisi comme exemple, pour illustrer notre méthode un système d'atterrissage. Dans ce chapitre, le scénario implique à la fois le système de commande et le système commandé avec les différents modes de fonctionnement: nominal, reconfiguration et défaillance.

L'application de notre méthode nécessite la modélisation du système par un réseau de Petri Prédicats Transitions Différentiel Stochastique. Une analyse préliminaire permettra d'affiner les domaines des variables suivant les divers marquages accessibles en raisonnant sur les invariants de places, les équations différentielles associées aux places et les seuils associés aux transitions. Les invariants de places servent à déterminer quelles sont les dynamiques possibles, quelles autres places peuvent être simultanément marquées, quand un jeton est présent dans une place donnée. Ensuite une abstraction temporelle est faite en associant aux transitions des intervalles temporels de tir correspondant au temps que peut mettre le système pour atteindre l'état en question.

5.2. Présentation de l'exemple

5.2.1. Description générale

Il s'agit d'un système de contrôle des trains d'atterrissage d'un avion de type *Rafale* de *Dassault Aviation.* Ce système est une version modifiée et simplifiée d'un étalon (benchmark) proposé par le groupe de travail français Systèmes Temps Réel et Qualité de Service [STRQDS 02], qui a été présenté par [Boniol et Carcenac 02] et étudié par [Villani et al 03] dans sa version hybride. Le but est d'appliquer notre approche pour extraire les scénarios redoutés.

Ce système est composé de trois trains d'atterrissage (un train avant, un train gauche et un train droit) qui doivent être en position rentrée pour le vol (rapide) et en position sortie pour l'atterrissage. Ces trois trains sont interdépendants. Chaque train comporte (figure 5.1):

- un boîtier d'accrochage du train en position rentrée.
- une trappe (porte) qui s'ouvre avant la sortie ou la rentrée d'un train et se ferme automatiquement à l'achèvement du mouvement.

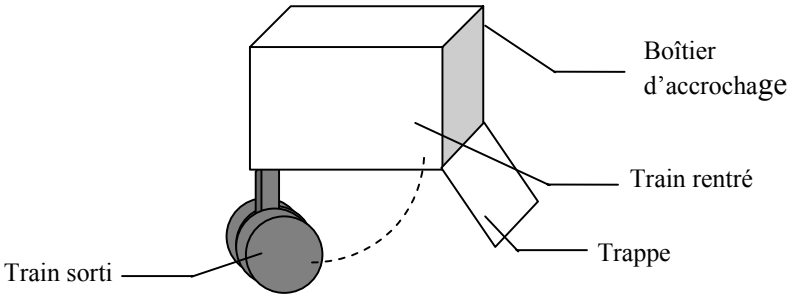


Figure 5.1. Train d'atterrissage *i*

5.2.2. Fonctionnement du système

Le contrôle des trains est assuré par trois commandes *E*, *R* et *B* :

- la commande *E* réalise en séquence l'ouverture des portes, l'extension des roues et la fermeture des portes. Cette séquence est présentée sur la figure 5.2
- la commande *R* réalise en séquence l'ouverture des portes, la rentrée des roues et la fermeture des portes.
- la commande *B* est intermédiaire, dans ce cas les roues sont bloquées dans leur position courante.

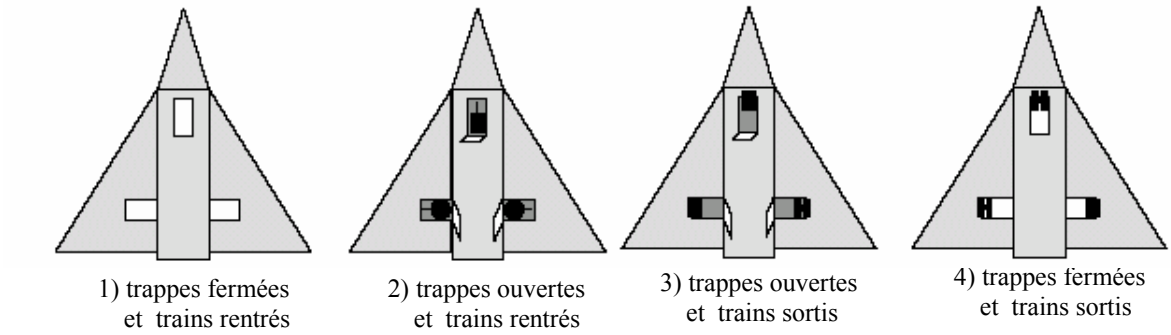


Figure 5.2. Séquence de la sortie d'un train *i*

5.2.3. Evénements redoutés

Nous nous intéressons uniquement aux événements redoutés liés à la sortie du train. Nous avons trois événements redoutés pour chaque train :

- Non-ouverture de la trappe pour la sortie du train,
- Non-sortie du train,
- Non-fermeture de la trappe après la sortie du train.

Les événements redoutés liés à la rentrée des trains peuvent être obtenus d'une façon similaire. Dans cette partie on ne tient pas compte des événements redoutés liés à la commande B. Nous ne décrivons pas la commande *B* ainsi que les événements redoutés qui lui sont associés.

5.2.4. Composition du système

Le système est constitué d'un calculateur qui commande trois ensembles train/ trappe. En fonction de la position des trains, trappes (fournie par les capteurs) et de l'état de l'avion le calculateur établit la commande à réaliser sur le système via les vérins hydrauliques. La figure 5.3 représente le schéma simplifié des interactions entre les différents éléments des trains d'atterrissage.

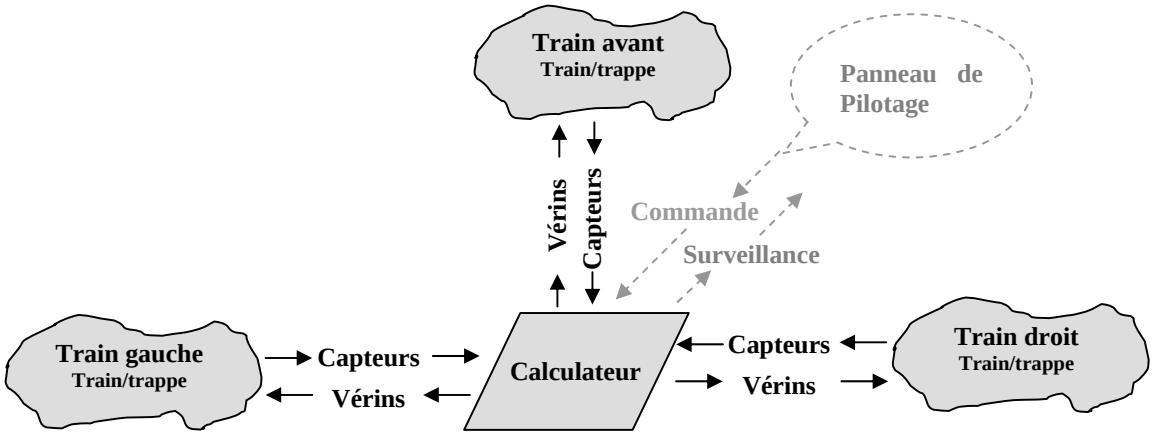


Figure 5.3. Système de contrôle des trains d'atterrissage

Un schéma incluant les différents composants du système de contrôle des trains d'atterrissage est donné en figure 5.4. Il comporte :

- **Un calculateur** qui émet la commande E pour la sortie des trains, la commande R pour la rentrée des trains et une commande de secours qui ne permet que la descente des trains en cas de blocage en sortie. Pour ordonner la sortie ou la rentrée des trains, le pilote dispose dans la cabine d'une palette UP/DOWN de commande à deux positions UP (commande E pour la sortie des trains) et DOWN (commande R pour la rentrée des trains).
- **Trois ensembles train/ trappe** actionnés par des vérins hydrauliques. Chaque vérin d'une trappe est commandé par une électrovanne d'ouverture qui déclenche l'ouverture de la trappe et une électrovanne de fermeture qui déclenche sa fermeture. Chaque trappe ferme un logement d'un train. Chaque vérin d'un train est commandé par une électrovanne de montée qui déclenche la montée d'un train et une électrovanne de descente pour la sortie du train. Les vérins des trains ont deux fonctionnalités : entrée/sortie et verrouillage en position basse.

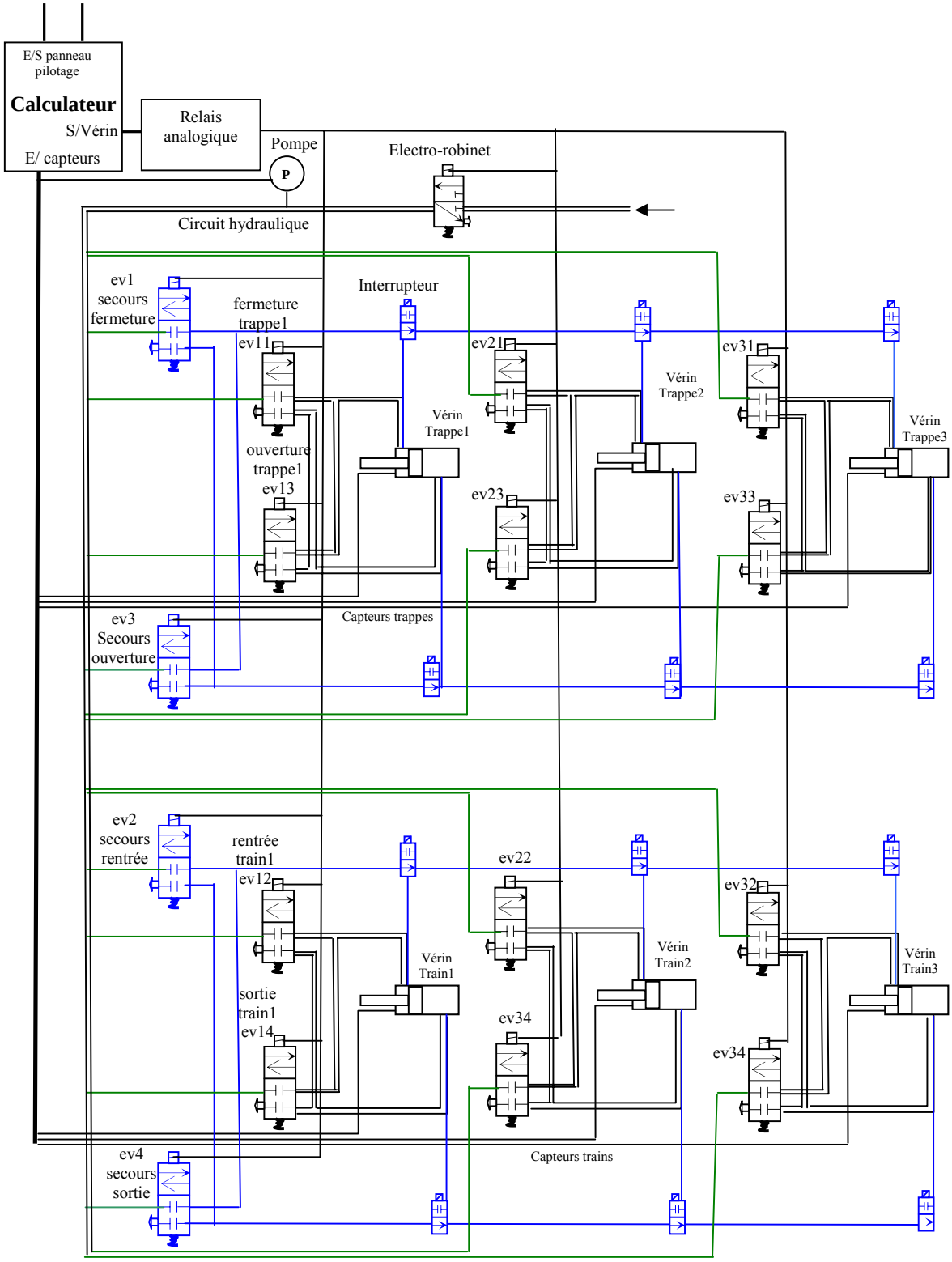


Figure 5.4. Composants du système des trains d'atterrissage

- **Quatre électrovannes de secours** pour débloquer un train d'atterrissage en sortie en cas de défaillance de l'une des électrovannes spécifiées (blocage de vérin). La première électrovanne de secours remplace une des trois électrovannes d'ouverture des trois trappes. La deuxième électrovanne de secours remplace une des trois électrovannes qui commande la sortie des trains. La troisième électrovanne remplace une des trois électrovannes de fermeture des trappes et la quatrième intervient en cas de blocage en rétraction des trains. L'électrovanne de secours ne peut commander qu'un vérin à la fois. Les interrupteurs sur la figure 5.4 correspondent à des électrovannes qui peuvent être ouverte ou fermée selon la commande de calculateur. Le calculateur commande l'ouverture d'une électrovanne (interrupteur) pour alimenter le vérin d'une trappe i et commande la fermeture des autres pour que l'électrovanne de secours sera utilisé par une seule trappe à la fois (donner le maximum de pression).
- **Une pompe et un électro-robinet** fournissent l'énergie hydraulique au circuit.
- **Des capteurs de position** indiquant la position des trains et des trappes.
- **Un relais analogique** isolant le calculateur des électrovannes. Le relais reste ouvert un certain temps après le dernier changement de la commande (palette UP/DOWN). Ce temps est supposé suffisant pour la sortie ou la rentrée des trains d'atterrissage. Quand le pilote manœuvre à nouveau la palette UP/DOWN, le relais se ferme. Au sol, il permet l'interdiction de l'excitation de l'électro-robinet par le calculateur et donc la mise en pression du circuit hydraulique pour qu'un mécanicien puisse intervenir sur l'atterrisseur.

5.2.5. Modélisation du système

Un modèle global de réseau de Petri de ce système est représenté sur la figure 5.5. La place p_1 représente l'état dans lequel les trois trains sont rentrés. La transition t_1 est franchie quand la commande *E* est réalisée. Cet événement génère trois branches parallèles correspondant aux trois trains. La transition t_2 est franchie quand les trois trains sont sortis. La partie droite de la figure correspond à la commande de rentrée. Notons que le bloc entre pi0 et pi4 correspond à un modèle réseau de Petri.

5.2.6. Modélisation de la sortie des trains d'atterrissage avec défaillance

Pour simplifier le modèle, on suppose que le calculateur, la pompe, l'électro-robinet et les capteurs assurent leur fonction et ne sont pas défaillants. Ils ne seront donc pas modélisés. Dans cet exemple nous considérons uniquement les défaillances des électrovannes qui commandent les différents vérins des trappes et des trains.

La figure 5.6 représente un modèle réseau de Petri simplifié de la sortie des trains d'atterrissage. Notons que les places ps_{i0} et p_{i4} sont les mêmes que celles représentées sur la figure 5.5. Le modèle contient trois réseaux identiques qui se succèdent : 1) ouverture des trois trappes, 2) sortie des trois trains, 3) fermeture des trois trappes.

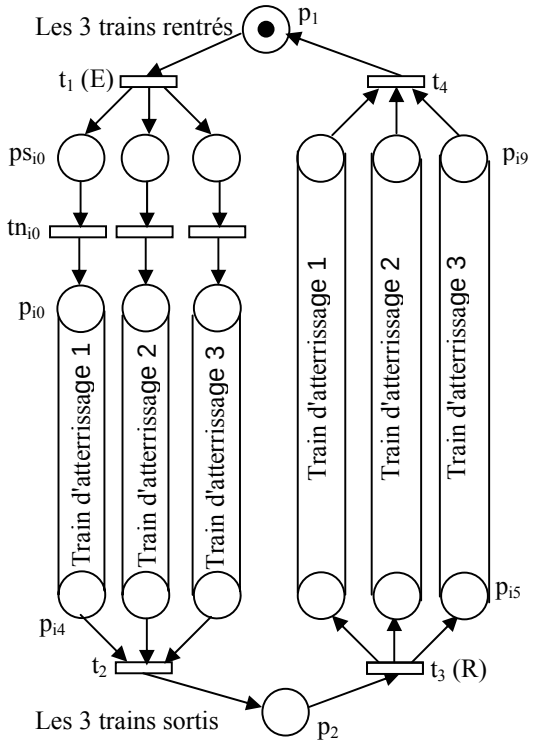


Figure 5.5. Vue générale du RdP du système d'atterrissage

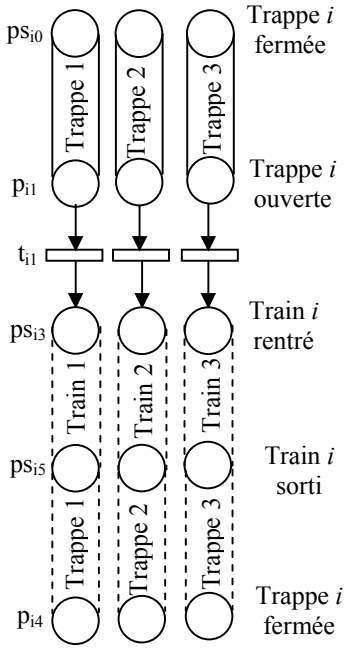


Figure 5.6. Vue générale du RdP de la sortie des trains d'atterrissage

Nous allons chercher les scénarios redoutés d'une façon modulaire. Comme nous l'avons mentionné dans le paragraphe 5.2.3, l'apparition d'un état redouté est la conséquence de l'un des événements suivants : 1) non-ouverture d'une trappe *i* pour la sortie d'un train *i*, 2) non-sortie d'un train *i*, 3) non-fermeture d'une trappe *i* après la sortie d'un train *i.*

Nous allons chercher les scénarios redoutés correspondant à la non-ouverture d'une trappe *i* pour la sortie d'un train *i*. Nous considérons donc le premier réseau correspondant à l'ouverture des trois trappes. Ces trois trappes sont interdépendantes car elles utilisent la même électrovanne de secours qui ne peut être utilisée que par une trappe à la fois.

5.2.6.1 Modèle réseau de Petri de l'ouverture d'une trappe

Comme nous l'avons mentionné, la position d'une trappe est déterminée par la position du vérin hydraulique correspondant. La figure 5.7 représente les différents états de la trappe. Quand la trappe est fermée et la pression P_B est supérieure à la pression P_A, la porte s'ouvre avec une vitesse constante, x étant la position de la porte, y la largeur de la tête du piston et (l-x) la position de la porte ouverte. La porte s'ouvre dans le cas inverse et se bloque en cas d'égalité de pression entre les deux entrées du vérin, ce qui signifie que l'électrovanne qui l'alimente en pression est défaillante.

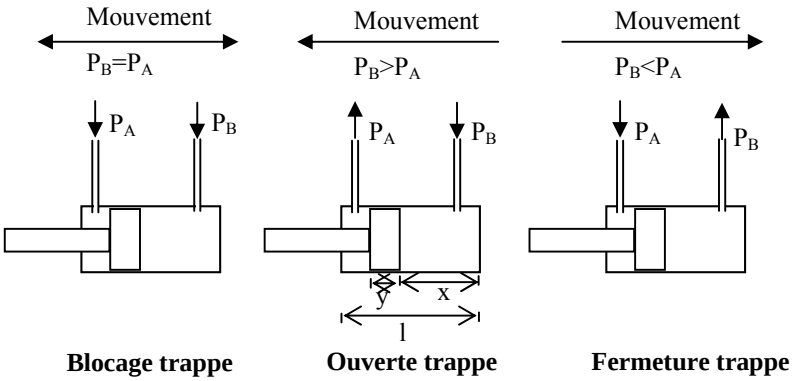


Figure 5.7 Différents états d'une trappe i

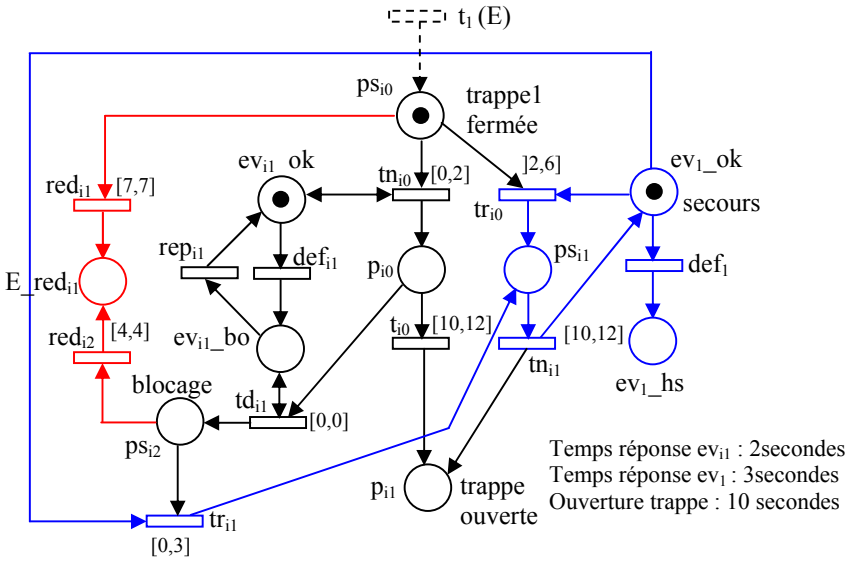


Figure 5.8. Modèle RdP de l'ouverture d'une trappe *i*

Dans le réseau de Petri de la figure 5.8, la transition tn_{i0} représente la demande d'ouverture de l'électrovanne ev_{i1}. Cette électrovanne peut être en bon fonctionnement (marquage de la place ev_{i1}_ok) ou défaillante par le franchissement de la transition def_{i1}. Si ev_{i1} est en bon fonctionnement, la transition tn_{i0} est franchie et la place p_{i0} est marquée. La place p_{i0} correspond à l'ouverture de la porte de la trappe. Sa dynamique continue est prise en compte par une abstraction temporelle. La durée minimale pour l'ouverture de la trappe d'un train *i* est 10 secondes et sa durée maximale est 12 secondes. Le temps correspondant à la contrainte temporelle est représenté par l'intervalle [10, 12] associé à la transition t_{i0}. Comme il n y a pas d'autres places d'entrée de la transition t_{i0}, ceci signifie que le jeton doit rester dans la place p_{i0} au moins 10 secondes et au plus 12 secondes avant que cette transition soit franchie (marquage de la place p_{i1} correspondant à la position trappe ouverte).

Le temps de réponse de l'électrovanne ev_{i1} est de 2 secondes représenté par l'intervalle temporel [0,2] associé à la transition tn_{i0}. Ceci signifie qu'elle est défaillante s'il y a absence de mouvement de l'ouverture de la trappe *i* après l'excitation de l'ouverture des trappes pendant deux secondes. Le calculateur demande alors l'utilisation de l'électrovanne de secours ev1 avec un temps de réponse de 3 secondes représenté par l'intervalle temporel]2,6] associé à la transition tr_{i0}. Si cette électrovanne est en bon fonctionnement, la transition tr_{i0} est franchie et la place ps_{i1} (représentant une reconfiguration du système) marquée. Comme il n'y a aucune autre transition de sortie qui peut consommer ce jeton, il ne peut rester dans ps_{i1} plus de 12 secondes. L'électrovanne de secours est alors libérée (marquage de la place ev_1_ok) par le franchissement de la transition tn_{i1}. Si l'électrovanne ev_{i1} est défaillante et l'électrovanne de secours est hors service ou occupée par une des deux autres trappes, alors la première transition red_{i1} menant vers l'état redouté est franchie 7 secondes après le marquage de ps_{i0}. Nous avons alors le marquage de la place E_red_{i1}.

La dynamique de la place p_{i0} peut être interrompue par le franchissement de la transition immédiate td_{i1} suite à la défaillance de l'électrovanne ev_{i1} après le début de l'ouverture de la trappe. Ceci correspond à un blocage de la trappe en ouverture représenté par le marquage de la place ps_{i2}. Dans ce cas l'électrovanne de secours ev_1 est sollicitée. Si elle n'est pas défaillante ou occupée par une des deux autres trappes, la transition tr_{i1} correspondant à une reconfiguration après blocage est franchie. Sinon la deuxième transition red_{i2} menant vers l'état redouté est franchie. On suppose que l'électrovanne ev_{i1} peut être réparée après défaillance par le franchissement de la transition rep_{i1}. Si l'électrovanne de secours est défaillante, elle est hors service. Les transitions liées aux défaillances et réparations des électrovannes sont stochastiques et peuvent être franchies à tout instant. L'abstraction temporelle de ce modèle est un réseau de Petri t-temporel modélisant la priorité de tir de transitions.

5.2.6.2 Modèle réseau de Petri de l'ouverture des trois trappes

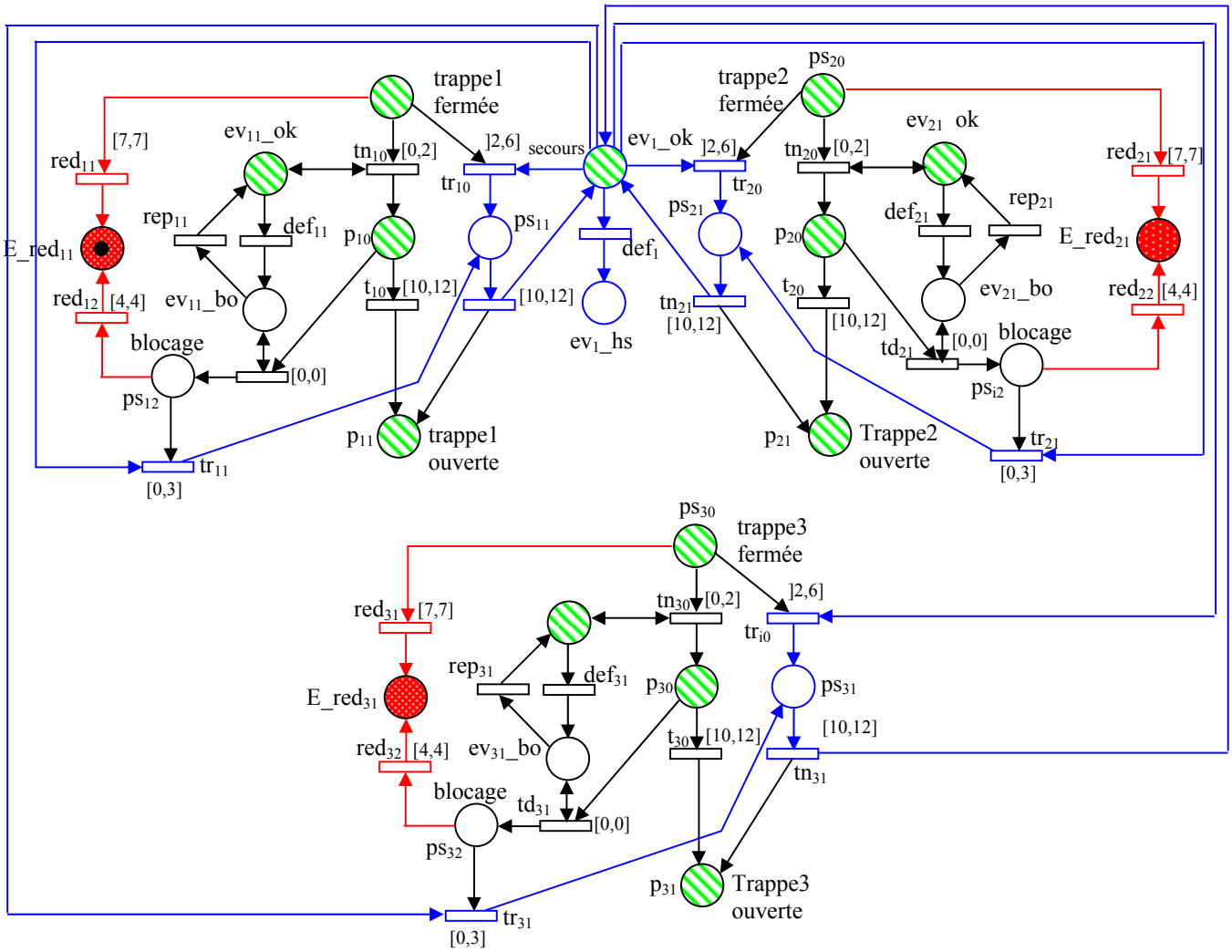


Figure 5.9. Modèle RdP de l'ouverture des trois trappes

Dans la figure 5.9 modélisant l'ouverture des trois trappes, les places rayées représentent le fonctionnement normal. Nous allons chercher les scénarios redoutés correspondant à la non-ouverture de la trappe 1, c'est-à-dire tous les scénarios qui mènent au marquage de la place E_red_{11}. Les scénarios redoutés correspondant à la non-ouverture de la trappe 2 et la trappe 3 peuvent être obtenus d'une façon similaire. L'analyse structurelle de l'outil TINA nous donne les invariants des places suivants:

M(11_bo) + M(ev11_ok) = 1
M(ev21_bo) + M(ev21_ok) = 1
M(ev31_bo) + (ev31_ok) = 1
M(E_red11) + M(p10) + M(p11) + M(ps10) + M(ps11) + M(ps12) = 1
M(E_red21) + M(p20) + M(p21) + M(ps20) + M(ps21) + M(ps22) = 1
M(ev1_hs) + M(ev1_ok) + M(ps11) + M(ps21) + M(ps31) = 1
M(E_red31) + M(p30) + M(p31) + M(ps30) + M(ps31) + M(ps32) =1

5.3. Génération de scénarios par ESA_PetriNet

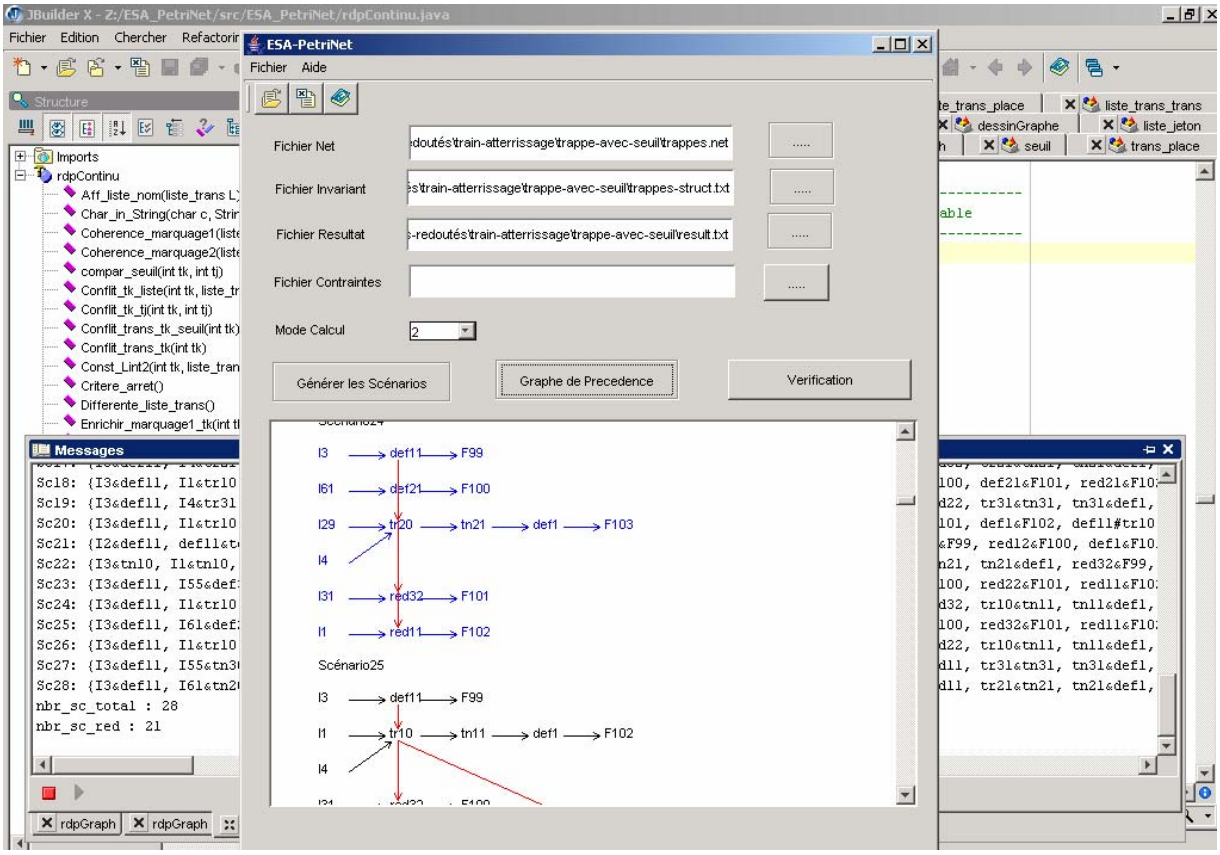


Figure 5. 10. Génération des scénarios par ESA_PetriNet

ESA_PetriNet génère tous les scénarios amenant les jetons dans la place E_red11. Nous obtenons alors 28 scénarios dont 21 sont redoutés en tenant compte de l'aspect continu du système (figure

5.10) et 114 scénarios dont 87 sont redoutés si l'on ne tient pas compte de l'aspect continu du système (figure 5.11). Dans cet exemple nous n'avons pas traité les boucles, nous avont choisi de mettre les transitions rep11, rep21, rep31 dans la liste des transitions interdites *LintEntree* en leur associant le label *F*.

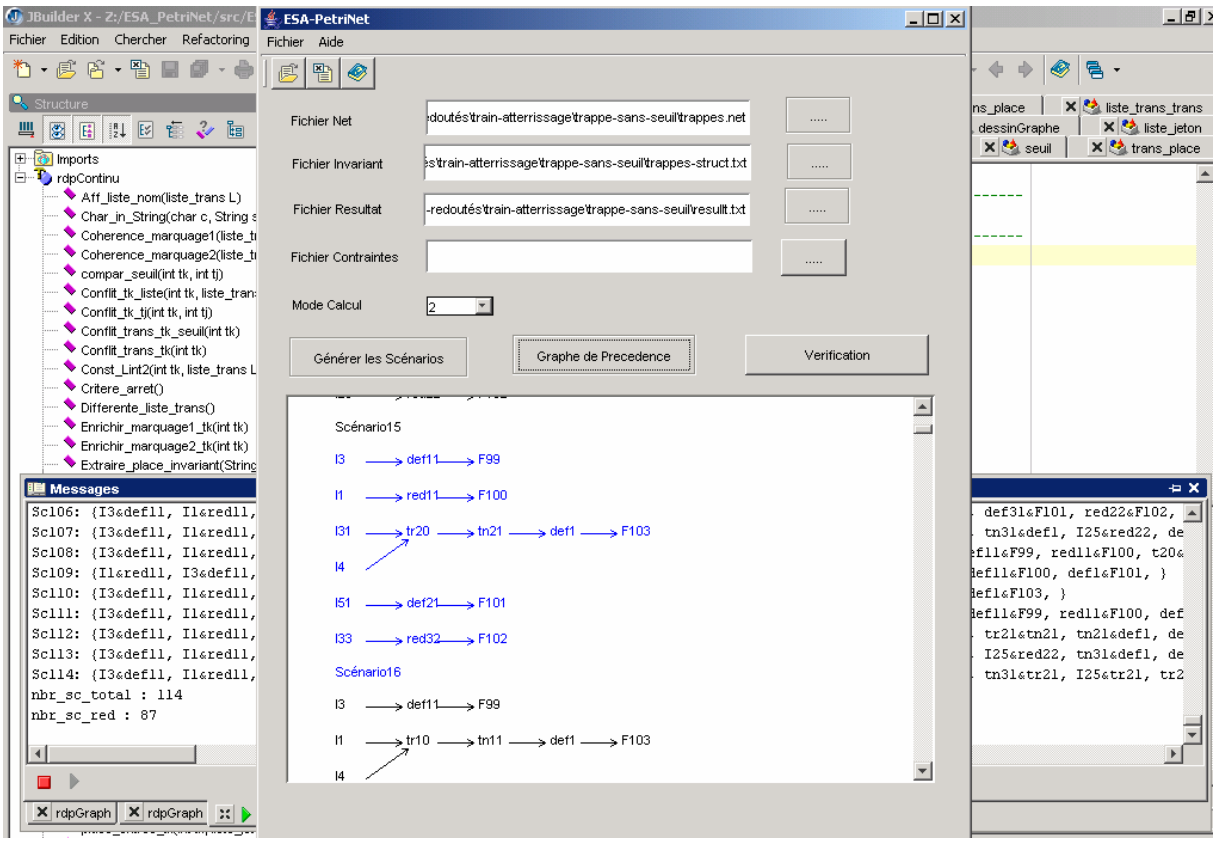


Figure 5.11. Génération des scénarios sans l'aspect continu

Parmi les 21 scénarios redoutés générés en tenant compte de l'aspect continu du système (figure 5.12), on trouve ceux correspondant à :

- Défaillance d'une électrovanne ev_{i1} (def_{i1}) et la défaillance de l'électrovanne de secours ev_1 (def_1) : sc1, sc2, sc3, sc4, sc5, sc10, sc12.
- Défaillance de l'électrovanne ev_{i1} (def_{i1}) suivi de l'utilisation de l'électrovanne de secours par une autre trappe (tr_{i0} *ou* tr_{i1}) : sc6, sc7, sc8, sc9, sc11, sc13, sc17, sc19, sc21, sc22, sc23, sc25, sc27, sc28.

Sc1: {I1,def11,def1,red32,red22,red11}
Sc2: {I1,def11,td11,def1,red32,red12,red22}
Sc3: {I1,tn10,def11,td11,def1,red32,red12,red22}
Sc4: {I1,def11,def1,red22,red11,def31,red31}
Sc5: {I1,def11,def1,red32,red11,def21,red21}
Sc6: {I1,def11,tr31,red22,red11,tn31,def1}
Sc7: {I1,def11,td11,tr31,red12,red22,tn31,def1}
Sc8: {I1,tn10,def11,td11,tr31,red12,red22,tn31,def1}
Sc9: {I1,def11,tr21,red11,def31,red31,tn21,def1}
Sc10: {I1,def11,def1,red22,red11,tn30,def31,td31,red32}
Sc11: {I1,def11,tr31,red11,def21,red21,tn31,def1}
Sc12: {I1,def11,def1,red32,red11,tn20,def21,td21,red22}
Sc13: {I1,def11,tr21,red32,red11,tn21,def1}
Sc17: {I1,def11,tr21,red11,tn30,def31,td31,red32,tn21,def1}
Sc19: {I1,def11,tr31,red11,tn20,def21,td21,red22,tn31,def1}
Sc21: {I1,def11,td11,tr21,red32,red12,tn21,def1}
Sc22: {I1,tn10,def11,td11,tr21,red32,red12,tn21,def1}
Sc23: {I1,def11,def31,tr30,red22,red11,tn31,def1}
Sc25: {I1,def11,def21,tr20,red32,red11,tn21,def1}
Sc27: {I1,def11,tn30,def31,td31,tr31,red22,red11,tn31,def1}
Sc28: {I1,def11,tn20,def21,td21,tr21,red32,red11,tn21,def1}

Figure 5.12. Les 21 scénarios redoutés correspondant à la trappe 1

5.4. Comparaison des résultats

5.4.1. Avec l'aspect contiu

Les scénarios générés en tenant compte de l'aspect continu du système et l'ordre d'apparition des événements lié à la dynamique continue est pris en compte. Si l'on prend par exemple le scénario sc13 **:** {I1, **def11**, **tr21**, red32, **red11**, tn21, def1}, dont la graphe de précédence est donné en figure 5.14, on voit bien que la cause de l'événement redouté (transition *red11*) est bien la défaillance de la propre électrovanne de la trappe (transition *def11*) et l'utilisation de l'électrovanne de secours par la trappe 2 (transition *tr21*). Les flèches en pointillées entre def11, red11 et tr21, red11 représentent les relations de causalité indirecte.

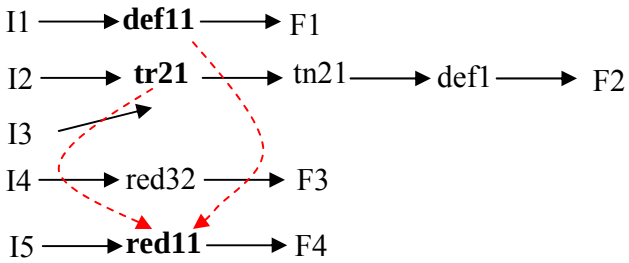
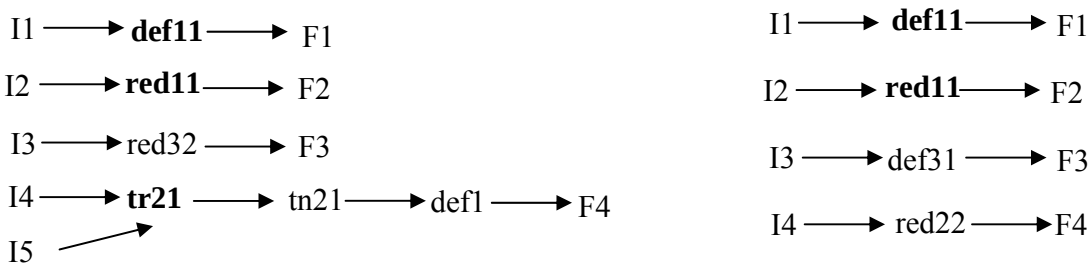


Figure 5.13. Pris en compte d'ordre lié à la dynamique continue

5.4.2. Sans l'aspect continu

En analysant les scénarios générés sans tenir compte de l'aspect continu du système, on voit bien que comme le montre la graphe de précédence de la figure 5.15 correspondant au scénario Sc37: {I1, **def11**, **red11**, red32, **tr21**, tn21, def1} que l'ordre d'apparition des événements lié à la dynamique continue n'est pas pris en compte. D'autre scénarion non recevables comme Sc73: {I1, **def11**, **red11**, def31, red22} dont le graphe de précédence est donnée en figure 5.16 sont.



a) Non pris en compte d'ordre lié au continu b) Scénario non recevable

Figure 5.14. Exemples de scénarios générés sans le continu

5.5. Génération de scénarios dans le réseau global

La figure 5.13 représente le réseau de Petri global de la sortie des trois trains d'atterrissage. Il regroupe les trois sous-réseaux : ouverture des trappes, sortie des trains et fermeture des trappes. Pour avoir un état redouté, il suffit que l'une des places E_red_{i1}, E_red_{i2}, E_red_{i3} soit marquée. On définit un nouvel état redouté *E_red* qui regroupe les trois précédentes comme le montrent les figures 5.13 et 5.15. Nous cherchons tous les scénarios qui mènent au marquage de la place *E_red*, c'est-à-dire le franchissement de l'une des transitions t_red_{i1}, t_red_{i2}, t_red_{i3}.

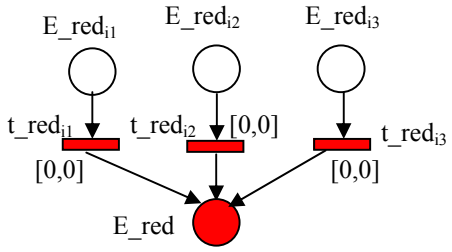


Figure 5.15. Etat redouté lié à la sortie des trois trains d'atterrissage

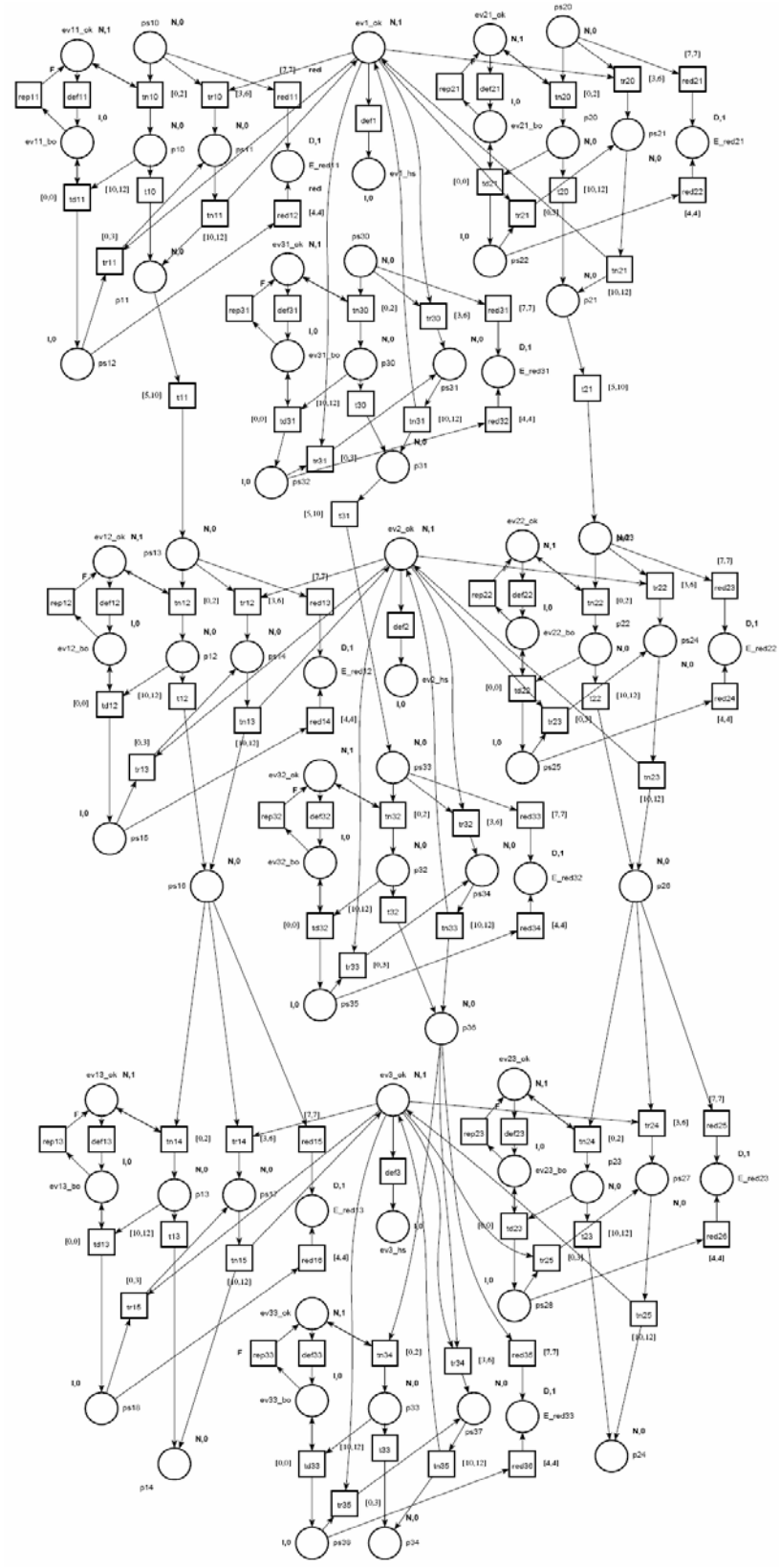


Figure 5.16. Modèle RdP de la sortie des trains d'atterrissage édité sur TINA

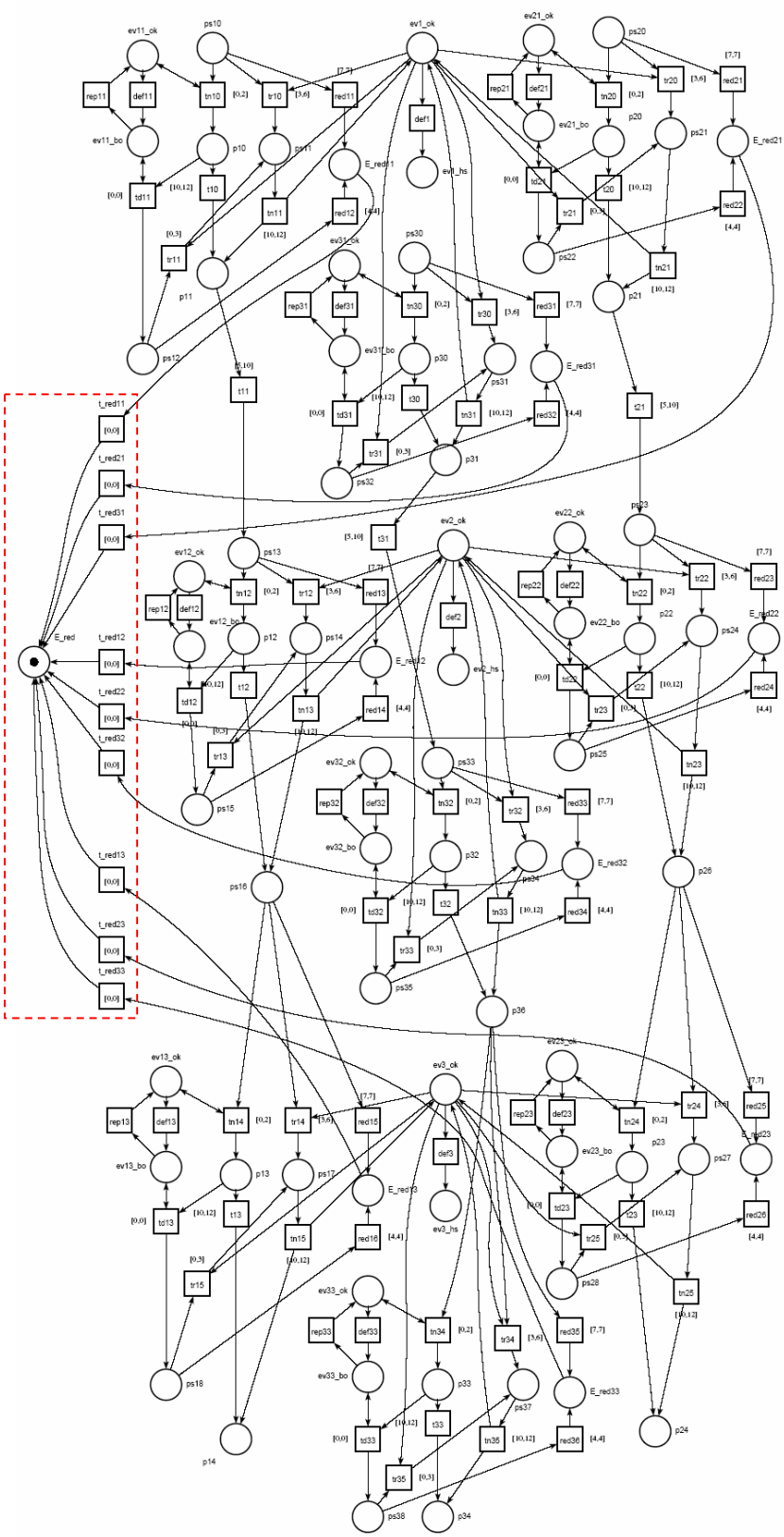


Figure 5.17. Modèle RdP de la sortie des trains avec E_red édité sur TINA

5.6. Conclusion

Ce chapitre nous a permis de valider notre approche en l'appliquant à un exemple de taille conséquente, conçu à partir d'un système réel (avion Rafale de Dassault Aviation). Nous avons délibérément restreint l'exemple d'application pour des soucis de clarté. Le réseau de Petri de l'exemple simplifié contient près d'une centaine de places. En mettant à profit la symétrie du système, il est possible de constituer le modèle global d'une manière relativement aisée.

Ce chapitre a également permis de montrer l'utilité de l'outil ESA_PetriNet que nous avons développé, ainsi que sa validité à travers la cohérence des scénarios fournis. Ceux-ci sont cohérents vis-à-vis de la dynamique continue du système et nous avons pu constater que le nombre de scénarios éliminés pour leur incohérence vis-à-vis de la dynamique continue est important.

Parmi les perspectives de développement de l'outil, il y a la prise en compte de la minimalité des scénarios, qui permettra d'éliminer les événements non nécessaires.

Dans le chapitre suivant, nous présentons deux exemples d'utilisation de l'outil pour la vérification des propriétés temporelles.

Chapitre 6 : Deux exemples de vérification de propriétés temporelles

6.1. Introduction

L'outil ESA_PetriNet a été développé pour la construction des scénarios redoutés. Le but de ce chapitre est de montrer qu'il peut également être utilisé dans un autre contexte, celui de la vérification de certaines propriétés temporelles. Nous avons choisi, pour illustrer notre démarche deux exemples : les trains d'atterrissage présenté dans le chapitre précédent et les calculateurs dans un avion. Dans le premier exemple l'utilisation est assez directe. Dans le second, elle nécessite de modifier les scénarios après leur génération.

La vérification consiste à déterminer si le système satisfait certaines propriétés. Le but de ce chapitre est d'adapter notre approche d'extraction des scénarios redoutés dans les systèmes hybrides à la vérification de propriétés. L'adaptation consiste à rajouter une fonction de vérification une fois tous les scénarios générés. Les propriétés que nous voulons vérifier sont de type temporel qui concerne la durée maximale d'un scénario. Les contraintes temporelles peuvent être aussi la durée entre deux commandes. Nous allons vérifier dans le premier exemple que la durée d'un scénario est inférieure à une certaine borne et dans le deuxième la durée entre deux commandes. Dans le premier exemple, le scénario implique à la fois le système de commande et le système commandé. Dans le deuxième il ne concernera que la politique de reconfiguration du système commandé.

6.2. Principe de l'utilisation d' ESA_PetriNet pour la vérification

Plaçons-nous donc dans le cas où la propriété à prouver est que lorsque les commandes du système sont dans une certaine configuration, c'est-à-dire lorsque l'on a un certain jeu de commandes, un certain état cible est atteint au bout d'un temps borné.

Alors que pour la recherche des scénarios redoutés il est important de fournir au concepteur une bonne description des bifurcations entre un comportement redouté et les comportements normaux proches (c'est ce qui nécessite de compléter l'exploration arrière par l'exploration avant), la vérification impose simplement de donner un ensemble de scénarios. Par exemple pour montrer qu'un état est atteint au bout d'un délai fini avec un certain jeu de commandes, il faut donner tous les scénarios compatibles avec cette commande et terminant dans l'état désiré. Il s'agit donc d'un raisonnement arrière. L'étape avant devant se réduire au simple fait de vérifier que le jeu de commandes considéré résout tous les conflits et que quand la commande est émise, le système va nécessairement évoluer suivant l'un des scénarios obtenus.

Un scénario est un ordre partiel entre des événements. Si l'on veut une vérification quantitative, c'est-à-dire si l'on veut montrer que pour un jeu de valeur fixé, la borne est inférieure à une certaine valeur, alors, l'abstraction temporelle du système hybride permet le calcul de la durée maximale de chaque scénario et l'on peut vérifier si la propriété est vraie ou non. Ce que nous allons illustrer par

cet exemple, c'est que le fait de connaître les scénarios sous la forme d'un ordre partiel, permet d'exprimer en fait leurs durées sous la forme d'expressions algébriques dans l'algèbre (max, +). Au lieu de simplement prouver que la propriété est vraie ou fausse, il est alors possible de donner une contrainte concernant un certain nombre de paramètres, pour que la propriété soit vérifiée.

La démarche consiste à modéliser le système par un réseau de Petri Prédicats Transitions Différentiels Stochastiques, à générer tous les scénarios puis l'abstraction temporelle est utilisée pour exprimer les contraintes temporelles quantitatives associées à chaque scénario. Enfin on en déduit les contraintes sur les paramètres pour que la propriété soit vérifiée. Dans cette démarche, l'outil ESA_PetriNet est utilisé pour l'obtention des scénarios et le module "struct" de TINA pour l'obtention des invariants de places.

6.3. Exemple 1 : trains d'atterrissage

Comme nous l'avons mentionné au chapitre précédent, Le contrôle des trains est assuré par trois commandes *E*, *R* et *B*. La commande E pour l'ouverture des trappes, la commande R pour la fermeture et la commande *B* est intermédiaire, dans ce cas les roues sont bloquées dans leur position courante.

Nous supposons dans ce chapitre que les mouvements d'ouverture et la fermeture des portes sont réalisés jusqu'au bout dans tous les cas. C'est-à-dire que dès qu'une porte commence à se fermer la commande (E ou R) reste effective jusqu'à la fin du mouvement pour les trois trains. La propriété à vérifier est que quand le pilote maintient la commande *E* le temps nécessaire pour atteindre l'état final (prêt pour l'atterrissage) est borné par Δ quelles que soient les commandes qui ont été réalisées précédemment (et donc quel que soit l'état du système d'atterrissage au début du scénario).

6.3.1. Modèle réseau de Petri pour la sortie d'un train

Une vue détaillée d'une rentrée d'un train (*train i*) est représentée sur la figure 5.5 du chapitre précédent. Notons que les places p_{i0} et p_{i4} sont les mêmes que celles représentées sur la figure 5.1. Dans des travaux précédents [Boniol et Carcenac 02] le système a été représenté par Lustre et Esterel et des automates temporisés. Nous supposons que les trains peuvent être bloqués en n'importe quelle position. Nous supposons que les trois trains sont indépendants.

Nous supposons que la dynamique continue du train peut être représentée par un délai (retard pur) et deux comportements linéaires (abstraction temporelle) comme le montre la figure 6.2. Le délai est délimité par l'intervalle [*dmi*, *dMi*] qui signifie que le retard peut varier entre la valeur minimale *dmi* et la maximale *dMi*. La pente de l'enveloppe inférieure est a_i et celle de l'enveloppe supérieure est b_i.

La place p_{i0} correspond à l'ouverture de la porte de la trappe. Sa dynamique continue est prise en compte par une abstraction temporelle. La durée minimale pour l'ouverture ou la fermeture de la trappe d'un train *i* est D_{mi} et sa durée maximale est D_{Mi}. Le temps correspondant à la contrainte temporelle est représenté par l'intervalle $[D_{mi}, D_{Mi}]$ associé à l'arc (p_{i0}, t_{i0}). Ceci signifie que le jeton doit rester dans la place p_{i0} au moins D_{mi} et au plus D_{Mi}. Comme il n'y a aucune autre transition qui peut consommer ce jeton, il ne peut rester dans p_{i0} plus de D_{Mi}.

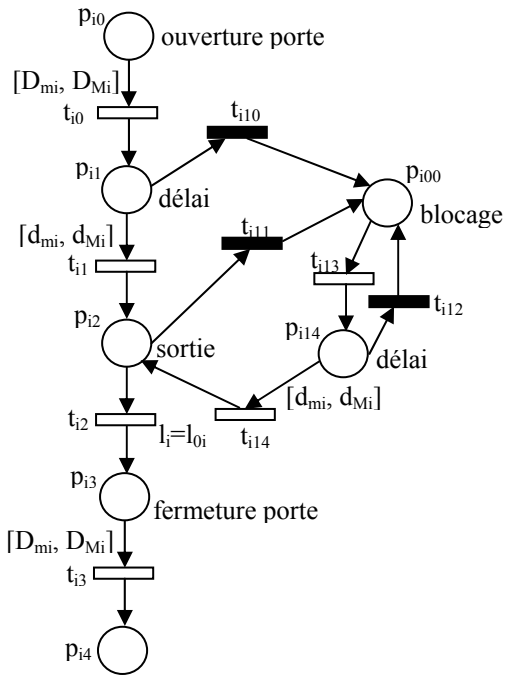


Figure 6.1. Modèle réseau de Petri pour la sortie d'un train *i* (cmd E)

L'abstraction temporelle de ce modèle est un réseau de Petri pt-arc-temporel (les contraintes temporelles sont associées aux arcs reliant les places aux transitions). C'est très semblable aux réseaux de Petri p-temporels dont les contraintes temporelles sont associées aux places. La seule différence est que les contraintes temporelles associées aux places peuvent changer en tenant compte de chaque transition de sortie de la place.

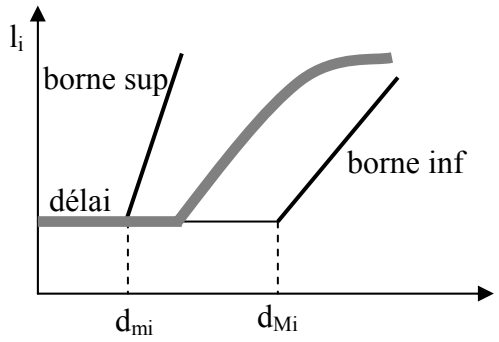


Figure 6.2. La dynamique continue d'un train *i*

Par exemple la place p_{i1} représente le retard pur du mouvement du train *i*, la contrainte temporelle correspondante est représentée par l'intervalle $[d_{mi}, d_{Mi}]$ associé à l'arc (p_{i1}, t_{i1}). La transition t_{i1} est associée à l'instant où le train commence à se déplacer après le retard pur.

La place p_{i2} représente l'interpolation linéaire de mouvement du train. L'équation différentielle :

$$dli/dt = ci \text{ avec } ci \in [ai, bi] \text{ et } ai \geq = 0 \qquad (1)$$

est associée à cette place. La valeur courante l_i de chaque position de train est associée au jeton correspondant. La transition t_{i2} est franchie quand $l_i = l_{i0}$. Pour être précis, elle est exactement franchie quand le temps correspondant à cette condition est atteint.

Finalement, la place p_{i3} correspond à la fermeture de la porte (même contrainte temporelle que pour p_{i0}) et la place p_{i4} à l'état d'attente nécessaire avant la synchronisation des trois trains représentée par la transition t_2.

Une bonne partie du réseau représente les états atteints quand la commande du pilote n'est pas maintenue dans la position E. Quand le pilote permute à R ou à B, les actions représentées par les places p_{i1} et p_{i2} sont interrompues *immédiatement* (transitions t_{i10} ou t_{i11}). Le jeton est mis dans la place p_{i00} et la valeur courante de l_i est mémorisée. Si la commande E est émise dans cet état, le processus est repris par les transitions t_{i13} et t_{i14}. Un nouveau retard pur initial est imposé par la place p_{i14} : un délai qui peut être interrompu par le tir de t_{i12} si le pilote déplace la commande à B ou encore à R.

6.3.2. Modèle de réseau de Petri pour la rentrée

Le modèle est similaire (figure 6.3). Le point important est que la place p_{i00} est partagée entre les deux réseaux de Petri des figures 6.1 et 6.3. L'état bloqué est le même car le pilote peut soudainement passer de mode rentrée au mode sortie et vice-versa. Les places p_{i5} et p_{i9} sont les mêmes que celles représentées sur la figure 5.1.

L'équation différentielle associée à la place p_{i7} est :

$$dli/dt = -ci \text{ avec } ci \in [ai, bi] \text{ et } ai \geq = 0 \qquad (2)$$

Il est important de préciser que le seuil avec lequel le franchissement de la transition t_{i7} est synchronisé est $l_i = 0$.

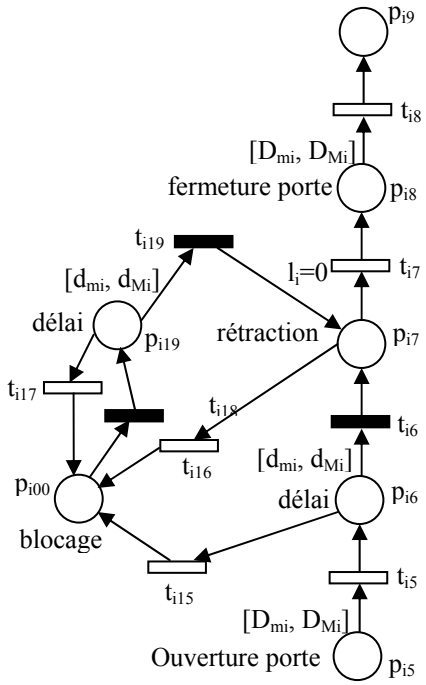


Figure 6.3. Modèle réseau de Petri pour la rentrée d'un train i (cmd R)

6.3.3. Première analyse et abstraction temporelle

6.3.3.1 Analyse des invariants

En remplaçant les rectangles *"train d'atterrissage i"* dans la figure 5.5 du chapitre 5 par les sous-réseaux des figures 6.1 et 6.3, on peut facilement montrer qu'il y a trois p-invariants positifs : pour chaque train d'atterrissage, la somme des marquages de toutes les places plus le marquage des places p_1 et p_2 est égale à 1. Ceci signifie que le réseau de Petri est 1 borné.

Les équations différentielles sont associées uniquement à six places : les deux places p_{i2} et p_{i7} pour chaque train. Ceci signifie que les valeurs des variables l_i représentant les positions de train changent uniquement quand une de ces places contient un jeton. Pour un train donné i, l_i ne peut être simultanément croissante et décroissante car il n'est pas possible d'avoir simultanément un jeton dans la place p_{i2} et un jeton dans la place p_{i7} (conséquence des p-invariants ci-dessus). On suppose que la valeur initiale de ces trois variables continues (quand $M(p_1) = 1$) est 0.

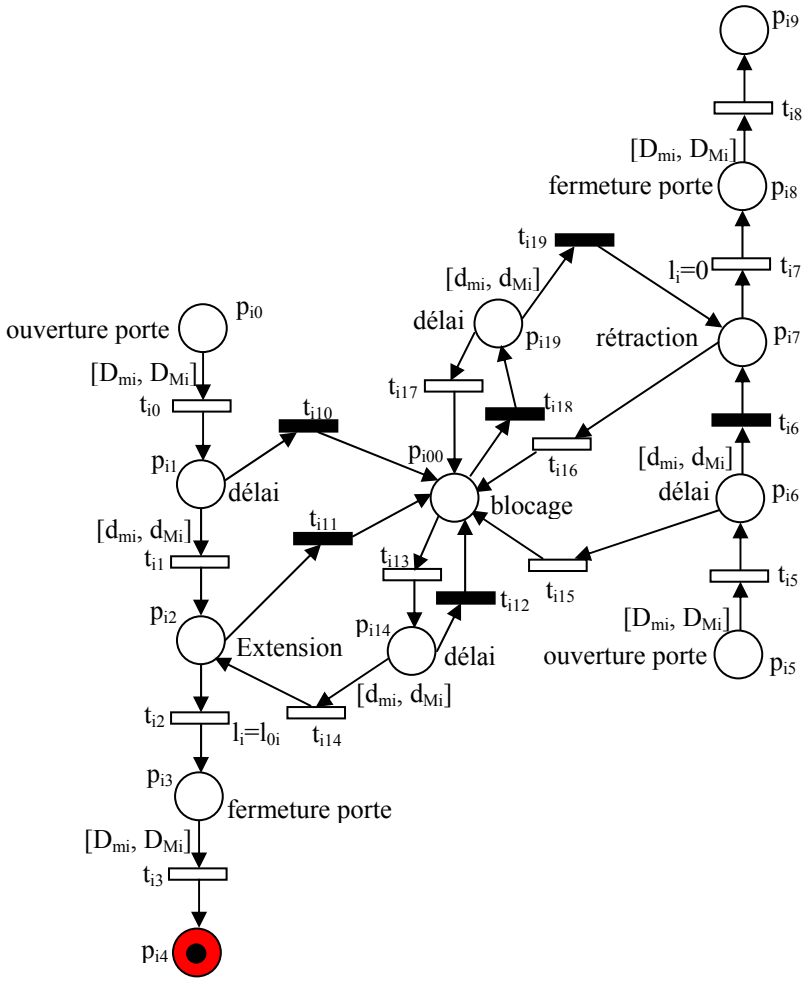


Figure 6.4. Modèle réseau de Petri complet d'un train *i* : sortie et rentrée

6.3.3.2 Délimitation des domaines des variables continues

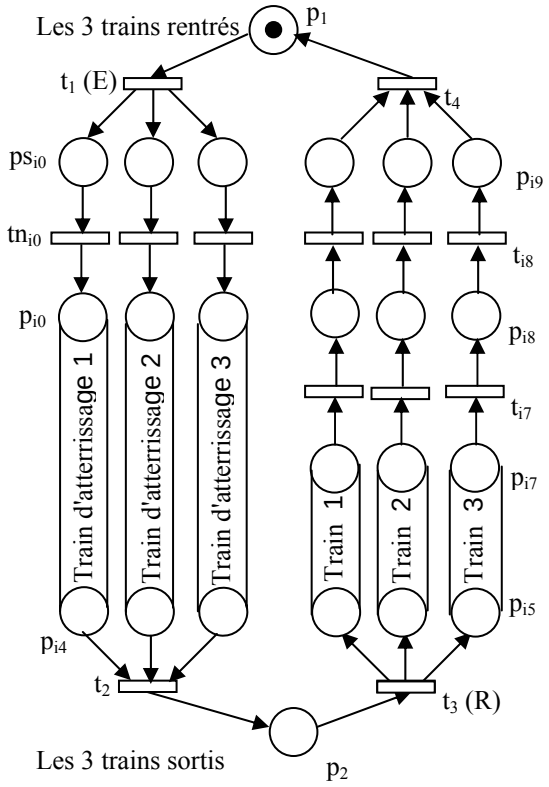


Figure 6.5. Modèle réseau de Petri des trois trains : sortie et rentrée

Par un raisonnement arrière à partir de la place p_1 du la figure 6.5, on peut montrer que n'importe quel scénario permettant la production d'un jeton dans la place p_1 génère le scénario unique s_1 caractérisé par le séquent :

$$p_{17} \otimes p_{27} \otimes p_{37}, t_{17}, t_{27}, t_{37}, t_{18}, t_{28}, t_{38}, t_4 \vdash p_1 \qquad (3)$$

Comme les transitions t_{i7} sont seulement tirées quand $l_i = 0$, et comme les variables l_i sont constantes dans les places p_{i8}, p_{i9} et p_1 alors nécessairement $\forall$i, $l_i = 0$ quand $M(p1) = 1$. Similairement, si $M(p2) = 1$ alors $\forall i$, $l_i = l_{0i}$. La position du train i est aussi $l_i = l_{0i}$ à la présence d'un jeton dans les places p_{i5}, ou p_{i6} et $l_i = 0$ à la présence d'un jeton dans les places p_{i0} ou p_{i1}.

Maintenant considérons l'hypothèse suivante :

$\forall$ M marquage accessible,

$$\forall i,\ 0 \le l_i \le l_{0i} \qquad (4)$$

La variable l_i est croissante uniquement quand $M(p_{i2}) = 1$. Pendant cette action, la transition t_{i2} reste sensibilisée et sera franchie dès que la valeur $l_i = l_{0i}$ est atteinte. Instantanément le jeton de la place p_{i2} sera déplacé et la valeur l_i reste constante. Pour la limite minimale de l_i on peut faire le même

raisonnement impliquant la place p_{i7} et la transition t_{i7}. Comme la condition 4 est initialement vraie, elle est vraie pour chaque état accessible.

6.3.3.3 Abstraction temporelle

Dans la description du problème, un certain nombre de comportements dynamiques ont déjà été donnés sous la forme d'une abstraction temporelle. Il s'agit d'une part de la durée d'ouverture et de fermeture des portes des trappes. On associe ainsi la durée $[D_{mi}, D_{Mi}]$ aux arcs (p_{i0}, t_{i0}) et (p_{i8}, t_{i8}). Il s'agit également des retards purs $[d_{mi}, d_{Mi}]$ associés aux arcs (p_{i1}, t_{i1}) et (p_{i14}, t_{i14}) ainsi qu'aux arcs (p_{i6}, t_{i6}) et (p_{i19}, t_{i19}). La durée de l'activité d'ouverture et de fermeture associée aux places p_{i2} et p_{i7} dépend de la dynamique décrite par l'équation (1) et de la position réelle de la roue lors du déroulement du scénario considéré. D'une façon générale, cette durée va varier entre 0 et l_{0i}/a_i.

6.3.4. Génération des scénarios et des réseaux de contraintes temporelles

Ce qu'il faut, c'est obtenir tous les scénarios amenant les jetons dans les places p_{i4}, quelles que soient leur localisation, lorsque la commande "ouverture" est appliquée. Cette commande autorise le franchissement de toutes les transitions représentées par un rectangle blanc sur la figure 6.4 et interdit toute les transitions représentées par un rectangle noir.

Notre outil ESA_PetriNet peut être utilisé de la façon suivante. Nous considérons que les places p_{i4} sont des états redoutés et nous recherchons tous les scénarios qui y mènent. Nous avons déjà en interne une structure de données correspondant à une liste de transitions interdites dans le but, notamment d'éviter les cycles ou de recalculer des scénarios déjà trouvés. Il suffit donc de l'initialiser par la liste des transitions bloquées par la commande (les transitions représentées en noir). Nous obtenons alors les 4 scénarios de la figure 6.6.

A partir de chacun de ces scénarios il est alors possible d'engendrer un ensemble de contraintes temporelles qui doivent être vérifiés. Par exemple pour les scénarios "Scénario1" nous obtenons le réseau de contraintes temporelles de la figure 6.7. Lorsqu'un arc entre une place et une transition est directement associée à une contrainte temporelle, nous portons la valeur de cette contrainte. La borne minimale est associée à l'arc dans le sens direct et la négation de la borne maximale est associée à l'arc inverse. Quand il s'agit d'une place associée à une équation différentielle, nous portons la valeur de l'abstraction temporelle avec les connaissances déduites du scénario étudié. Ainsi dans le cas du scénario "Scénario1", la roue est complètement rentrée quand le scénario commence (place p_{i0}). Donc $l_i=0$. Donc la durée minimale pour atteindre le seuil l_{0i} est de l_{0i}/b_i (borne max de la dynamique) et la valeur maximale sera de l_{0i}/a_i. Ce sont ces valeurs que l'on porte sur le réseau de contraintes temporelles.

Dans le cas du scénario "Scénario4", nous savons au contraire que la roue était déjà complètement sortie, mais qu'une commande intempestive avait juste demandée l'ouverture de la porte pour la faire entrer (place p_{i5}). Donc cette fois les contraintes minimales et maximales associées à l'arc reliant les franchissements des transitions t_{i14} et t_{i2} seront toutes deux égales à 0. Nous obtenons alors le réseau de contraintes temporelles de la figure 6.8.

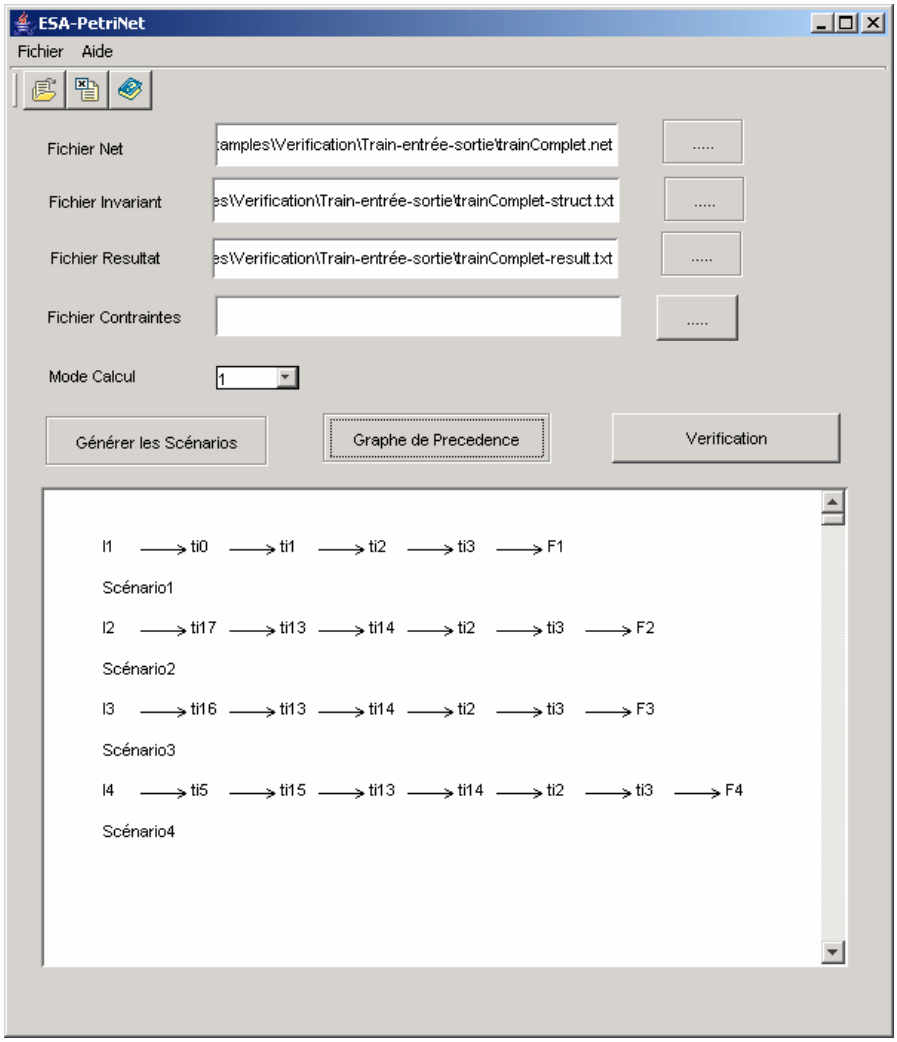


Figure 6.6. Génération exhaustive de scénarios avec ESA-PetriNet

6.3.5. Comparaison des scénarios

Il se trouve que nous sommes dans un cas simple car nous pouvons comparer les durées maximales des 4 scénarios sans devoir affecter des valeurs numériques aux paramètres. Cette durée maximale est $2.D_{Mi} + d_{Mi} + l_{0i}/a_i$. Donc si les paramètres vérifient la contrainte suivante :

$$2.D_{Mi} + d_{Mi} + l_{0i}/a_i < \Delta$$

alors la propriété est vérifiée.

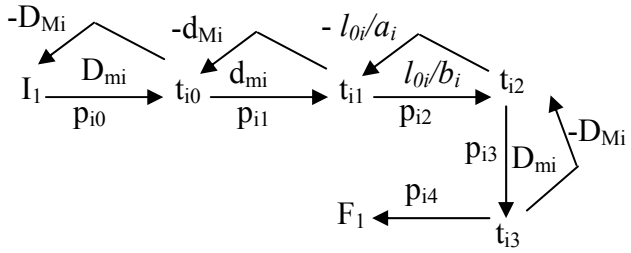


Figure 6.7. Réseau des contraintes temporelles du scénario S1

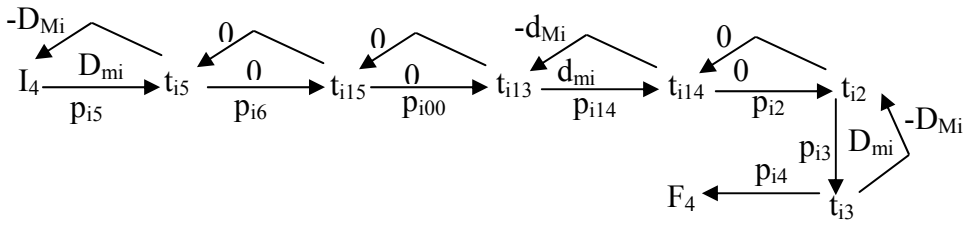


Figure 6.8. Réseau des contraintes temporelles du scénario S2

6.4. Exemple 2 : Calculateurs de vol

Dans l'exemple précédent, il y avait un aspect hybride dû au fait que les contraintes temporelles découlaient de la dynamique continue du système commandé. Mais cet aspect hybride n'a finalement aucun impact sur l'existence des scénarios. Il ne sert qu'à calculer leur durée et ne met donc pas en évidence l'intérêt de la modification de l'algorithme d'extraction des scénarios que nous avons introduite dans la section 2 du chapitre 3.

Dans cette partie nous allons considérer un exemple plus complexe où il s'agit de montrer que le comportement d'un système complexe de reconfigurations est correct. Ce deuxième cas d'étude ne présente qu'un aspect temporel, sans définition d'une dynamique hybride, car il se focalise sur la partie reconfiguration d'un système de commande sans modélisation de la partie commandée. Toutefois, des notions de priorité amènent à restreindre les scénarios trouvés en rajoutant des contraintes non explicitées par le réseau de Petri. Cet ajout de contraintes revient en fait à interdire certains comportements et le mécanisme à mettre en œuvre est finalement très proche de celui que nous avons introduit dans la section 2 du chapitre 3. C'est pourquoi nous le présentons ici. Il justifie que nous envisageons d'étendre notre outil pour prendre en compte le fait que certains scénarios puissent être invalidés par des mécanismes de priorité.

Cet exemple est un système d'avionique considéré comme un problème type dans la littérature. Il a été montré que l'on pouvait le traiter par des outils comme l'analyse sous contraintes [Bel et al 00] ou comme AltaRica [Pagetti 04]. Dans ce travail nous allons appliquer notre outil ESA_PetriNet pour vérifier que la durée minimale et la durée maximale de notre scénario redouté est comprise entre [20, 160] unités de temps.

6.4.1. Spécification et modélisation du système

6.4.1.1 Description générale

Le système est constitué de la façon suivante :

1. Trois calculateurs *Fl*, *Fr* et *Fb* se chargent de contrôler la gouverne.
2. Deux bus principaux réalisent les communications
   - *Clr* connecte *Fl* et *Fr*
   - *Clrb* relie *Fl* à *Fb* et *Fr* à *Fb*
   - un autre bus *Cd* assure la liaison entre les calculateurs *Fl*, *Fr* et *Fb* et la gouverne (il n'est pas représenté sur la figure 6.9 et les communications qu'il assure sont supposées fiables).

3. Initialement, le calculateur *Fl* contrôle la gouverne gauche tandis que *Fr* et *Fb* font d'autres tâches. La commande de la gouverne est "*cmd*" et elle est donc envoyée par *Fl*. Dans cette phase, *Fr* et *Fb* n'ont pas le droit d'envoyer cette commande.

4. Si *Fl* tombe en panne, il se déconnecte. Ceci est détecté par les bus *Clr* et *Clrb*. C'est le calculateur *Fr* (qui commande normalement la gouverne droite) qui doit reprendre la commande s'il n'est pas lui-même défaillant.

5. Si *Fr* tombe en panne, alors c'est *Fb* qui doit prendre la relève (on suppose qu'il n'est jamais défaillant).

Nous avons les données quantitatives suivantes:

- Temps de cycle des calculateurs *Fl*, *Fr* et *Fb* : *dl*, *dr*, *db*, respectivement
- Gigues des calculateurs *Fl*, *Fr* et *Fb* : *gl*, *gr* et *gb*, respectivement
- Nous supposons bien entendu que : $gl < dl$, $gr < dr$ et $gb < db$
- Temps de cycle des bus *Clr*, *Clrb* : *dc*, *dd*, respectivement
- Gigues des bus *Clr*, *Clrb* : *gc*, *gd*, respectivement

Nous supposons bien entendu que : $gc < dc$ et $gd < dd$

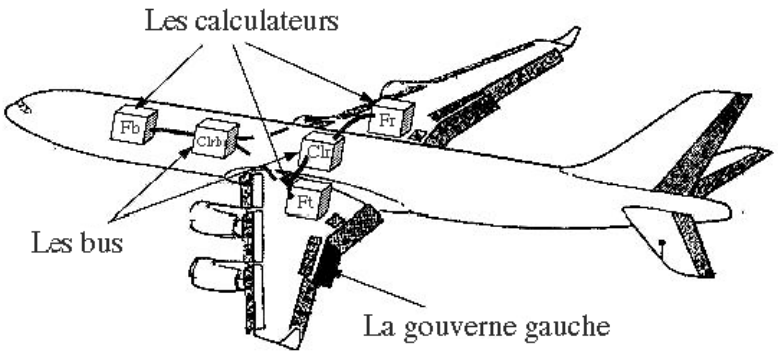


Figure 6.9. Vue du système dans l'avion

Propriété à vérifier

P1 : A tout instant, au plus un calculateur peut envoyer une commande "*cmd*"

P2 : Le délai entre deux commandes "*cmd*" doit être compris entre "*dmin*" et "*dmax*"

6.4.1.2 Modèle global

L'application étant complexe, nous avons choisi une représentation modulaire qui peut être vue comme un ensemble d'objets (figure 6.10). Le modèle global est formé de cinq objets représentant les trois calculateurs *Fl*, *Fr* et *Fb* et les deux systèmes de communication *Clr* et *Clrb*. Pour simplifier les notations, les paramètres des calculateurs *Fl*, *Fr* et *Fb* seront indicés respectivement par *l*, *r* et *b* et ceux des systèmes de communication *Clr* et *Clrb*, respectivement par *c* et *d*.

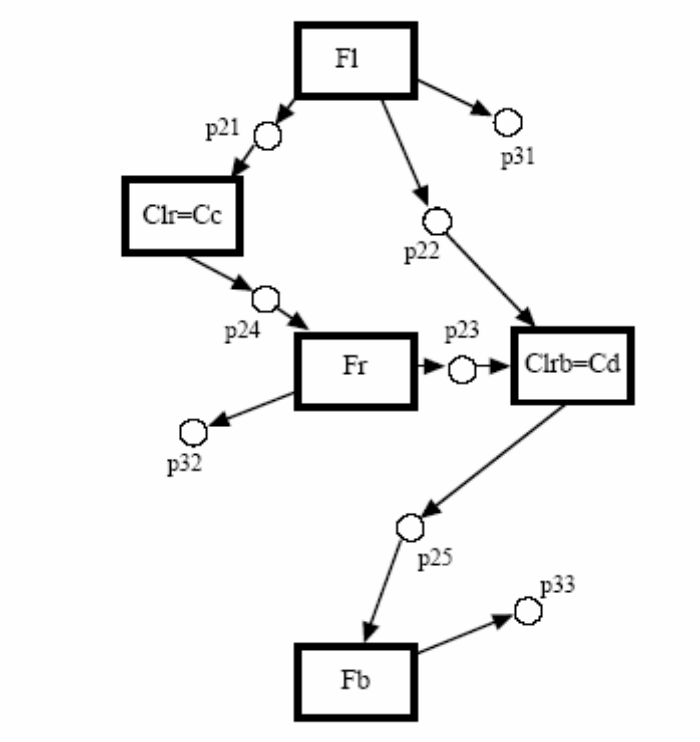


Figure 6.10. Schéma général du système

Les places *p31*, *p32* et *p33* correspondent aux émissions de commande "*cmd*" à partir des trois calculateurs. Dans une version plus complète, ces places seraient connectées au (ou aux) bus reliant les calculateurs aux actionneurs.

Les places *p21* et *p22* reçoivent un jeton quand le calculateur *Fl* est défaillant (passe de l'état de bon fonctionnement à l'état défaillant). La place *p23* reçoit un jeton quand le calculateur *Fr* est défaillant.

Le rôle de *Clr* est de prévenir le calculateur *Fr* en mettant un jeton dans la place *p24* quand *Fl* est défaillant, et celui de *Clrb* est de mettre un jeton dans la place *p25* pour prévenir *Fb* quand les deux calculateurs *Fl* et *Fr* sont défaillants. Dans les paragraphes suivant nous donnons les différents modèles réseaux de Petri que nous avons réalisé pour modéliser le système.

6.4.1.3 Modélisation du comportement du calculateur *Fr*

Le calculateur *Fr* est le seul qui comprenne à la fois la fonction de reprise en secours et qui peut lui-même être défaillant. C'est donc le plus complexe et il comprend tous les scénarios des calculateurs *Fl* et *Fb*. Le réseau de Petri représentant le comportement de ce calculateur est donné en figure 6.11.

La place p*r11* et la transition *tr11* correspondent au fonctionnement du calculateur *Fr* lorsqu'il ne commande pas la gouverne gauche. L'étiquette temporelle associée à l'arc (*pr11*, *tr11*) correspond au temps de cycle. Cela veut dire que le jeton doit rester dans la place *pr11* pendant une durée minimale et maximale (donc exactement cette durée) de *dr* pour les scénarios qui sont tels qu'il est consommé par la transition *tr11*. Il s'agit d'un réseau de Petri arc pt-temporel.

Les places pr22, pr23 et pr21 et les transitions tr21 et tr22 correspondent au fonctionnement de secours car la transition tr12 est franchie quand un jeton arrive dans la place p24. Pour que le

fonctionnement soit correct, il faut que la transition tr12 soit franchie sans attente, prioritairement vis-à-vis de tr11. Cette propriété n'est pas explicite dans le réseau de Petri et nous devrons la rajouter a posteriori dans les graphes de précédence représentant les scénarios. Chaque fois que la transition *tr21* est franchie, un jeton est mis dans la place *p32* ce qui correspond à l'émission d'une commande "*cmd*". L'intervalle [0, gr] associée à l'arc (*pr22*, *tr21*) correspond à la gigue. La commande n'est pas toujours émise en début de cycle et l'attente peut varier entre une durée de 0 et une durée de *gr*. La durée correcte du cycle en cas de commande est assurée par le temps associé à l'arc (pr21, tr22).

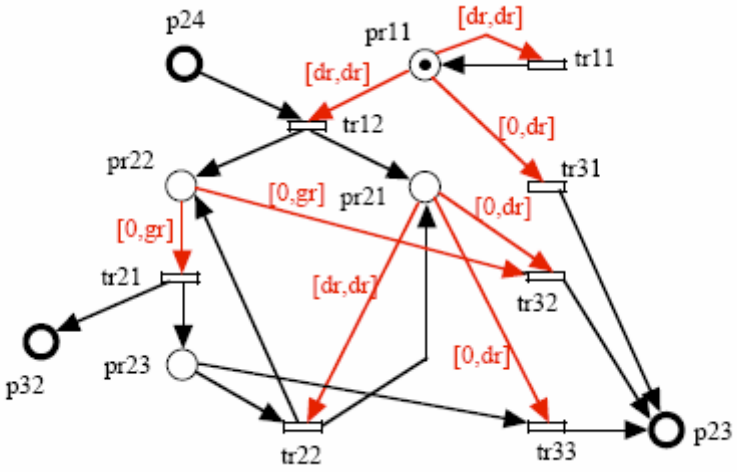


Figure 6.11. Réseau de Petri du calculateur Fr

Les trois transitions *tr31*, *tr32* et *tr33* correspondent aux défaillances du calculateur *Fr*. Dans le premier cas la défaillance se produit avant qu'il ait été utilisé en secours, le deuxième cas correspond à une défaillance se produisant en début de cycle, avant l'envoi d'une commande "*cmd*" et le troisième cas correspond à une défaillance en fin ce cycle.

Nous avons les invariants des places suivants :

$$M(pr11) + M(pr21) + M(p23) = 1 \quad (1)$$

$$M(pr11) + M(pr22) + M(pr23) = 1 \quad (2)$$

$$M(pr21) = M(pr22) + M(pr23) \quad (3)$$

6.4.1.4 Modélisation du comportement des calculateurs *Fl* et *Fb*

Les comportements de ces calculateurs sont des restrictions de celui du calculateur Fr. Le réseau de Petri représentant le comportement du calculateur *Fl* est donné dans la figure 6.12. Comme il n'a pas de commutation de mode de fonctionnement, l'équivalent des places p24 et pr11 n'apparaît pas.

Le réseau de Petri représentant le comportement du calculateur Fb est donné dans la figure 6.13. Comme il est supposé ne jamais être défaillant, c'est l'équivalent de la place p23 qui n'apparaît pas.

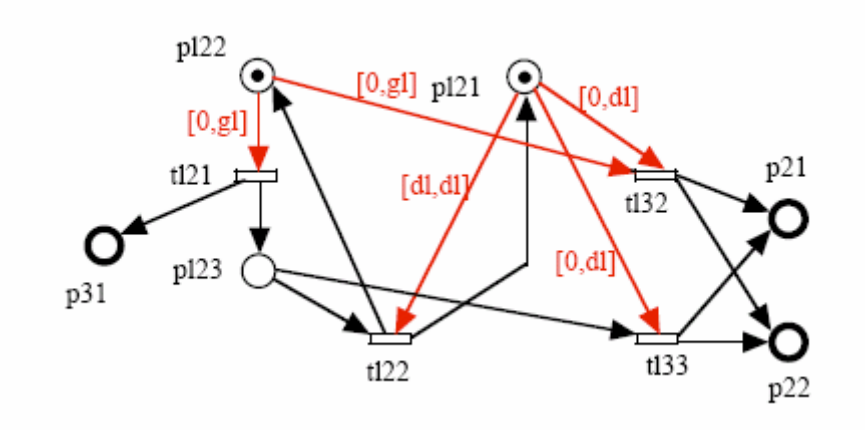


Figure 6.12. Réseau de Petri du calculateur Fl

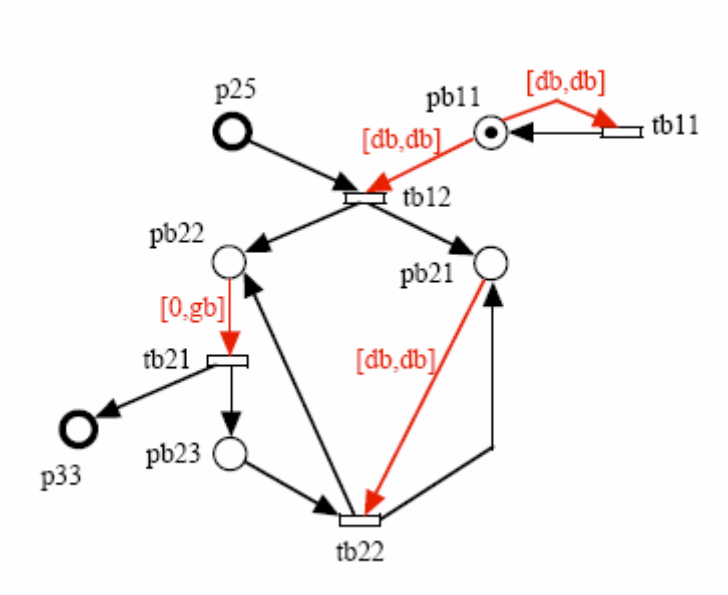


Figure 6.13. Réseau de Petri du calculateur Fb

6.4.1.5 Modélisation du comportement de *Clr*

Les systèmes de communication (bus) *Clr* et *Clrb* sont chargés de détecter si certains calculateurs sont défaillants, et dans ce cas de demander à un calculateur de changer de mode de fonctionnement pour assurer la reprise des commandes.

Le comportement de *Clr* est décrit par le réseau de Petri de la figure 6.14. Le rôle de la place *pc11* est de mettre en oeuvre une horloge. La transition *tc11* doit être franchie exactement tous les *dc* instant de temps pour respecter la contrainte temporelle [dc, dc] associée à l'arc (*pc11, tc11*). Lors de chaque cycle, mais avec une gigue pouvant varier entre *0* et *gc*, le bus lit le contenu de la place *p21*. S'il y a un jeton c'est que le calculateur *Fl* est défaillant. Dans ce cas un jeton est mis dans la place *p24*. S'il n'y a pas de jeton dans la place *p21* (et seulement dans ce cas) c'est la transition *tc14* qui est franchie. Notons que cette contrainte n'est pas exprimée par le réseau de Petri qui en cas de conflit peut franchir indifféremment *tc13* ou *tc14*. Cette contrainte sera ajoutée après coup dans le réseau de contrainte. Nous avons les invariants de places suivants :

$$M(pc11) = 1 \qquad (4)$$

$$M(pc12) + M(pc13) + M(pc14) = 1 \qquad (5)$$

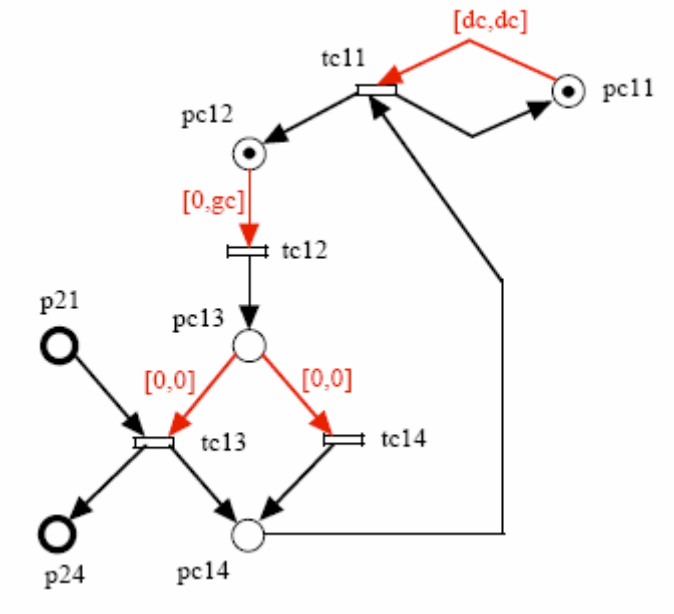


Figure 6.14. Réseau de Petri du bus Clr

6.4.1.6 Modélisation du comportement de *Clrb*

Le comportement est décrit par le réseau de Petri de la figure 6.15. La seule différence concerne le fait que la transition *td13* possède trois places en entrée et non deux. Ce qui complique, c'est que le fait que cette transition soit prioritaire vis-à-vis de la transition *td14* ne peut plus s'exprimer en disant que la production du jeton dans la place *p22* doit se situer entre le dernier franchissement de *td14* et le franchissement de *td13*. En effet, il faut qu'il y ait à la fois un jeton dans *p22* et un jeton dans *p23* pour pouvoir franchir *td13*. La contrainte à prendre en compte implique donc trois événements et non deux.

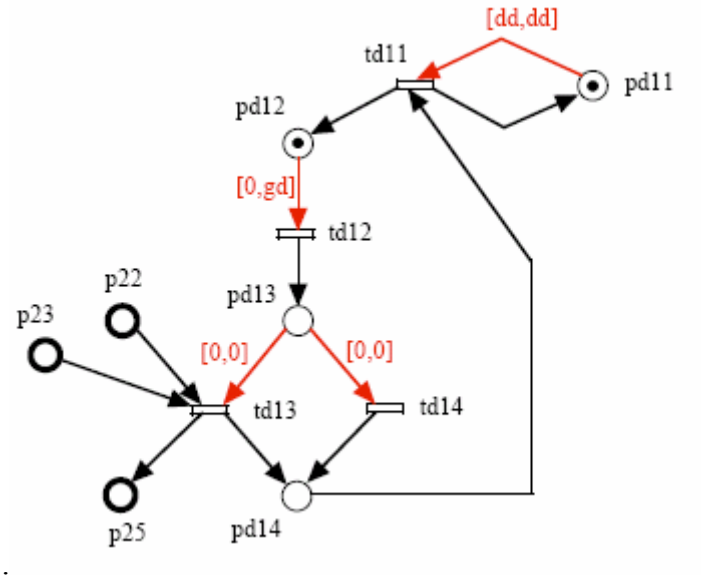


Figure 6.15. Réseau de Petri du bus Clrb

6.4.2. Génération des scénarios par ESA_PetriNet et analyse

6.4.2.1 Analyse du système global par TINA

Les contraintes temporelles étant associées aux arcs et non aux transitions, pour pouvoir entrer le réseau global dans l'outil TINA nous avons enlevé les contraintes temporelles. Le réseau global, tel qu'il est représenté graphiquement à l'aide de l'outil TINA est donné par la figure 6.16.

On peut remarquer que l'invariant :

$$M(pl22) + M(pl23) + M(p21) + M(p24) + M(pr22) + M(pr23) + M(p23) + M(p25) + M(pb22) + M(pb23) = 1 \quad (6)$$

garantit que deux commandes ne peuvent pas être émises simultanément. En effet, les trois transitions correspondant aux émissions de commande (*tl21, tr21* et *tb21*) possèdent une place d'entrée appartenant à cet invariant. Elles ne peuvent donc être simultanément sensibilisées. Cela prouve la propriété *P1*.

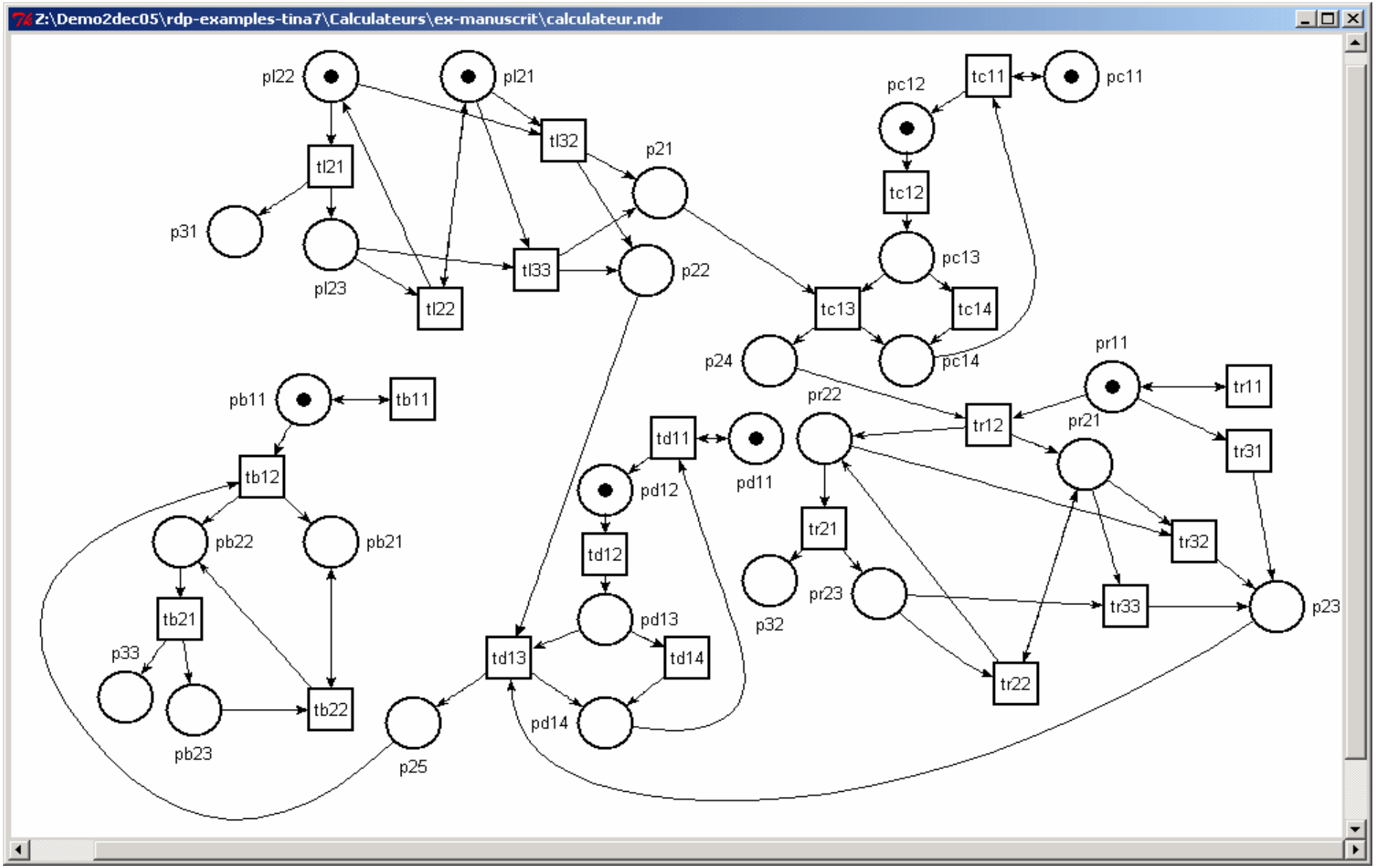


Figure 6.16. Réseau de Petri du système édité sur TINA

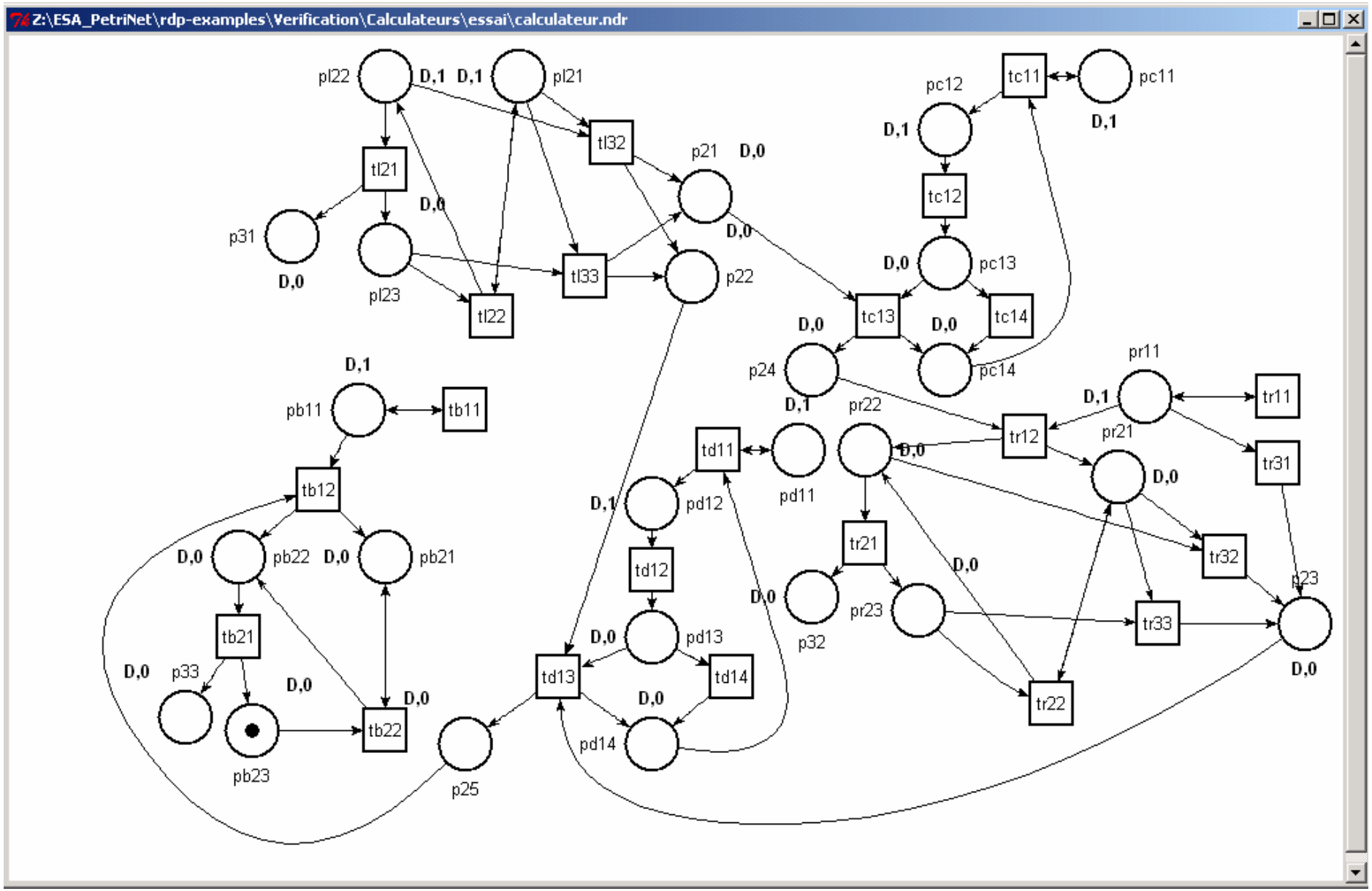


Figure 6.17. RdP du système avec le marquage initial

6.4.2.2 Recherche de scénarios par ESA_PetriNet

Parmi les scénarios générés en appliquant ESA_PetriNet sur l'exemple des calculateurs, on trouve ceux correspondant à deux commandes successives de deux calculateurs. Les scénarios correspondant à l'émission de deux commandes successives d'un même calculateur ne peuvent être générés actuellement par ESA_PetriNet à cause de l'interdiction de franchissement de la même transition deux fois pour éviter les boucles. La prise en compte plus fine des boucles devra donc faire partie des évolutions à envisager pour utiliser ESA_PetriNet dans un contexte de vérification.

La vérification des contraintes temporelles est un traitement qui se fait une fois tous les scénarios générés. Dans cet exemple, nous avons choisi les paramètres suivants :

$$dl = dc = db = 20$$

$$dd = 40$$

$$gl = gc = gd = gr = gb = 10$$

La figure 6.17 représente un des scénarios générés par ESA-PetriNet, il correspond à l'émission d'une commande par le calculateur *Fl* (transition *tl21*) suivi d'une commande émise par le calculateur *Fb* (transition *tb21*).

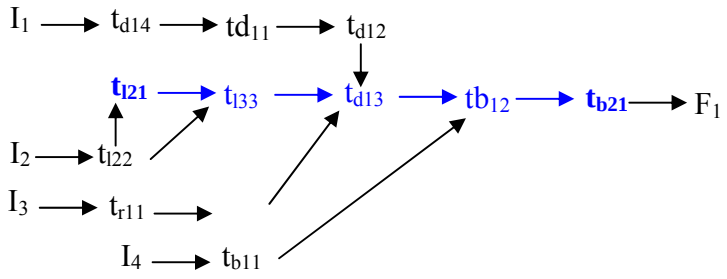


Figure 6.18. Scénario de deux commandes successives émises par Fl suivi de Fb

L'outil ESA_PetriNet nous le donne sous la forme du graphe de précédence de la figure 6.16, mais nous allons voir que cela est insuffisant. L'introduction des contraintes temporelles ne pose initialement pas de problème. Elles sont en effet associées à des arcs reliant les places aux transitions. Dans le graphe de précédence, un arc entre deux franchissements est associé à une place, mais comme on connaît le nom de la transition qui va consommer le jeton dans la place (celle qui est franchie lors du franchissement associé à la destination de l'arc), il n'y a aucune ambiguïté.

Pour le scénario de la figure 6.17, si nous ne considérons que le fragment situé entre les franchissements des transitions *tl21* et *tb21*, nous obtenons le réseau de contraintes temporelles de la figure 6.18. Par exemple, entre *tb12* et *tb21* nous avons la contrainte [0,10] qui correspond à la valeur numérique de la contrainte [0,*gb*] associée à l'arc (*pb22*, *tb21*). Quand il n'y a aucune contrainte associée, cela veut dire que l'arc ne représente qu'une contrainte de précédence

qualitative, c'est-à-dire que la contrainte quantitative équivalente est [0, ∞[. Dans ce scénario, I correspond à un événement initial et F à un événement final. La présence de telles contraintes sur le chemin reliant *tl21* à *tb21* fait que le réseau de contraintes temporelles de la figure 6.18 n'impose aucune borne maximale à la durée s'écoulant entre ces deux événements. Il n'est donc pas possible de prouver la propriété sans affiner la démarche.

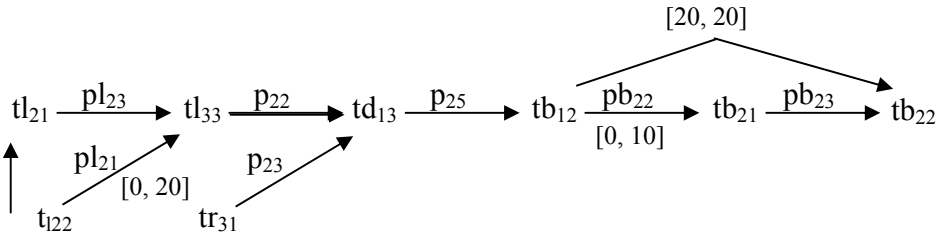


Figure 6.19. Réseau de contraintes temporelles du scénario de la figure 6.17

6.4.2.3 Prise en compte de priorités entre des transitions

Considérons les deux événements *td13* et *tb12*. D'après le scénario obtenu directement par ESA_PetriNet il n'y a pas de contrainte de durée maximale. Pourtant cela correspond à la présence d'un jeton dans la place *p25*, et la spécification dit que si une demande est présente elle doit être traitée pendant le cycle courant. Cela veut dire que la transition *tb12* est prioritaire par rapport à la transition *tb11*. On peut également exprimer cette contrainte sous la forme suivante: "la transition *tb11* ne peut pas être franchie si la transition *tb12* est sensibilisée". Ceci est tout à fait analogue au mécanisme que nous avons décrit dans la section 2 du chapitre 3. Cela veut dire que la transition td13 est nécessairement franchie entre le dernier franchissement de tb11 et celui de tb12. Nous avons donc le réseau de contraintes temporelles modifié de la figure 6.19. Comme il existe un chemin orienté de longueur -20 entre *tb12* et *td13* (celui passant par *tb11*) alors la durée maximale entre *td13* et *tb12* est de 20.

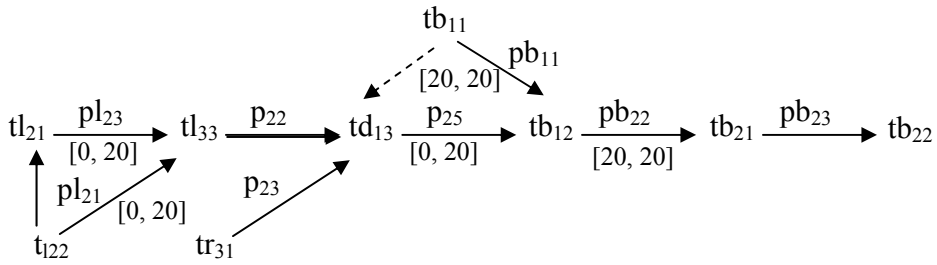


Figure 6.20. Réseau de contraintes temporelles modifié

Un raisonnement équivalent, bien que plus complexe, est possible pour borner la durée entre les franchissements de *tl33* et *td13*. Il est plus complexe car il y a deux places extérieures en entrée de la transition *td13*, les places *p22* et *p23*. Il faudra différencier le cas où la dernière place

marquée est *p22* de celui où c'est *p23*. Par exemple, si l'on suppose que le jeton est arrivé dans *p23* avant *p22*, on trouve que la durée maximale entre tl33 et td13 est de 30.

Finalement le fait qu'entre les événements *tl21* et *tl33* la durée maximale soit de 20 provient du dernier franchissement de *tl22* qui précède *tl21*.

Finalement la durée maximale de ce scénario quand l'arrivée du jeton dans la place p23 précède celle du jeton dans p22 est de 90.

6.5. Conclusion

Nous avons montré dans ce chapitre, en nous appuyant sur des exemples, que l'outil ESA_PetriNet que nous avons développé pouvait être utilisé dans un contexte plus large que celui pour lequel il a été développé : la recherche des scénarios critiques. On peut l'utiliser pour générer des scénarios à partir d'un modèle de système hybride correspondant à un système physique soumis à une commande. La liste des transitions interdites peut être utilisée pour prendre en compte l'effet de la commande sur le système. Une simple exploration arrière suffit pour engendrer tous les scénarios menant à l'état cible. On peut ensuite utiliser une abstraction temporelle pour obtenir des réseaux de contraintes temporelles (un réseau est associé à chaque scénario). Il est ainsi possible d'obtenir le pire cas, et dans des cas simple comme celui que nous venons de traiter sur les trains d'atterrissage, il n'est pas nécessaire d'affecter des valeurs numériques aux paramètres. Au lieu de simplement montrer que la propriété est vraie ou fausse pour un jeu de valeur donné des paramètres, nous sommes capable dans ce cas de donner la (ou les) contraintes que doivent vérifier les paramètres pour que la propriété soit vraie.

Nous avons présenté un deuxième cas d'étude de l'utilisation de notre outil ESA_PetriNet « Extraction & Scenarios Analyser by PetriNet model » pour la vérification des contraintes temporelles. Nous avons appliqué ESA_Petri Net sur l'exemple des calculateurs de vol qui est un système hybride assez complexe. Nous avons vérifié que la durée entre deux commandes émises par deux calculateurs différents est bornée par [dmin, dmax]. Ce qui veut dire que deux commandes ne peuvent jamais être émises simultanément. La démarche consiste à générer dans un premier temps tous les scénarios par une démarche arrière et à vérifier les contraintes temporelles dans un deuxième temps.

Dans ce dernier exemple, nous avons vu que les résultats d'ESA_PetriNet sont une base à partir de laquelle il faut développer une preuve manuelle. Toutefois le problème des priorités entre des transitions est suffisamment proche du problème que nous avons traité dans la section 2 du chapitre 3.

Une fois le réseau de contraintes temporelles construit, les calculs des contraintes temporelles associées aux chemins les plus longs sont des techniques typiques abordées en intelligence artificielle. La base en est l'algorithme classique de Floyd-Warshall [Floyd 62] [Warshall 62].

Conclusion générale

Les travaux présentés dans cette thèse ont contribué à l'analyse des systèmes pilotés par calculateur. Cette analyse qualitative consiste à extraire des scénarios redoutés de ces systèmes complexes et à vérifier certaines contraintes temporelles. Les systèmes pilotés par calculateur, utilisés aussi bien dans les domaines de la défense, du spatial, du nucléaire que dans le secteur automobile et avionique sont des systèmes dynamiques hybrides car ils combinent dans leur structure des technologies mécaniques, hydrauliques, électroniques et informatiques. La dynamique continue est associée à la partie énergétique et la dynamique discrète est liée à la commande numérique et à l'existence d'événements discrets (défaillances, dépassements de seuils). La flexibilité matérielle et logicielle des systèmes pilotés par calculateur, appelés dans certains domaines systèmes embarqués, a permis au concepteur de rajouter progressivement des composants électroniques pour assurer de nouvelles fonctions, ce qui les rend complexes et augmente la difficulté de maîtriser leur fiabilité.

Les études de sûreté de fonctionnement réalisées dès la phase de conception permettent de réduire le coût et le temps de réalisation (le moins de prototypes possible). L'un des moyens efficaces de l'évaluation de la sûreté de fonctionnement des systèmes dynamiques hybrides est la recherche de scénarios redoutés qui sont des événements rares.

La nature dynamique et hybride des systèmes pilotés par calculateur, nous a conduit au choix d'une modélisation associant Réseaux de Petri (RdP) et équations différentielles dans le formalisme Réseau de Petri Prédicats Transitions Différentiels Stochastiques (RdP PTDS). Le modèle RdP décrit le fonctionnement nominal, les défaillances et les mécanismes de reconfiguration. Les équations différentielles représentent l'évolution des variables continues de la partie énergétique du système. L'approche développée par Sarhane Khalfoui, basée sur l'extraction des scénarios redoutés directement à partir d'un modèle RdP a retenu notre attention dans la recherche de tels scénarios qualifiés de « rares » car elle permet d'éviter le problème d'explosion du nombre d'états. Pour cela, nous avons repris cette approche en tenant compte de l'aspect continu par des abstractions temporelles lorsque c'est nécessaire et en caractérisant au mieux le scénario et ses différentes propriétés comme la minimalité.

Bilan

La nouvelle méthode que nous avons proposée prend en compte les conditions associées au franchissement de certaines transitions qui sont des seuils d'évolution des variables continues. Nous avons transformé ces seuils en des intervalles temporels correspondant au temps que met le système pour les atteindre quand les transitions sont sensibilisées. Il s'agit donc d'une «abstraction temporelle». Cette abstraction temporelle transforme le modèle de départ, un Réseau de Petri Prédicats Transitions Différentiels Stochastiques, en un Réseau de Petri arc-pt-temporel.

Le principe consiste ensuite à interdire le franchissement d'une transition sensibilisée si elle est en conflit avec une autre transition sensibilisée de seuil temporel inférieur. Par exemple si dans un RdP nous avons deux transitions t1 et t2 en conflit et telles que le seuil associé à t2 est inférieur à celui associé à t1, le franchissement de t2 est prioritaire. La transition t1 ne sera franchie que si t2 n'est pas franchissable (non sensibilisée), nous avons alors une relation de causalité indirecte entre t2 et t1 qui induit une relation de précédence entre l'événement qui désensibilise t2 et celui qui est associé au franchissement de t1. Dans cette nouvelle approche le scénario correspondant au franchissement de t1 suivi de t2 sera éliminé car il est incohérent vis-à-vis de la dynamique continue du système.

D'autre part, nous avons précisé la notion de scénario en nous appuyant sur la Logique Linéaire, ce qui nous a permis de définir ce qu'était un scénario minimal.

Nous avons automatisé toutes les étapes de notre méthode en développant en Java un premier prototype d'un outil *ESA_PetriNet* (Extraction Scenarios & Analyzer by PetriNet model) qui permet à partir d'un modèle Réseau de Petri d'extraire les scénarios critiques qui mènent vers l'état redouté en tenant compte de l'aspect continu du système. L'interfaçage de notre outil avec l'outil TINA (Time Petri Net Analyzer) lui permet d'utiliser deux de ses fichiers de sortie comme fichier d'entrée. Bien que l'abstraction temporelle la plus naturelle de notre système soit sous la forme d'un RdP arc-pt-temporel, ce modèle doit être transformé en RdP t-temporel, le modèle utilisé par *ESA_PetriNet.*

Nous avons montré dans les cinquième et sixième chapitres que l'Outil *ESA_PetriNet,* développé au départ pour la construction des scénarios redoutés, peut être utilisé dans le contexte de la vérification de certaines propriétés temporelles. Nous avons réalisé l'adaptation en rajoutant une fonction de vérification une fois tous les scénarios générés et un fichier d'entrée contenant la contrainte à vérifier sous la forme de bornes. Les contraintes temporelles que nous avons vérifiées sont la durée maximale d'un scénario et la durée entre deux commandes. Contrairement à la recherche des scénarios redoutés pour laquelle *ESA_PetriNet* nécessite un raisonnement arrière et un raisonnement avant, dans le cas de la vérification, un raisonnement arrière pour l'extraction de tous les scénarios est suffisant. *ESA_PetriNet*, est alors équipé de deux modes pour réaliser ces fonctions. Les scénarios sont représentés sous forme d'un graphe de précédence qui contient toutes les relations de causalité entre les événements (transitions) de l'ordre partiel généré qu'elles soient directes ou indirectes (avec une autre couleur).

Perspectives

D'un point de vue théorique, il faudra mieux caractériser la minimalité des scénarios puis montrer que les mécanismes présents dans la version actuelle de l'algorithme (enrichissement contrôlé du marquage et la liste des transitions interdites) assure la minimalité des scénarios construit. Si cette preuve n'est pas possible, il faudra éventuellement modifier l'algorithme.

De toute façon un certain nombre d'améliorations sont à apporter. Parmi ces points à améliorer dans l'algorithme, le critère d'arrêt en définissant au mieux la notion d'un jeton normal et la complétude d'un scénario. La prise en compte plus fine des boucles est nécessaire dans certains cas pour avoir un scénario complet. L'interfaçage d'ESA_PetriNet avec des outils permettant de calculer la dynamique continue du système lui permettra de travailler sur des intervalles

temporels dynamiques et non pas statiques comme dans la version actuelle (priorité dans le tir des transitions). La partie vérification, peut être améliorée en lui intégrant des algorithmes de recherche des chemins les plus longs.

Nous avons volontairement choisi de rendre ESA_PetriNet compatible avec l'outil TINA. Dans le contexte de la vérification, il devait aussi être possible de comparer les résultats obtenus par les deux outils sur des exemple types. Néanmoins, le fait que le modèle temporel obtenu par abstraction d'un réseau de Petri Prédicats Transitions Différentiels Stochastiques soit un réseau de Petri arc-pt-temporel et non un réseau de Petri t-temporel est source de difficultés. En effet il n'y a pas d'outil permettant de faire des traductions automatiques pour passer d'un type de modèle à un autre. Il n'est même pas sûr qu'ils soient équivalents.

Pour des systèmes plus complexes, une modélisation modulaire est nécessaire. Une approche orientée objet est en cours [Sadou et al 05] pour les systèmes dynamiques hybrides.

L'analyse quantitative est un point important à explorer pour estimer la probabilité d'apparition de l'état redouté. Cette analyse quantitative peut aussi participer à la vérification de l'existance physique des scénarios identifiés.

Glossaire

Gigue: Variation non désirée de l'intervalle entre deux tops successifs d'horloge ou entre deux débuts d'une tâche cyclique.

ABS: Anti Blocking System

AdD : Arbre de Défaillance

AFU : Assistance au Freinage d'Urgence

AMDE : Analyse des Modes de Défaillance et de leurs Effets

AMDEC : Analyse des Modes de Défaillance, de leurs Effets et de leur Criticités

APD : Analyse Préliminaire des Dangers

APR : Analyse Préliminaire des Risques

ASR: Anti Slip Regulation

BAS: Braking Assistance System

CMC: Compositional Model Checking

COTS: Commercial Off The Shelf

CTL: Computation Tree Logic

DDET : Discret Dynamic Event Trees

DdF : Diagramme de Fiabilité

DETAM: Dynamic Event Tree Analysis Method

DFM: Dynamic Flowgraph Methodology

DYLAM: Dynamic Logical Analytical Methodolgy

EBD: Electronics Brakeforce Distribution

EPS : Etudes Probabilistes de Sûreté

EPS: Electronic Stability Program

ESA_PetriNet: Extraction Scenarios & Analyzer by PetriNet model

ESD: Event Sequence Diagrams

Hol: Higher Order Logic

LCF: Logic of Computable Functions

LL: Logique Linéaire

LP; Larch Prover

LTL: Linear Temporal Logic

MCDET: Monte Carlo Dynamic Event Tree

MILL: Multiplicative Intuitionniste Linear Logic

MSR: Motor Schelepp Regulung

PLTL: Propositional Linear Temporal Logic

PSA: Probabilistic Safety Assessment

PVS : Prototype Vérification Système

RdP PTD : Réseaux de Petri Prédicats Transitions Différentiels

RdP PTDS : Réseaux de Petri Prédicats Transitions Différentiels Stochastiques

RdP: Réseau de Petri

RdPSD : Réseaux de Petri Stochastiques et Déterministes

RdPSE : Réseaux de Petri Stochastiques Etendus

RDPSG : Réseaux de Petri Stochastiques Généralisés

SdF: Sûreté de Fonctionnement

SDH:Systèmes Dynamiques Hybrides

STRQDS : Systèmes Temps Réel et Qualité de Service

TCS: Traction Control System

TCTL: Timed Computation Tree Logic

TINA: Time Petri net Analyzer

TLA: Temporal logic of Action

TLP: Temporal Logic Prove

Bibliographie

[Abrial 96] J.R. Abrial, «The B-Book: Assigning Programs to Meanings», Cambridge University Press, 1996.

[Acosta et Siu 93] C. Acosta, Siu. N, « Dynamic event trees in accident sequence analysis: application to steam generator tube rupture », Reliability Engineering and System Safety 41, pp. 135-154, 1993.

[Aldmir 87] T. Aldemir, « Computer-assisted Markov failure modelling of process control systems». *IEEE Trans. Reliab.,* R-36 (1987) 133-144

[Allam 98] M. Allam, «Sur l'analyse qualitative des réseaux de Petri hybrides Une approche basée sur les automates hybrides», thèse de doctorat de l'Institut National Polytechnique de Grenoble, 7 décembre 1998.

[Alur et al 93] R. Alur, C. Courcoubetis, D. Dill, «Model-checking in dense real-time», Information and Computation, 104(1): p. 2-34, 1993.

[Alur et al 95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.H. Ho, X. Nocollin, A. Olivero, J. Sifakis, S. Yovine, «The algorithmic analysis of hybrid systems», Theoretical Computer science, n°138, 1995.

[Andreu 96] David Andreu, «commande et supervision des procédés discontinus», thèse présentée au LAAS le 15 novembre 1996.

[Aralia 96] Groupe Aralia, «Computation of prime implicants of a fault tree within Aralia», Reliability Engineering and System Safty, Special issue on selected papers from ESREL'95, 1996.

[Arnold 90] A. Arnold, «MEC: A System for Constructing and Analysing Transition Systems», Joseph Sifakis, editor, International Workshop on Automatic Verification Methods for Finite State Systems, Lecture Notes in Computer Science, pages 117-132, Gronoble, France, Springer-Verlag, LNCS 407, 1990.

[Arnold et al 94] A. Arnold, D. Begay, P. Crubille, «Construction and analysis of systems with MEC», World Scientific, 1994.

[Arnold et Brlek 95] A. Arnold, S. Brlek, «Automatic Verification of Properties in Transition Systems», Software-Practice and Experience, 25(6): 579-596, 1995.

[Arnold et Nivat 82] A. Arnold, M. Nivat, «Comportements de Processus», Colloque AFCET, Les Mathématique de l'informatique, p 35-68, 1982.

[AtelierB] http://www.atelierb.societe.com/BOILER/UK/main.htm

[Barras et al 99] B. Barras, S. Boutin, C. Cornes, J. Courant, Y. Coscoy, D. Delahaye, D. de Rauglaudre, J.C. Filliatre, E. Giménez, H. Herbelin, G. Huet, H. Lahlhére, C. Munoz, C. Murthy, C. Parent-Vigouroux, P. Loiseleur, C. Paulin-Mohring, A. Saïbi, B. Werner, «The Coq Proof Assistant Reference Manual, Version 6.3», 1999.

[Behnia et Waeselynck 99] S. Behnia, H. Waeselynck, «Test Criteria Definition for B Models», in World Congress on Formal Methods, LNCS 1708, p. 509-529, Toulouse, Springer Verlag, 1999.

[Bel et al 00] G. Bel, F. Boniol, J. Foisseau, « Implantation de lois de commande dans une architecture distribuée : vérification d'exigences temporelles par des méthodes de satisfaction de contraintes », Journées FAC'2000 Formalisation des Activités Concurrentes IRIT-UPS, Toulouse, 18-19 mai 2000.

[Bérard et al 99] B.Bérard, M. Bidoit, F. Laroussinie, A. Petit, P. Schnobelen, «Vérification de logiciels, Techniques et outils du model-checking», Vuibert, Ouvrage Collectif, Coordination Philippe Schnoebelen, 1999.

[Berry et Gonthier 92] G. Berry, G. Gonthier, «The ESTEREL Synchronous programming language: Design, semantics, implementation», Science of Computer Programming, 19(2): p.87-152, 1992.

[Berthomieu et al 03] B. Berthomieu, P.O. Ribet, F. Vernadat, «L'outil TINA Construction d'espaces d'états abstraits pour les réseaux de Petri et réseaux Temporels», Modélisation des Systèmes Réactifs, MSR'2003, Hermes, 2003.

[Berthomieu et Menasche 83] B. Berthomieu, M. Menasche, «An Enumerative Approach for Analyzing Time Petri Nets», IFIP Congress 1983, Paris, 1983.

[Betous 02] Claudia Betous-Almeida, «Construction et affinement de modèles de sûreté de fonctionnement-Application aux systèmes de contrôle-commande», Thèse de Doctorat de l'Institut National Polytechnique de Toulouse, 11 juin 2002.

[Bieber et al 05] P. Bieber, C. Castel, L. Cholvy, H. Demmou, C. Kehren, M. Medjoudj, N. Riviere, C. Seguin, R. Valette, «Qualitative formalisation of critical scenarios with respect to dynamic system models», 6ème Congrès International Pluridisciplinaire Qualité et Sûreté de Fonctionnement (QUALITA 2005), 16-18 Mars 2005, p.583-590, Bordeaux (France), 2005.

[Bogza et al 98] M. Bogza, C. Dawns, O. Maler, A. Olivero, S. Tripakis, S. Yovine, «KRONOS: a model-checking tool for real-time systems», 10th International Conference on Computer Aided Verification (CAV'98), Lecture Notes in Computer Science, pages 546-550, Vancouver, Canada, Springer-Verlag, LNCS 1427, 1998.

[Boniol et Carcenac 02] F. Boniol et F. Carcenac, « Une étude de cas pour la vérification formelle de propriétés temporelles » *Journées FAC,* Toulouse, France, *26-26 March 2002*.

[Boyer et Moor 79] R. Boyer, J. Moore, «A Computational Logic», Academic Press, 1979.

[Bryant 86] R. Bryant, «Graph-based algorithms for boolean function manipulation», IEEE Transactions on Computers, 35(8): p.677-691, 1986.

[Bryant 92] R. Bryant, «Symbolic Boolean manipulation with ordered binary-decision diagrams», ACM Computer Surveys, 24(3):293-318, 1992.

[Buisson 93] J. Buisson, «Analysis of switching devices with Bond Graphs», journal of the Franklin Institute, vol. 330, n°6, p. 1165-175, 1993.

[Burch et al 92] J. Burch, E. Clarke, K. McMillan, D. Dill, L.Hwang, «Symbolic model checking: 10^{20} states and beyond», Information and Computation, 98(2):142-170, 1992.

[Chabot 98] J.L. Chabot, «Approche probabiliste relative à l'étude des scénarios d'incendie», thèse de l'Université de Poitiers-ENSMA, 16 octobre 1998.

[Champagnat 98] R. Champagnat, «Supervision des systèmes discontinus: définition d'un modèle hybride et pilotage en temps-réel », Thèse de Doctorat de l'Université Paul Sabatier de Toulouse, soutenue le 1er octobre 1998.

[Chiol et al 93] G. Chiol, M. A Marsan, G. Balbo, G.Conte, «Generalized Stochastic Petri Nets: A Definition at the Net Level and its Implications», IEEE Transactions on Software Engineering, Vol 19, Issue 2, p. 89- 107, February 1993.

[Clark et Emerson 81] E. Clarke, E. Emerson, «Design and synthesis of synchronization skeletons using branching time temporal logic », logics of programs workshop, p. 52-71, Yorktown Heights, NY, USA, Springer-Verlag, 1981, LNCS 131.

[Clarke et al 94] E. M. Clarke, O. Grumberg, D. E. Long, «Model Checking and Abstraction», ACM Transaction on Programming Languages and Systems (TOPLA), 16 (5): 1512-1542, 1994.

[Closse et al 01] E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, Yovine, «TAXYS : a Tool for the Development and Verification of Real-Time Embedded Systems», 13th International Conference on Computer Aided Verification (CAV 01), pages 391-395, Paris, France, Springer-Verlag, 2001, LNCS 2102.

[Crow et Vito 96] J.Crow, B.D. Vito, «Formalizing Space Shuttle software requirements», 1st Workshop on Formal Methods in Software Practice (FMSP), p. 40-48, ACM, 1996.

[David et Alla 89] R. David, H. Alla, «Du Grafcet aux réseaux de Petri», Hermes, 1989.

[Daws et Yovine 96] C. Daws, S. Yovine, «Reducing the number of clock variables of timed automata», 7th IEEE Real Time Systems Symposium, RTSS'96, pages 73-81, IEEE Computer Society Press, Washington, DC, USA, 1996.

[De Bruijn 68] N. De Bruijn, «The mathematical language AUTOMATH, its usage, and some of its extensions», Symposium on Automatic Demonstration, volume 125 de Lecture Notes on Mathematics, pages 29-61, LNM 125, Versailles, France, Springer-Verlag, 1968.

[Devooght et Smidts 94] J. Devooght, C. Smidts, « A theoretical analysis of DYLAM-type event tree sequences », Proceedings of PSAM-II, Volume 1, pp. 011.1-011.6, 1994.

[Dufour et Dutuit 02] F. Dufour, Y. Dutuit, «Dynamic reliability: a new model», Lambda- Mu 13-Esrel 2002 European Conference. 2002.

[Dugan et al 84] J.B. Dugan, K.S. Trivedi, R.M. Geist, V.F. Nicola, «Extende Stochastic Petri Nets: Applications and Analysis», 10th International Symposium on Computer Performance, p. 507-519, 1984.

[Dutertre 00] B. Dutertre, «Formal analysis of the priority ceiling protocol», IEEE Real Time systems Symposium (RTSS'00), pages 151-160, 200, Orland, FL. 2000.

[Emerson et Clarke 80] E. Emerson, E. Clarke, «Characterizing correctness properties of parallel programs using fixpoints», proceedings of Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, and (Pays-Bas), p. 169-181, 1980.

[Engberg 95] U. Engberg, «Reasoning in the Temporal Logic of Actgions», thèse de doctorat, University of Aarhus, Department of Computer Science, Denmark, 1995.

[Fink et Bishop 97] G. Fink, M. Bishop, «Property-Based Testing: A new Approach to Testing for Assurance», ACM SIGSOFT, Software Engineering Notes, 22(4), p. 74-80, 1997.

[Floyd 62] Robert W Floyd, «Algorithm 97: Shortest path», Communications of the ACM Vol.5 Issue 6, (June 1962), page.345.

[Garland et Guttag 91] S. Garland, J. Guttag, «A guide to LP, the Larch Prover», Rapport technique 82. Digital Equipement Corporation, Systems Research Center, 1991.

[Garret 93] C.J. Garret, S.B. Guarro, G.E. Apostolakis, «Development of a methodology for assessing the safety of embedded software systems», American institute of Aeronautics and Astronautics, 1993.

[Girard 87] J.Y. Girard, « Linear Logic », Theoretical Computer Science, 50, p.1-102, 1987.

[Girault 97] F. Girault, « Formalisation en Logique Linéaire du fonctionnement des réseaux de Petri », Thèse de Doctorat, N°2870, Université Paul Sabatier, Toulouse, 1997.

[Gordon et al 79] M. Gordon, R. Milner, C. Wadsworth, Edinbourgh «LCF: A Mechanised Logic of Computation», Springer-Verlag, LNCS 78, 1979.

[Gordon et Melham 93] M. Gordon, T. Melham, «Introduction to HOL: A Theorem Proving Environment for Higher Order Logic», Cambridge University Press, 1993.

[Griffault et al 98] A. Griffault, S. Lajeunesse, G. Point, A. Rauzy, J.P. Signoret, and P. Thomas, «The AltaRica language», In Proceedings of International Conference on Safety and Reliability, ESREL'98. Balkema Publishers, June 20-24 1998.

[Guttag et al 85] J. Guttag, J. Horning, J. Wing, «The LARCH family of specification languages», IEEE Software, 2(5):24-36, 1985.

[Harel 87] D. Harel, «Statecharts: A visual Formalism for complex systems », The Science of Computer Programming, p.231-274, 1987.

[Havelund et al 01] K. Havelund, M. Lowry, J. Penix, «Formal Analsys of a Space-Craft Controller Using SPIN», IEEE Transactions on Software Testing, Verification, and Reliability, 7 PP. 19-33, 1997.

[Havelund et al 97] K. Havelund, A. Skou, K. Larsen, Lund, «Formal modelling and analysis of an audio/video protocol: An industrial case study using UPPAAL», 18th IEEE Real-Time Systems Symposium (RTSS'97), p. 2-13, San Francisco, CA, USA, IEEE Computer Society Press, 1997.

[Hélias et al 04] A. Hélias, F. Guerrin, J.-P.Steyer, «Abstracting continuous system behaviours into timed automata: Application to diagnosis of an anaerobic digestion process», Proc DX-2004, 5th International Workshop on Principles of Diagnosis, Carcassonne (France), June 23-25 2004.

[Henzinger et al 97] T.A. Henzinger, P.H. HO, H. Wong Toi, «HyTech: A Model Checker for Hybrids Systems», International Jornal on Software Tools for Technology Transfer, 1 (1):110-122, Springer-Verlag, 1997.

[Hofer et al 01] E. Hofer, M. Kloos, B. Krzykacz-Hausmann, J. Peschke, M. Sonnenkalb, « Methodenentwicklung zur simulativen Behandlung der Stochastik in probabilistischen»

Sicherheitsanalysen der Stufe 2, Abschlußbericht, GRS-A-2997, Gesellschaft für Anlagen- und Reaktorsicherheit, Germany (2001).

[Holzmann 97] G. J. Holzmann, «The model-checker SPIN», IEEE Transcripts on Software Engineering, 23 (5): 279-295, 1997.

[Holzmann et Peled 94] G.J. Holzmann, D. Peled, «An Improvement in Formal Verification», 7th IFIP WG6, International Conference on Formal Description Techniques (FORTE 1994), p. 197-211, Bern, Suisse, 1994.

[Jensen 92] K. Jensen, «Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use», Vol. 1, Basic Concepts, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1992.

[Kassaagi 01] M.O. Kassaagi, «Caractérisation expérimentale du comportements des conducteurs en situation d'urgence pour la specification de systèmes de sécurité active», thèse de l'Ecole Centrale Paris, 11 juillet 2001.

[Kehren et al 04] C. Kehren, C. Seguin, P. Bieber; C. Castel, C. Bougnol, J.P. Heckmann, S. Metge, «Advanced Multi-System Simulation Capabilities with AltaRica», proceedings of Safety Critial System Conference, Rhodes Island, USA, august 2004.

[Kehren et Seguin 03] C. Kehren et C. Seguin, «Evaluation qualitative de systèmes physiques pour la sûreté de fonctionnement», Journées FAC'2003, Formalisation des Activités Concurrentes à l'IRIT, Toulouse, les 12 et 13 mars 2003.

[Kermisch et Labeau 02] C. Kermisch et P.E. Labeau, «Approche dynamique de la fiabilité des systèmes», Rapport MNFD 2002-10, Service de Métrologie Nucléaire, Université Libre de Bruxelles (Belgique), novembre 2002.

[Khalfaoui 03] Sarhane Khalfaoui «Méthode de recherche des scénarios redoutés pour l'évaluation de la sûreté de fonctionnement des systèmes mécatroniques du monde automobile», thèse de Doctorat, No 03574, Institut National Polytechnique, Toulouse, 26 septembre 2003.

[Kropf 99] T. Kropf, «Recent advancements in hardware verification-how to make theorem proving fit for an industrial usage», 12th International Conference on Theorem Proving in Higher Order Logics (TPHOls'99), pages 1-4, Springer-Verlag, LNCS 1690, 1999.

[Laprie et al 04] J.C. Laprie, «Sûreté de fonctionnement des systèmes : concepts de base et terminologie », Rapport LAAS N°04520, 2004.

[Laprie et al 95] J.C Laprie, J. Arlat, J.P. Blanquart, A. Costes, Y. Crouzet, Y. Destwarte, J.C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac et P. Thévenod, «Guide de la Sûreté de Fonctionnement», Cépaduès Editions, 1995.

[Laprie et al 96] J.C. Laprie, J. Arlat, J-P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J-C. Fabre, H. Guillermain, M. Kaaniche, C. Mazet, D.Powell, C. Rabéjac et P. Thévenod., «Guide de la Sûreté de Fonctionnement », 2ème édition (Cépaduès), ISBN : 2.85428.382.1,1996

[Laroussinie et Larsen 98] F. Laroussinie, K.G. Larsen, «CMC: A tool for compositional model-checking of real-time systems». In Proc. IFIP Joint Int. Conf. Formal Description Techniques & Protocol

Specification, Testing, and Verification (FORTE-PSTV'98), pages 439-456. Kluwer cademic Publishers, 1998.

[Larsen et al 97] K. Larsen, P. Pettersson, Yi W, «UPPAAL in a nutshell», International Journal of Software Tools for Technology Transfer, 1(1-2): pp. 134-152, 1997.

[Lewerentz et Lindner 95] C. Lewerentz, T. Lindner, «Formal development of reactive systems: Case study production cell», LNCS 891, Springer-Verlag, Janvier 1995.

[Loeffler et Serhouchni 97] S. Loeffler, A. Serhouchni, «Creating a validation implementation of the Steam boiler control», 3rd SPIN Workshop,Enschede, The Netherlands, 1997.

[Loures et Pascal 04] E. Rocha Loures, J.C. Pascal, «Abstraction de la dynamique continue par réseau de Petri temporel flou pour la détection et diagnostic de systèmes hybrides», 12èmes Rencontres Francophones sur la Logique Floue et ses Applications (LFA'2004), p. 321-328, Nantes (France), 18-19 Novembre 2004.

[Lunze 99] J. Lunze, «A timed discrete-event abstraction of continuous-variable systems», International Journal of Control, 72(13): p. 1147-1164, 1999.

[Marsan et al 84] M.A Marsan, G. Balbo, G.Conte, «A class of generalised stochastic petri nets for the analysis of multiprocessor systems», ACM Trans, On Computer Systems, 2(1) May 1984

[Marsan et Chiola 86] M.A. Marsan, G. Chiola, «On Petri Nets with Deterministic and Exponentially Distributed Firing Times», 7th European Workshop on Applications and Theory of Petri Nets, p. 132-145, Springer-Verlag, Lecture Notes in Computer Science 266, Advances in Petri Nets, 1987, June 1986.

[McMillan 93] K. McMillan «Symbolic Model Checking», Kluwer Academic Publishers, Boston 1993.

[Medjoudj 04] M. Medjoudj, «Extraction des scénarios critiques pour l'évaluation de la sûreté de fonctionnement des systèmes mécatroniques. Approche hybride basée sur un modèle réseau de Petri et la logique linéaire», 5ème Congrès de l'Ecole Doctorale Systèmes (EDSYS), Toulouse (France), 13-14 Mai 2004.

[Medjoudj et al 04a] M. Medjoudj, S.Khalfaoui, H. Demmou, R. Valette, «Un algorithme pour l'extraction des scénarios critiques dans les systèmes hybrides», Journées "Formalisation des Activités Concurrentes" (FAC'2004), Toulouse (France), 9-10 Mars 2004.

[Medjoudj et al 04b] M. Medjoudj, S.Khalfaoui, H. Demmou, R. Valette, «A method for deriving feared scenarios in hybrid systems», Probabilistic Safety Assessment and Management (PSAM'7 - ESREL'04), Berlin (Allemagne), 14-18 Juin 2004.

[Merz 01] S. Merz, «Modeling and Verification of Parallel Processes», Model Checking:A Tutorial Overview , pages 3-38, Springer-Verlag, F. Cassez et al ; (ed), LNCS 2067, 2001.

[Molloy 82] M.K. Molloy «Performance analysis using stochastic Petri nets», IEEE Trans on Computer vol C-31, N°9, p.913-917, Sept 1982.

[Moncelet 98] G. Moncelet, «Application des réseaux de Petri à l'évaluation de la sûreté de fonctionnement des systèmes mécatroniques du monde automobile», Thèse de Doctorat, N°3076, Université Paul Sabatier, Toulouse, 9 octobre 1998.

[Monin 00] J.F. Monin, «Comprendre les méthodes formelles, Panorama et outils logiques», Masson& CENT-ENST, 2nd édition, 2000.

[Natkin 80] S.O. Natkin, «Les réseaux de Petri stochastiques et leur application à l'évaluation des systèmes informatiques», thèse de Docteur Ingénieur, CNAM Paris, juin 1980.

[Owre et al 95] S. Owre, J. Rushby, N. Shankar, F. Von Henke, «Formal Verification for Fault-Tolerant Architectures: Rolegomena to Design of PVS», IEEE Trans. On Software Engineering, 21(2):107-125, 1995.

[Pagani 96] F. Pagani. «Partial orders and verification of real-time systems», In B. Jonsson and J. Parrow, editors, Proceedings of the Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, volume LNCS 1135, pages 327-346. Springer Verlag, Uppsala, Sweden, 1996.

[Pagetti 04] C. Pagetti, «Extension temps réel d'AltaRica» Thèse de Doctorat de l'Ecole Centrale de Nante et l'Université de Nante, 20 avril 2004.

[Paulson 94] L. Paulson, «Isabelle: A Generic Theorem Prover», LNCS 828, Springer-Verlag, 1994.

[Pnueli 77] A. Pneuli, «The temporal logic of programs», proceedings of 18th IEEE Symposium on Foundation of Computer Science, Providence (USA) p. 46-57, 1977.

[Pnueli 81] A.Pnuelli, «The temporal semantics of concurrent programs», Theoretical computer Science, p. 45-60, 1981.

[Pradin et al 99] B. Pradin-Chézalviel, R. Valette, L.A Künzle, «Scenario duration characterization of t-timed Petri nets using linear logic», IEEE PNPM'99, 8th International Workshop on Petri Nets and Performance Models, p.208-217, Zaragoza, Spain, September 6-10, 1999.

[Prasetya 95] W. Prasetya, «Mechanically Supported Design of Self-stabilizing Algorithms», thèse de doctorat, Institute for programming research and Algorithmics (IPA), Utrecht University 1995.

[Rauzy et Dutuit 97] A. Rauzy, Y. Dutuit, «Exact and truncated computations of prime implicants of coherent and non coherent fault trees within Aralia», Reliability Engineering & System Safety, Vol. 58, 1997.

[Ribert et al 02] P. Ribert, F. Vernadat, B. Berthomieu, «On Combining Persistent Sets Method with the Covering Steps Graph Method», 22th IFIP WG6.1 International Conference on Formal Techniques for Networked and Sistributed Systems (FORTE' 2002), pages 344-359, Springer-Verlag, LNCS 2529, 2003.

[Rivière 03] N. Rivière, «Modélisation et analyse temporelle par réseaux de Petri et logique linéaire», thèse de l'INSA de Toulouse, le 26 novembre 2003.

[Rivière et al 05] N. Riviere, H. Demmou, R. Valette, M. medjoudj, «Symbolic temporal constraint analysis, an approach for verifying hybrid systems», 16th IFAC World Congress, Prague (République Tchèque), 3-8 Juillet 2005.

[Rushby 93] J. Rushby, «Formal Methods and the Certification of Critical Systems», Rapport technique, Computer Science Laboratory, SRI Int, CSL-93-7, 1993.

[Rushby 99] J. Rushby, «Integrated formal verification: Using model checking with automated abstraction, invariant generation and theorem proving», Theoretical and Practical Aspects of SPIN Model Checking: 5th and 6th Int. SPIN Workshops, pages 1-11, Springer-Verlag, LNCS 1680, 1999.

[Sadou et al 05] N. Sadou, H. Demmou, J.C. Pascal, R. Valette, « Object oriented approach for deriving feared scenarios in hybrid systems t », European Simulation and Modelling Conference, p.572-578, Porto (Portugal), 24-26 Octobre 2005.

[Schnoebelen 99] P. Schnoebelen, «Vérification de Logiciels: Techniques et outils du model-checking», ouvrage collectif, coordination P. Schnoebelen, Vuibert, ISBN 2-7117-8646-3, , Paris, 1999.

[Schoenig 04] R. Schoenig, «Définition d'une méthodologie de conception des systèmes mécatroniques sûrs de fonctionnement», Thèse de Doctorat de l'Institut National Polytechnique de Lorraine, 26 octobre 2004.

[Sinnamon et Andrews 97] R.M. Sinnamon & J.D. Andrews, «New approaches to evaluating fault trees», Reliability Engineering & System Safety, Vol. 58, 1997

[Smidts et Devooght 92b] [Kermisch et Labeau 02], J. Devooght, [Kermisch et Labeau 02] «Probabilistic Reactor Dynamics - II: A Monte-Carlo Study of a Fast Reactor Transcient», Nuclear Science and Engineering, Vol. 111, p241-256, 1992.

[Sofia] logiciel développé par la société SGTE Sofreten de génération automatique d'arbre de dfaillance et de l'AMDE associée

[Stauner et al 97] T. Stauner, O. Müller, M. Fuchs, «Using HYTECH to verify an automotive control systems», O. Maler, editor, International Workshop on Hybrid and Real-Time Systems (HART 97), pages 139-153, Springer-Vzrlag, 1997, LNCS 1201, 1997.

[STRQDS 02] STRQDS, « *Présentations d'études de cas* », Octobre 2002. http://www.laas.fr/strqds.

[Valette 92] R. Valette, «Les réseaux de Petri», support de cours, France, 1992.

[Villani et al 03] E. Villani, J.C. Pascal, P.E. Miyagi and R. Valette, « Differential predicate transition Petri nets and objects, an aid for proving properties in hybrid systems », *ADHS 03, IFAC Conference on Analysis and Design of Hybrid Systems*, p. 117–122, Saint-Malo, France, 16-18 June 2003.

[Villemeur 88] A. Villemeur, «Sûreté de fonctionnement des systèmes industriels», collection de la Direction des Etudes et Recherche d'Electricité de France, 1988.

[Villén-Altamirano 94] M. Villén-Altamirano J. Villén-Altamirano, « RESTART: A straightforward method for fast simulation of rare events », Proceedings of the 1994 Winter Simulation Conference, p. 282-294, 1994

[Vincent 03] A. Vincent, «The MEC V compiler», Available at http://altarica.labri.fr, 2003

[Waeselynck et Fosse 99] H. Waeselynck et P. Thévenod-Fosse, «A Case Study in Statical Testing of Reusable Concurrent Objects», in Proc, 3rd European Dependable Computing Conference (EDCC-3), Prague, Czech Republic, Springer, LNCS 1667, pp. 401-18, September 1999.

[Waeselynk 93] H. Weaselynck, «Vérification des logiciels critiques par le test statique» Thèse de doctorat, Institut National Polytechnique de Toulouse 1993.

[Warshall 62] Stephen Warshall, A theorem on Boolean matrices. Journal of the ACM Vol.9 Issue .1, pages 11-12, January 1962

[Wensly et al 78] J. Wensley, L. Lamport, J. Goldberg, M. Green, K. Levitt, P. Melliar-Smith, R. Shostack, C. Weinstock, «SIFT: The Design and Analysis of a Fault-Tolerant Computer for Aircraft Contol», Proceedings of the IEEE, 66(10) : 1240-1255, 1978.

[Weyuker 82] E. Weyuker, «On Testing Non-Testable Programs», The Computer Journal, 25(4), 1982.

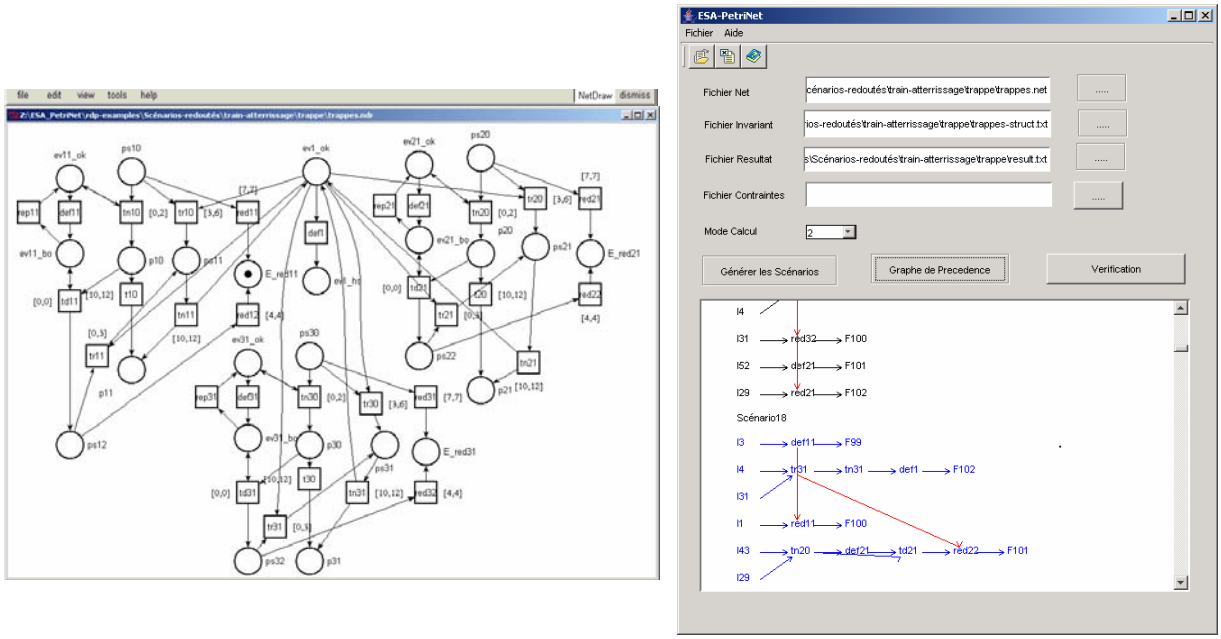
[Yovine 97] S. Yovine, «KRONOS: a verification tool for real-time systems», International Journal of Software Tools for Technology Transfer, 1(1/2):123-133, 1997.

[Zaytoon 01] J. Zaytoon, «Systèmes dynamiques hybrides», ouvrage collectif sous la direction de, collection Hermes Science, ISBN 2-7462-0247-6.

Manuel d'utilisation d'ESA_PetriNet : Extraction Scenarios & Analyzer by PetriNet model

Vue générale de l'outil

ESA_PetriNet est interfacé avec Tina (Time Petri Net Analyzer version 2.7.4) (figure 1). Après édition de réseau de Petri du système sur l'éditeur graphique (ndr) de TINA, on génère deux fichiers d'entrée : un *fichier.net* qui est un fichier descriptif du réseau de Petri (RdP) et un *fichier-struxt.txt* pour les invariant. On introduit ces deux fichiers dans l'interface d'ESA_PetriNet et on donne un non pour le fichier résultats, ensuite on clique sur le bouton générer scénarios et puis sur le bouton graphe de précédence pour afficher ces scénarios sous forme de graphe de précédence. Pour générer les scénarios redoutés on choisi le mode de calcul 2 et le mode de calcul 1 pour les scénarios de vérifications.



TINA ⟶ Fichiers d'entrées ⟶ ESA_PetriNet ⟶ Scénarios

Figure1. Vue générale d'ESA_PetriNet

Editer le RdP sur TINA

Le RdP édité sur TINA prend comme marquage initial l'état redouté. Pour l'exemple des trappes des trains d'atterrissage (chapitre5), le fichier d'édition correspond à *trappe.ndr* (figure2).

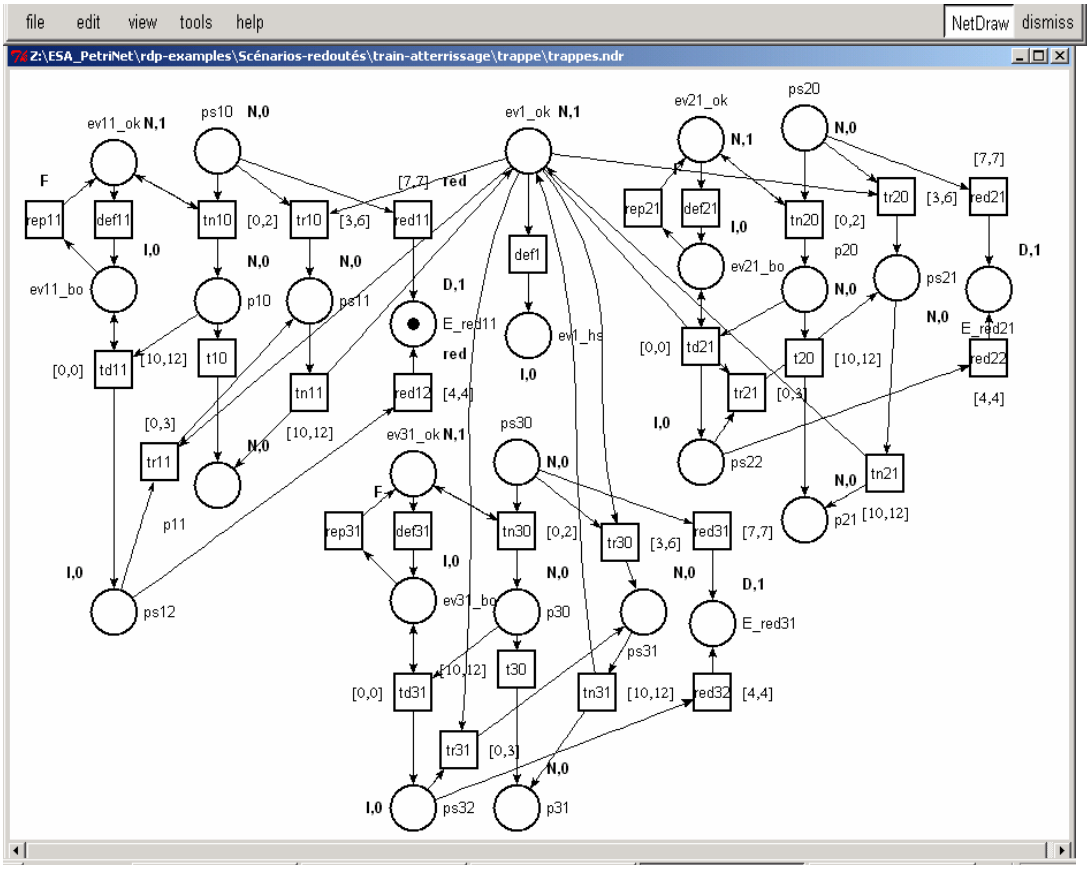


Figure2. *trappe.ndr*

Chaque place possède trois attributs (figure3). Par exemple la place *E_red11*, a les attributs suivants : *name (E_red11), label (D,1), marking (1)*. Le *D* pour défaillance (les places normales ont le label *N*) et le *1* pour le calcul de la composante conservative de l'invariant de places auquel elle appartient. La génération des invariants (figure 8) montre que *E_red11* fait parti de l'invariant « *E_red11, p10, p11, ps10, ps11, ps12* », et comme la composante sa conservative est égal à *1*et le label *E_red11* est de *1* nous avons alors associé des *0* au autres places de cet invariant.

Chaque transition possède trois attributs (figure3). Par exemple la transition *tn10,* a les attributs suivants : *name (tn10)*, *interval ([0 2])*. Pour cette transition l'attribut label est vide, cet attribut est utilisé uniquement dans le cas des transitions que l'on veut interdire (label *F*) comme le cas de la transition *rep11 (rep 21, rep31)* pour éviter les boucles et red11 (red12) à qui on a attribué le label *red* pour redouter pour ce qui permet de sélectionner les scénarios redoutés parmi tous les

scénarios générés. Un clic droit sur la place (ou transition) permet l'affichage de la fenêtre correspondante.

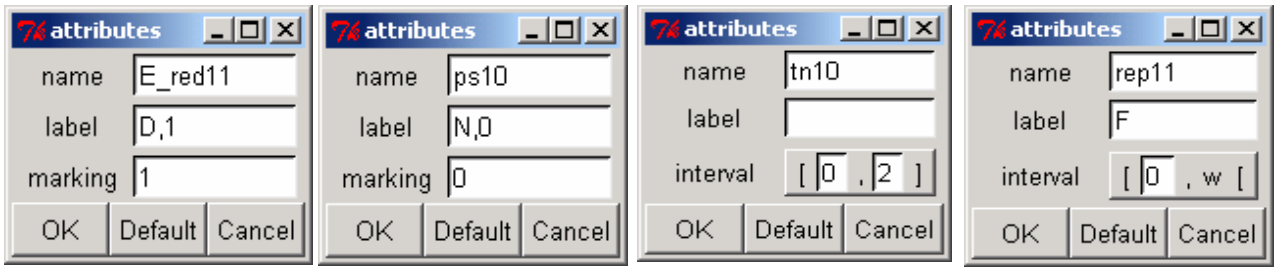


Figure3 : Attributs des places et transitions

Pour afficher les différents labels (figure 4), il faut activer *labels* dans *view* de la barre d'outils.

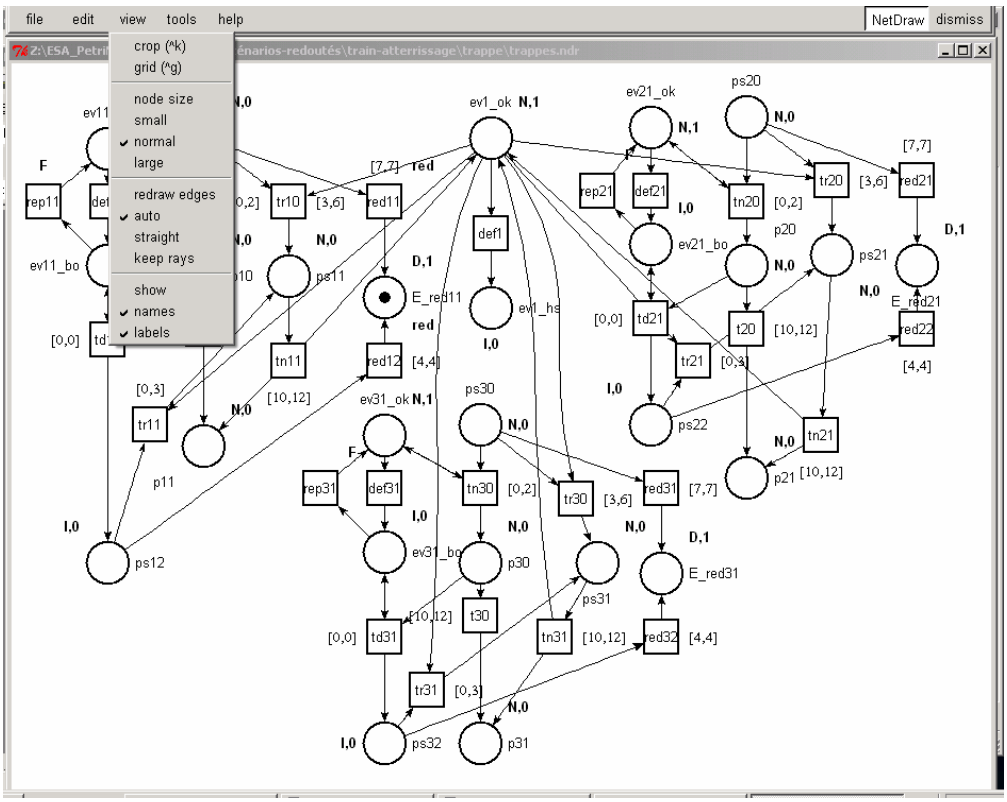


Figure4. Afficher les labels

Générer le fichier.net

Cliquer sur *textify* dans la barre d'outils (figure 5) pour générer le fichier *trappe.net* (figure6).

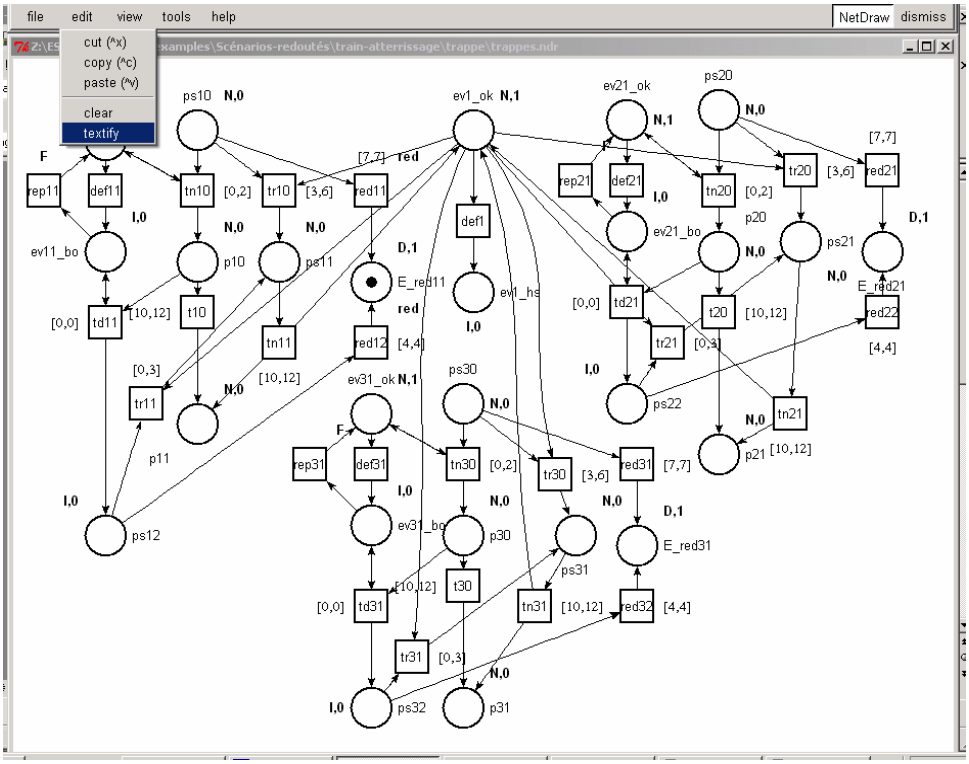


Figure 5. Générer le fichier *trappe.net*

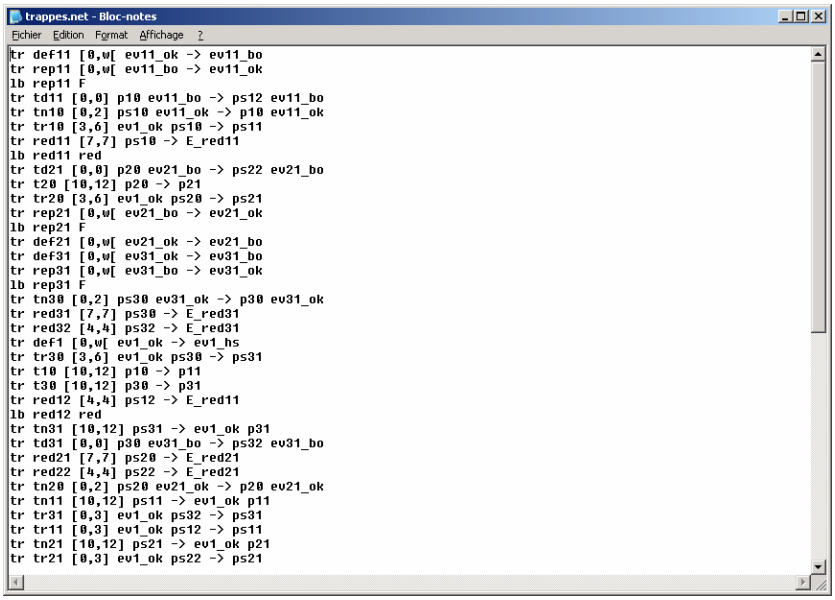


Figure 6. Le fichier *trappe.net*

Générer le fichier-struct.txt

Cliquer sur *structural analysis* dans la barre d'outil puis *ok* (figure 7) pour générer le fichier *trappe-struct.txt* (figure 8).

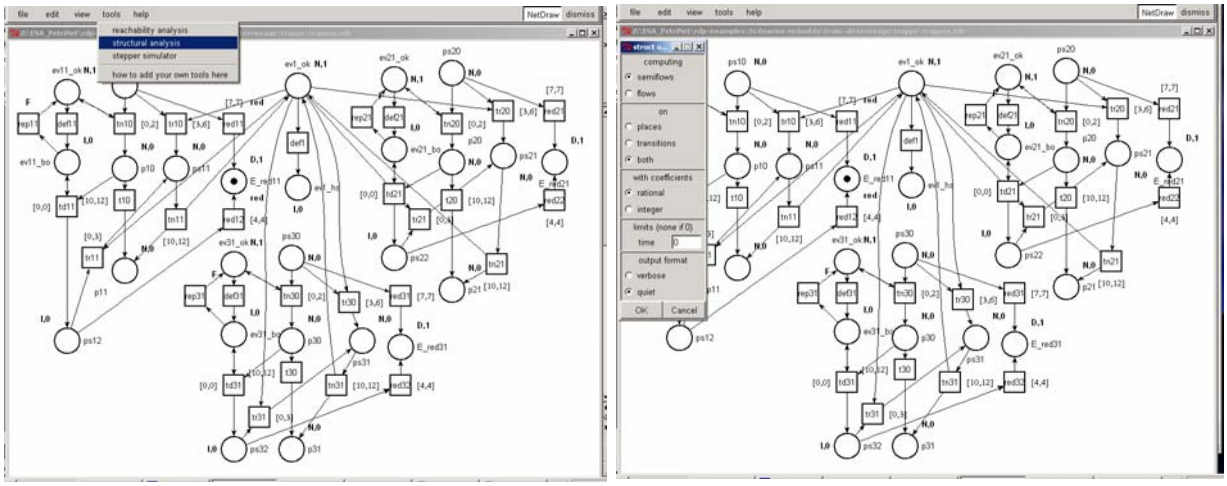


Figure 7. Générer le fichier *trappe-struct.txt*

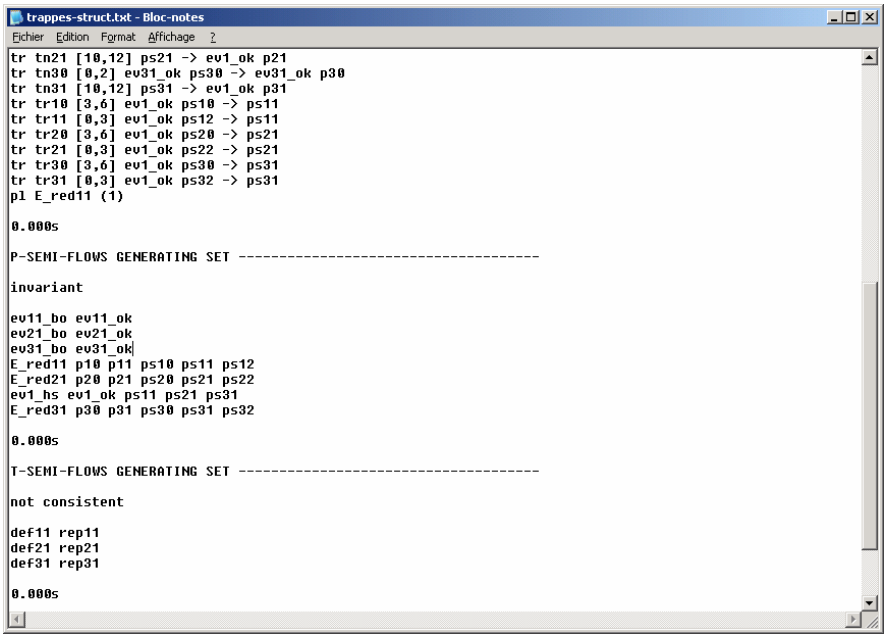


Figure 8. Le fichier *trappe-struct.txt*

Générer les scénarios avec ESA_PetriNet

Une fois, on a chargé les fichiers *trappe.net*, *trappe-struct.txt* et le fichier résultat *trappe-result.txt* (figure 9), on clique sur le bouton *générer scénarios* puis sur *graphe précédence* pour les afficher sous forme de graphe de précédence. Dans cet exemple le mode de calcul choisi est 2 pour les scénarios redoutés.

Nous avons choisi d'afficher les scénarios redoutés en couleur bleue et les scénarios de fonctionnement normal et reconfiguration en couleur noire. Les relations de précédence indirecte sont affichées en rouge.

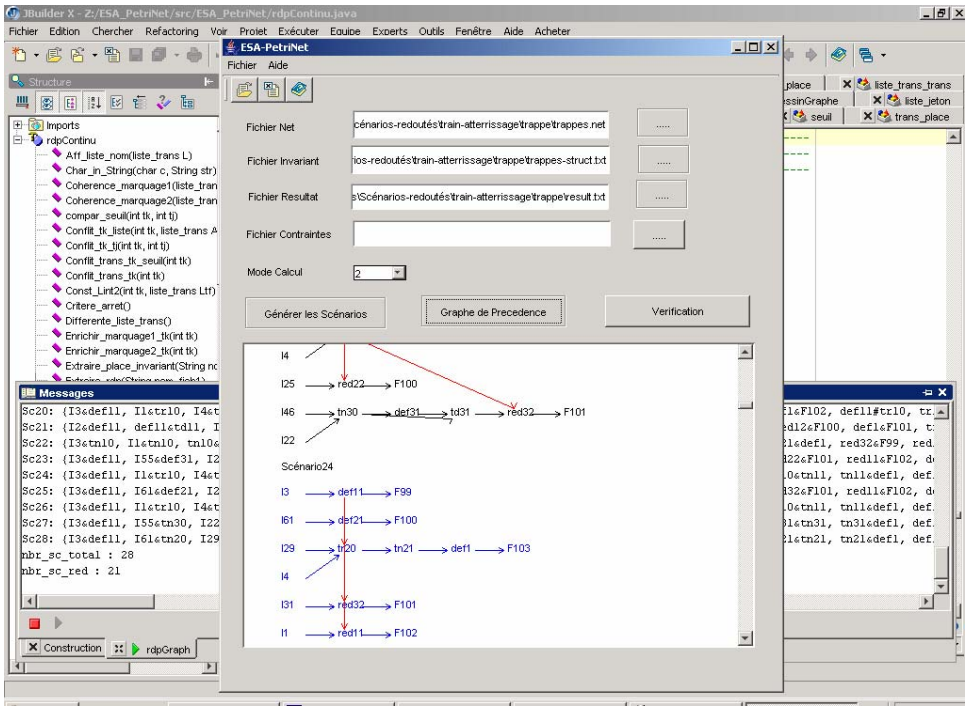


Figure 9. Génération scénarios avec ESA_PetriNet

Utilisation d'ESA_PetriNet pour la vérification

Dans ce cas toutes les places doivent avoir le label *D* puisque la démarche est en arrière, car si on met le *N*, l'algorithme détecte des places de fonctionnement normal et il arrête le raisonnement avant d'avoir le scénario complet. La figure 10 représente le RdP de l'entrée et la sortie du train d'atterrissage étudié dans le chapitre 6. Le mode de calcul utilisé pour générer les scénarios est *1*(figure 11).

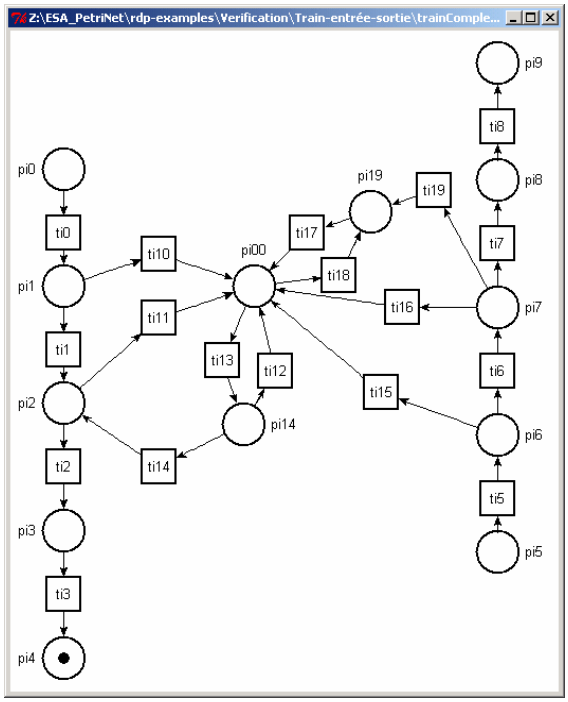


Figure 10. RdP pour l'entrée et la sortie d'un train d'atterrissage

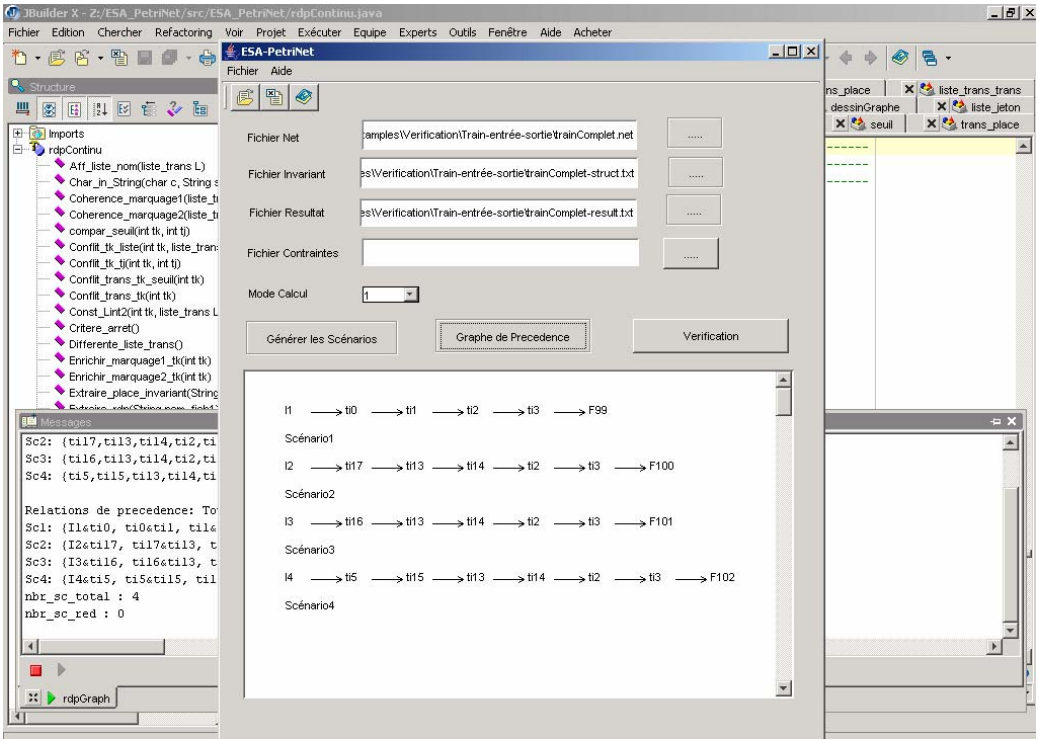


Figure 11. Génération de scénarios pour la vérification

Contribution à l'analyse des systèmes pilotés par calculateur: Extraction de scénarios redoutés et vérification de contraintes temporelles

L'intégration progressive de l'électronique dans les secteurs automobile et avionique a amélioré le confort et les services rendus. Toutefois, cela a complexifié la conception des systèmes pilotés par calculateurs (systèmes mécatroniques, calculateurs de vol, etc.), ce qui rend difficile la maîtrise de leur fiabilité. Par ailleurs, la phase de conception doit être rapide et peu coûteuse (le moins de prototypes possible, le plus tard possible) avec un niveau de sécurité garantie. De plus, les ressources en moyens matériels étant limitées, pour des raisons de coûts et de mise en œuvre, les concepteurs évitent au maximum les redondances matérielles. Des études de Sûreté de Fonctionnement réalisées dès la phase de conception permettent une meilleure maîtrise des risques et de la fiabilité des systèmes conçus. En effet, les points faibles qui sont mis en évidence lors de l'évaluation du niveau de sûreté des systèmes conçus permettent aux concepteurs de spécifier des stratégies de pilotage et des modes de reconfiguration avant les premiers essais sur un prototype réel. Les systèmes pilotés par calculateurs combinant des technologies mécaniques, hydrauliques, électroniques et informatiques sont hybrides: la dynamique continue est associée à la partie énergétique et la dynamique discrète est liée à la commande numérique et à l'existence d'événements discrets (défaillances, dépassements de seuils). L'étude de la sûreté de fonctionnement de tels systèmes doit nécessairement tenir compte des interactions existantes entre leurs paramètres physiques (température, pression, vitesse…) et le dysfonctionnement de leurs composants. Les méthodes classiques de la sûreté de fonctionnement, comme les arbres de défaillances sont insuffisantes pour de tels systèmes complexes et hybrides car ils sont dynamiques. La sûreté de ces systèmes doit tenir compte du temps et de l'ordre d'apparition des événements. Pour résoudre le problème de la rareté de ces scénarios auquel sont exposées les méthodes basées sur la simulation, des techniques d'accélération de la simulation ont été développées et largement utilisées, avec succès, dans l'ingénierie nucléaire notamment. Les méthodes d'analyse de modèles à événements discrets (automates, Réseau de Petri) ont leur contribution dans ce domaine mais l'utilisation de graphe d'accessibilité est vite confrontée au problème d'explosion combinatoire.

Mes travaux de thèse sont placés dans le cadre de la fiabilité dynamique. L'objectif est de réaliser une analyse qualitative de la sûreté de fonctionnement des systèmes pilotés par calculateurs pour extraire des scénarios menant à des états redoutés. Il s'agit de caractériser ces scénarios au plus tôt dans la phase de conception, ce qui permet d'évaluer leurs probabilités d'occurrence pour valider l'architecture du système. Nous proposons une approche basée sur la logique linéaire et les Réseaux de Petri Prédicats Transitions Différentiels Stochastiques (RdP PTDS) qui garantissent le respect de la nature hybride de ces systèmes. Cette approche tient partiellement compte de l'aspect continu du système et plus particulièrement des seuils associés à certaines transitions dans le modèle RdP. Cela permet de déterminer plus précisément les conditions exactes de l'occurrence de l'événement redouté : ce qui pousse le système à quitter son fonctionnement normal et à évoluer vers l'état redouté. L'originalité de notre approche est que l'ordre d'occurrence des événements est pris en compte et les scénarios incohérents vis-à-vis de la dynamique continue du système sont éliminés. Notre approche est également orientée vers la vérification de certaines propriétés des systèmes pilotés par calculateur. Ces propriétés peuvent êtres de type temporel (la durée maximale d'un scénario ou la durée entre deux commandes) ou de type accessibilité entre deux états. L'automatisation de toutes les étapes de notre approche nous a paru indispensable dans le cas des systèmes complexes où le risque d'erreur humaine est très important. C'est pourquoi, j'ai développé un outil ESA_PetriNet (Extraction & Scenarios Analyser by PetriNet model) qui permet d'extraire les scénarios critiques qui mènent vers l'état redouté à partir d'un modèle Réseau de Petri et de vérifier certaines propriétés des systèmes pilotés par calculateurs.

Mots clés *: Sûreté de fonctionnement, systèmes pilotés par calculateurs, fiabilité dynamique, vérification de propriétés, systèmes dynamique hybride, réseau de Petri, Logique Linéaire.*

Contribution to the analysis of computer-controlled systems: Extraction of feared scenarios and checking of temporal constraints

The progressive integration of electronics in the car and avionics fields has lead to improvements in both functions and services. However, this has caused an increased complexity in the design of these systems, typically involving computers (e.g. mecatronic systems, flight computers, etc.), which makes the control of their reliability difficult. In addition, the phase of design must be fast and inexpensive (i.e. less prototypes and at later stages) with a level of guaranteed safety. In more cases for reasons of cost and implementation, material resources are limited and the system designers must avoid component redundancies within the system as much as possible. Reliability studies performed at the design phase have allowed a better control of the risks and reliability of the conceived systems. Indeed, the weak points which are highlighted during the evaluation of the level of safety of the conceived systems make it possible for the designers to specify strategies of piloting, and modes of reconfiguration, before the first tests on a real prototype. Computer-controlled systems that combine mechanical, hydraulic, electronic and data-processing technologies are referred to as hybrid. For these systems, continuous dynamics is applied to the power characteristics, and discrete dynamics is related to the numerical control and the existence of discrete events (for example, failures and thresholds). The study of reliability for such systems must necessarily take into account the existing interactions between their physical parameters (e.g. temperature, pressure, speed, etc.) and the failure of their components. Traditional methods for reliability such as Failures Trees are insufficient for these complex and hybrid systems because of their dynamic nature. Therefore, the time and order of execution of the events must be taken into account to ensure the safety of these systems. To solve the problem of the scarcity of these scenarios to which methods based on simulation are exposed, techniques of acceleration of simulation were developed and largely used successfully, in particular, in nuclear engineering. The discrete events methods analyses (automats, Petri net) have their contribution in this field but the use of accessibility graph is quickly confronted to the problem of combinative explosion.

Our research work is placed within the framework of dynamic reliability. The objective is to carry out a qualitative analysis of the reliability of computer-based systems to extract the scenarios leading to feared states. This is a question of characterizing these scenarios as soon as possible in the design phase, which makes it possible to evaluate their probabilities of occurrence in order to validate the architecture of the system. We propose an approach based on linear logic and Predicates Transitions Differential Stochastic Petri Nets model (RdP PTDS), which respects the hybrid nature of these systems. This approach takes into account the continuous aspect of the system, and more particularly the thresholds associated to certain transitions in the Petri Net model. This approach determines more precisely the exact conditions of the occurrence of the feared event, i.e what has led the system to leave its normal operation and to evolve into the feared state. The originality of our approach is that the order of occurrence of the events is taken into account, and impossible scenarios with respect to continuous dynamics of the system are eliminated. Our approach is also directed towards the checking of certain properties of the computer-controlled systems. These properties can be temporal (e.g. maximum duration of a scenario, or duration between two orders), or related to the accessibility between two states. We believe that the automation of all stages of our approach is essential for the cases of complex systems where the risk of human error is very significant. This is why we developed a tool ESA_PetriNet (Extraction & Scenarios Analyser by PetriNet model), which makes it possible to extract the critical scenarios from a Petri Net model, and to check certain properties of these computer-based systems.

Key words: *Reliability, computer-based systems, dynamic reliability, properties checking, hybrid and dynamic systems, extended Petri Nets for safety, Linear Logic.*