Architecture pour la reconfiguration en temps réel des systèmes complexes

Ahmed Guadri

N° d'ordre : | 1 | 1 | 4 |

ÉCOLE CENTRALE DE LILLE

THÈSE

présentée en vue d'obtenir le grade de

DOCTEUR

en

Spécialité : Automatique et Informatique Industrielle

par

Ahmed Guadri

Doctorat délivré par l'École Centrale de Lille

Architecture pour la reconfiguration en temps réel des systèmes complexes

Soutenue le 15 décembre 2009 devant le jury d'examen :

Rapporteur	Eric Niel	Professeur à l'INSA de Lyon
Rapporteur	Hervé Gueguen	Professeur à SUPELEC de Rennes
Examinateur	François Pérès	Maître de conférences HDR à l'École Nationale d'Ingénieurs de Tarbes
Examinateur	Vincent Cocquempot	Professeur à l'Université des Sciences et Technologies de Lille (Lille 1)
Directeur de thèse	Etienne Craye	Professeur à l'École Centrale de Lille
Co-directeur de thèse	Nathalie Dangoumau	Maître de Conférences à l'École Centrale de Lille

Thèse préparée au sein du Laboratoire LAGIS

École Doctorale SPI 072

À mes parents

À mes frères et soeurs Hamdi, Tarek, Rim et Afef

À mes amis

Remerciements

D'abord, je tiens à remercier mes directeurs de thèse le Professeur Etienne Craye et Nathalie Dangoumau, Maître de Conférences pour leur patience, disponibilité et confiance qu'ils m'ont témoigné durant ces trois années de thèse.

Je veux aussi remercier le Professeur Hervé Gueguen et le Professeur Eric Niel pour avoir accepté d'être les rapporteurs de ma thèse.

Je tiens à remercier à le Professeur Vincent Cocquempot et François Pérès, Maître de Conférences pour avoir accepté d'être membres de jury en tant qu'examinateurs.

Enfin, je remercie tous mes amis de Lille qui ont permi de faire de mon séjour en France une expérience enrichissante et plaisante. Ils auront toujours une place dans mon coeur.

Table des matières

Table des figures

Introduction Générale

De plus en plus, assurer la sureté de fonctionnement en temps réel est devenu un enjeu incontournable lors de la conception des systèmes de commande et de supervision. Ceci est dû notamment à la complexité croissante des systèmes artificiels et à la criticité des missions qui sont associés à ces systèmes. Par exemple, un système de commande de vol doit gérer la commande d'un système constitué de plusieurs microcontrôleurs, réseaux locaux, calculateurs, capteurs, actionneurs... Il doit assurer la conduite de vol selon des normes drastiques de sécurité.

En général, la tolérance aux fautes peut être assurée en greffant une couche supplémentaire de supervision au système complexe commandé. Ceci peut être réalisé selon plusieurs approches de conception d'architectures. Dans l'industrie, les approches les plus prisées au niveau global s'appuient sur des implémentations avec des redondances surtout matérielle et une structuration en couches. Par ailleurs, l'occurrence de défaillances peut être traitée lorsque des procédures de recouvrement sont correctement prévues dans la phase hors ligne. Au niveau local, la tolérance aux fautes peut être assurée avec des méthodologies dédiées aux formalismes continus (linéaires ou nonlinéaires) ou discrets. Un aperçu de ces méthodologies est donné dans le chapitre 1.

L'utilisation de ces différentes méthodes n'est cependant pas aisée. En

effet, la construction et reconfiguration des superviseurs tolérants aux fautes nécessite l'utilisation d'un modèle abstrait par des experts à partir du modèle exhaustif de bas niveau disponible du système complexe. Cette tâche peut être très difficile et fastidieuse dans le cas de grands systèmes. En plus, le fait que de tels modèles soient générés et vérifiés par des humains a pour conséquence la possibilité de subsistance de fautes de modélisation ou de possibilités de comportements importants pour la commande et la sureté. Enfin, notons que les méthodes automatiques de génération de superviseurs souffrent d'une complexité très grande. Pour pallier toutes ces limites, nous avons essayé de proposer une méthodologie de construction automatique d'architectures de commande tolérantes aux fautes. Cette architecture doit idéalement assurer la tolérance aux fautes pour des systèmes complexes en présence de défaillances éventuellement partielles et imprévues. En contraste avec les méthodologies usuelles utilisant des modèles abstraits, notre approche doit se baser sur une modélisation de bas niveau exhaustive. Pour celà, nous avons envisagé une méthodologie de conception de l'architecture de commande axée sur deux phases principales. La phase hors ligne permet la construction d'un modèle abstrait et de superviseurs appropriés discrets à partir de la modélisation bas niveau selon des techniques d'abstraction. Par contre, la phase en ligne permet une mise à jour du modèle abstrait et la reconfiguration de l'architecture de commande selon l'état constaté du système. La formulation de la problématique et l'approche de résolution globale sont donnés dans le chapitre 1.

Lors du fonctionnement du système, l'occurrence d'une défaillance se traduit par l'invalidation de plusieurs comportements dans le modèle abstrait et l'introduction d'incertitudes. Par la suite, les modules de diagnostic et d'identification (qui ne rentrent pas dans l'objet de notre thèse) fournissent

des informations à propos de l'état structurel du système. Ces informations sont exploités afin de réduire de façon progressive le modèle de bas niveau hybride au cours du temps. Le chapitre 2 discute de la modélisation bas niveau exhaustive et de l'incorporation des défaillances dans la modélisation.

Afin de pouvoir mettre à jour le modèle discret, nous utilisons des algorithmes permettant de vérifier ou trouver des propriétés qualitatives du système analysé à partir de la description exhaustive de bas niveau. D'un côté, nous proposons des algorithmes pour le calcul d'atteignabilité à partir de régions de l'espace de phase en exploitant le caractère Lipchitzien des fonctions définissant l'évolution du système. Des techniques simples et itératives sont démontrées et illustrées dans le chapitre 3. En plus, d'autres algorithmes de vérification et de génération de commande stabilisatrice dans une région sont développés dans le chapitre 4. Ces différents techniques constituent la boite à outils nécessaire pour la génération et vérification de propriétés qualitatives.

Pour pouvoir piloter un tel système selon des objectifs de fonctionnement discrets, l'utilisation de méthodologies d'abstraction est nécessaire afin de transformer le modèle bas niveau exhaustif en un modèle discret approprié pour la commande ou la reconfiguration. Pour cela, le modèle hybride de bas niveau (qui a un nombre d'états infini) et le système abstrait discret (avec un nombre d'états fini) peuvent être interprétés en tant que systèmes à transitions. Des méthodes existent afin de réduire les systèmes à transitions à états éventuellement infinis en des systèmes avec un nombre d'états finis (génération d'une relation de bisimulation) en s'appuyant sur les techniques des chapitres 3 et 4 précédemment proposés. Ces méthodes ont cependant l'inconvénient de ne pas être très pratiques car elles présentent des conditions d'arrêt drastiques (Ainsi, leur implémentation risque de se traduire par une boucle infinie). Nous adressons cet inconvénient en proposant des modi-

fications sur l'algorithme d'abstraction permettant de relaxer les conditions d'arrêt grâce à l'introduction de la notion de contexte. le chapitre 5 adresse cette problématique.

Pour le module implémentant la reconfiguration, seul le modèle discret abstrait (généré et mis à jour par les algorithmes d'abstraction) est visible. Notre approche de reconfiguration est réalisée en se basant sur la synthèse hors ligne de superviseurs pour plusieurs schémas prédéfinis et des objectifs dégradés. Lorsqu'une défaillance est détectée, la procédure de reconfiguration est déclenchée en essayant, au fur et à mesure de l'enrichissement du modèle abstrait, de réduire le fonctionnement du système défaillant dans un des schémas prédéfinis. Le succès de cette étape garantit le succès de la reconfiguration.

Nous avons illustré chaque étape de cette approche sur un exemple simple de deux réservoirs communicants, ou l'objectif de fonctionnement est de réguler le niveau de fluide même en présence de défaillances partielles au niveau des vannes (blocage) ou réservoirs (fuite).

Chapitre 1

Etat de l'Art

1.1 Introduction

Ce chapitre commence par une formulation, formalisation et exploration de la problématique de la reconfiguration des systèmes complexes. Les cas particuliers des modélisations discrètes, continues et hybrides seront présentés. Le chapitre est enfin clôturé par l'exposé de l'architecture globale proposée pour la résolution de la problématique de reconfiguration.

1.2 Problématique générale

Dans cette partie, nous donnons une brève introduction à la discipline de sûreté de fonctionnement et à la problématique de tolérance aux fautes. Une formulation de la problématique traitée sera également abordée.

Un système dynamique peut être défini comme étant un système exhibant un comportement déterministe et causal. Généralement, l'évolution d'un tel système dépend de son *état* et de ses *entrées* de commande ou d'environnement (les entrées d'environnement peuvent être des perturbations,

un bruit...). Plusieurs phénomènes naturels ou artificiels (tout ce qui est construit par l'homme) peuvent être considérés au niveau macroscopique comme causaux. En particulier, les systèmes dynamiques artificiels présentent un intérêt majeur puisqu'ils sous entendent souvent la notion de *finalité* : un système artificiel est presque toujours conçu pour satisfaire un objectif défini. Par exemple, les usines font des produits finis, les ordinateurs traitent stockent et diffusent l'information, les centrales d'énergie délivrent de l'électricité...

Souvent, la commande des systèmes dynamiques est réalisée grâce à une boucle de rétroaction : Un module de commande reçoit des *observations* de la part du système, et injecte en fonction de ces informations et de son état interne une entrée de commande. Ce principe général peut être adapté selon la nature du système (système informatique avec un très grand nombre de composants contenant des comportements discrets, mécanique, fluides...), le type de modélisation choisie ou adaptée (discrète, continue, UML...) ou l'objectif poursuivi (poursuite de trajectoire, sûreté de fonctionnement...). En pratique, deux modules additionnels doivent être ajoutés afin de rendre compte plus précisément du fonctionnement réel : un module de détection et observation qu'on appelle *générateur* (incarné en pratique par des capteurs) et d'injection de commande (par des actionneurs). La figure 1.1 illustre le principe général de rétroaction comme étant un échange entre différents blocs durant le temps.

Lorsque le degré de sophistication du système dynamique le rend difficile ou impossible à analyser ou contrôler par un agent humain il est qualifié de *complexe*. Ainsi, une grande partie des phénomènes naturels reconnus comme déterministes sont complexes à l'instar des changements météorologiques, des phénomènes de société (si l'on admet que les vivants sont sujets au détermi-

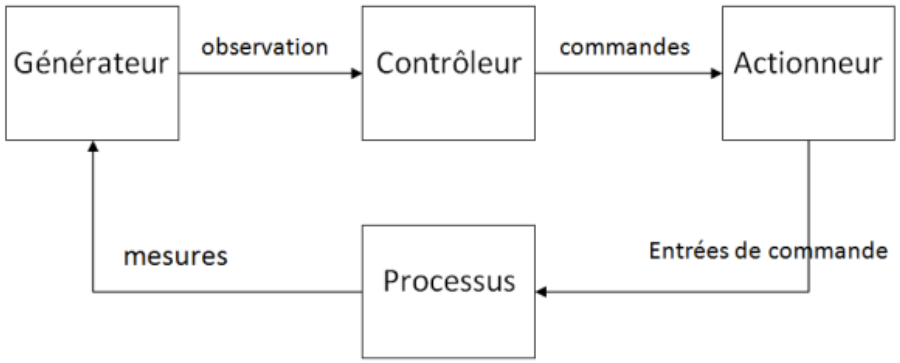

FIG. 1.1 – *Principe général de la commande*

nisme, ce qui est un postulat de base pour les sciences humaines). Il est alors courant d'avoir recours à plusieurs boucles fermées au lieu d'une seule (voir la Figure 1.2.

Notons que le principe général de la commande dans les figures 1.1 et 1.2 est indépendant de la modélisation, nature et objectifs d'utilisation du système traité. Il n'exclut pas l'introduction de perturbations ou d'incertitudes. Par la suite, l'introduction de nouveaux modules de supervision et/ou de reconfiguration serait alors faite de façon naturelle en tant que nouvelle couche $n + 1$.

Un système complexe artificiel est toujours conçu pour assurer des *objectifs*. Ceux-ci peuvent être classés selon différents degrés de priorité, de criticité ou selon le niveau d'abstraction. Un exemple assez caractéristique est celui des réseaux de télécommunication. En effet, ceux-ci sont sujets assez fréquemment à des pannes de matériel (par exemple, des routeurs hors service, des segments du réseau en congestion...); cependant, une qualité de service minimale reste garantie (accès au réseau, fonctionnement de services tels que messagerie, web...). Une performance minimale peut cependant être assurée grâce à une hiérarchisation du système et l'utilisation de protocoles

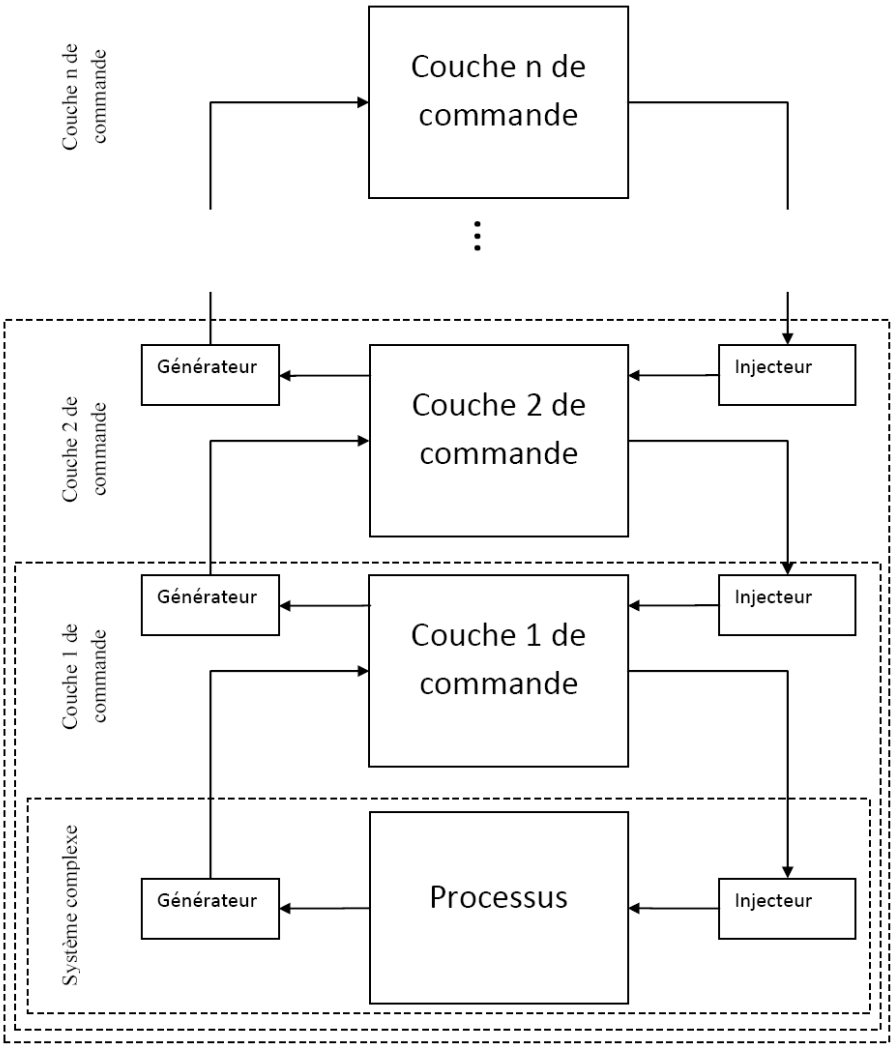

FIG. 1.2 – *Principe général de la commande hiérarchique*

de haut niveau qui s'adaptent aux pannes ou perturbations de bas niveau.

Un tel système peut être qualifié de *système tolérant aux fautes*. Cependant, notons qu'un tel système peut cependant *échouer* (c'est à dire ne pas assurer les objectifs de bases assignés) si une défaillance d'une partie du système n'a pas été précédemment correctement analysée et anticipée. Dans le cas des réseaux de télécommunication, une panne de courant généralisée au niveau d'une ville à la suite d'intempéries peut provoquer l'échec de fonctionnement du système lorsque des sources d'énergie de secours ne sont pas anticipées quels que soient les protocoles utilisés pour la commande de haut niveau.

Plusieurs autres systèmes complexes artificiels sont construits selon une hiérarchisation des objectifs de fonctionnement. Cette hiérarchisation permet de limiter l'effet d'un échec de fonctionnement à un niveau local. Elle permet aussi de donner au système contrôlé de façon automatique un moyen concret pour l'estimation de la gravité de l'état global du système et de la nécessité éventuelle de mesures de maintenance ou de mise à jour. En considérant par exemple un réseau local d'entreprise en tant que système complexe, une panne au niveau d'un poste d'informatique ne se traduit pas par une mesure aussi radicale que l'enregistrement de toutes les données de l'entreprise et la mise hors tension de tous les appareils informatiques. Par contre, une telle mesure peut être nécessaire lorsque l'alimentation globale du réseau est défaillante. La modélisation hiérarchique des systèmes dynamiques a été traitée de façon formelle dans plusieurs travaux tels que [18] et [71]. Elle est très présente dans le cas des systèmes de commande de vol, des architectures informatiques, des réseaux de capteurs...

En traitant un niveau précis d'abstraction, il est commode de diviser les objectifs de fonctionnement en des objectifs *standards* (qui quantifient les

performances voulues du système) et *dégradés* (qui concernent des objectifs de sécurité ou de fonctionnement impératives à assurer) (voir [26], [9], [11] et [12] pour plus de détails).

L'apparition de la discipline de la sûreté de fonctionnement a été une conséquence de la complexité grandissante des systèmes artificiels (par exemple, une voiture contient désormais des dizaines de microcontrôleurs afin de faciliter le fonctionnement global), et du coût et criticité croissants de quelques systèmes développés pour des applications dédiées (tels que les centrales chimiques et nucléaires, les systèmes autonomes, les réseaux de communication...). Ainsi, la conception des systèmes artificiels est de plus en plus soumise à des normes et exigences élevées par rapport à la fiabilité dynamique.

Dans ce cadre, la tolérance aux fautes est la capacité d'un système de continuer le fonctionnement, éventuellement de façon dégradée, quand une partie devient défaillante. Les approches pour l'achèvement de cette propriété peuvent être classifiées en deux grandes catégories : les approches passives et les approches actives. Une approche active permet de réaliser la tolérance aux fautes en veillant à ce que les lois de commande soient *robustes* par rapport aux fautes possibles du système. Par contre, une approche active se base sur une adaptation de la stratégie de commande ou l'architecture matérielle système en cours de fonctionnement selon les défaillances constatées. En pratique, une approche active est souvent implémentée grâce à une couche supplémentaire de supervision.

Definition 1 *[9] La supervision consiste à surveiller si les objectifs de fonctionnement sont respectés par le système commandé et à modifier la stratégie de commande selon des objectifs de fonctionnement éventuellement dégradés.*

Ainsi, la supervision contient deux volets essentiels : la surveillance (né-

cessitant le diagnostic et l'identification du fonctionnement) et la commande (nécessitant l'injection d'ordres et commandes en continu selon l'état dynamique observé du système et l'état structurel résultant de l'identification et du diagnostic réalisés depuis le début de fonctionnement). Notons que cette couche de supervision n'exclut pas l'intégration d'une décision humaine pour la surveillance ou le changement de mode de fonctionnement.

L'objectif de cette thèse est de proposer une méthodologie pour la construction de superviseurs garantissant de façon active la tolérance aux fautes des systèmes complexes.

1.3 Terminologie et formulation

Plusieurs définitions peuvent être avancées pour la reconfiguration selon la discipline (systèmes informatiques, théorie de la commande...). La définition 2 a été proposée dans [11] pour la reconfiguration dans le contexte de la théorie de la commande et de la sûreté de fonctionnement.

Definition 2 *La reconfiguration est toute approche active permettant d'accomplir un fonctionnement tolérant aux fautes des systèmes dynamiques.*

Cette définition de la reconfiguration est spécifique au domaine de la tolérance aux fautes, puisqu'elle n'inclut pas la reconfiguration nécessitée par un changement d'exigences et objectifs du fonctionnement (Exemple : la reconfiguration des systèmes manufacturiers reconfigurables lors d'un changement des exigences du marché, voir paragraphe 1.4.1.1).

En général, une approche de reconfiguration peut être implémentée en tant que nouvelle couche de commande au dessus des autres couches basiques de commande; on l'appellera *superviseur* (voir la Figure 1.3).

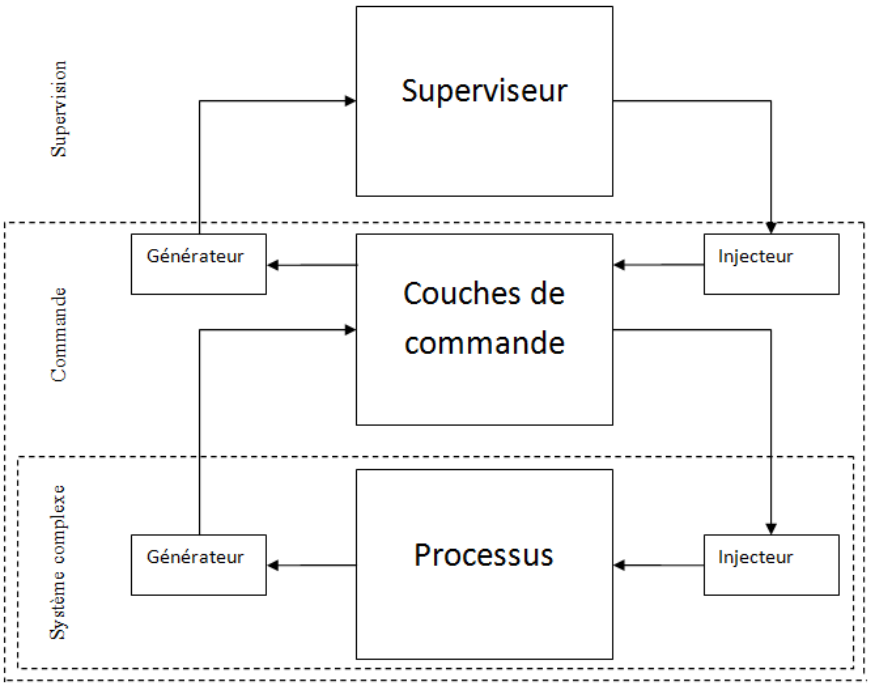

FIG. 1.3 – *Architecture générale de la supervision*

En général, un système dynamique artificiel est conçu par l'homme pour remplir un rôle bien défini. Ces objectifs peuvent être la poursuite d'une trajectoire, une stabilisation de comportement d'un module externe, une production continue... Lorsqu'une partie du système est défaillante (coupure de courant dans un réseau électrique, segments défaillants dans un réseau informatique...) il est possible que les objectifs de fonctionnement soient relaxés afin de garantir un fonctionnement en mode *dégradé*. Dans ce cas, le système devrait fonctionner selon les ressources encore disponibles afin de satisfaire les objectifs jugés indispensables (ainsi, un exemple typique d'objectif de fonctionnement dégradé est de garantir la sécurité des personnes dans le cas d'une panne sévère). Une analyse plus approfondie de la spécification des objectifs sera donnée par la suite.

Nous commençons d'abord par détailler la terminologie utilisée.

Afin de réduire les risques inhérents au fonctionnement des systèmes complexes, deux voies peuvent être pratiquées : diminution de la probabilité d'occurrence de l'événement indésirable (fiabilité et disponibilité, voir la définition 3), ou l'atténuation des conséquences d'un tel événement.

Definition 3 *D'après la norme NF X 60-500 (voir [73]).*

1. *La fiabilité est l'aptitude d'une entité à accomplir une fonction requise, dans des conditions données, pendant un intervalle de temps donné.*
2. *La disponibilité est l'aptitude d'une entité à être en état d'accomplir une fonction requise dans des conditions données, à un instant donné ou pendant un intervalle de temps donné, en supposant que la fourniture des moyens extérieurs nécessaires soit assurée.*

La disponibilité à un instant t est une grandeur instantanée, puisqu'elle ne fournit pas d'informations par rapport au fonctionnement du système durant l'intervalle $[0,t[$.

D'autres notions importantes de la sûrété de fonctionnement sont exposées dans [73] tel que la maintenabilité, la sécurité... Nous nous intéressons par la suite au concept fondamental de défaillance dans la définition 4.

1.3.1 Défaillances et Diagnostic

1.3.1.1 La notion de défaillance

Definition 4 *(voir [73]) Une défaillance est l'altération ou la cessation de l'aptitude d'un dispositif à accomplir ses fonctions requises avec les performances définies dans les spécifications techniques.*

La classification des défaillances selon leurs natures, causes et conséquences a été déjà explorée dans divers domaines de l'industrie, notamment

dans le cas des systèmes *critiques* (dont les conséquences de dysfonctionnements ne sont pas négligeables [58]) telles que les architectures de commande de vol dans le domaine de l'aéronautique, la supervision des centrales chimiques... On peut classifier les défaillances par rapport à leurs vitesses d'apparition (soudaines ou progressives), leur nature (complète ou partielle)... Quelques notions sont détaillées dans la définition 5.

Definition 5 *(voir [73])*

- *Défaillances partielles : ce sont des défaillances qui résultent de déviation d'une ou des caractéristiques au delà des durées spécifiées, mais telle qu'elle n'entraîne pas une disparition complète de la fonction requise.*
- *Défaillances complètes : ce sont des défaillances résultant de déviation d'une ou des caractéristiques au-delà des limites spécifiées, telle qu'elle entraîne une disparition complète de la fonction requise.*
- *Défaillances soudaines : ce sont des défaillances qui n'auraient pas pu être prévues par un examen ou une surveillance antérieure.*
- *Défaillances progressives : ce sont des défaillances qui auraient pu être prévues par un examen ou surveillance antérieure.*

Une défaillance peut en général résulter d'une modification du fonctionnement dynamique interne d'une partie du système. Dans ce cas, cette modification peut être immuable et instantanée (défaillance soudaine). Elle peut aussi induire un changement en durée du fonctionnement du système.

Afin de pouvoir analyser et superviser le système en défaillance, l'état courant du système doit être correctement identifié. C'est la mission des modules assignés au diagnostic.

1.3.1.2 La notion de diagnostic

Definition 6 *Le diagnostic est l'identification de la cause probable de la défaillance du système à l'aide d'un raisonnement logique fondé sur un ensemble d'informations provenant des symptômes, effets, inspections, contrôles ou tests de maintenance.*

Cette tâche de diagnostic peut être découpée en deux étapes majeures [26]. La *détection* permettant d'engendrer des symptômes à partir d'un ensemble d'indicateurs. La *classification* permet de remonter d'un ensemble de symptômes à un ensemble minimal de composants dont le dysfonctionnement est suffisant pour expliquer l'apparition des symptômes observés.

En général, l'occurrence d'une défaillance à un instant t introduit une *incertitude* à propos de la dynamique de comportement du système. Cependant, cette incertitude peut être réduite au fur et à mesure du fonctionnement du système grâce à des dispositifs additionnels dédiés à l'identification. Afin de pouvoir mettre à jour le modèle du système défaillant, les modules dédiés au diagnostic et à l'identification doivent fournir à la fin de ce processus un modèle, éventuellement incertain, qui est réduit au fur et à mesure du temps. Diverses méthodologies de diagnostic sont fournies dans [11] et [73]. Cette perception du diagnostic en tant que processus permettant de réduire l'incertitude et de déduire des nouveaux comportements du système en temps réel sera admise dans la suite de cet exposé.

1.3.2 Formulation de la problématique de reconfiguration

Dans ce qui suit, nous donnons la formulation de la problématique de reconfiguration dans le cas général. Ensuite, nous donnons une formulation

selon les paradigmes de modélisation différents. Nous commençons par la formulation donnée dans [9] et [12].

Un problème de commande peut être défini par un ensemble de comportements objectifs B c'est à dire des exécutions qui sont acceptables lors du système, de commandes U et de contraintes C. Les contraintes sont des relations régissant l'évolution dynamique et l'intéraction des composants du système (elles peuvent être des équations algébrodifférentielles, des automates à états finis, des bond graphs...). En général, on définit les contraintes en tant que *structure* S sous des paramètres θ. La résolution du problème de commande est possible en trouvant dans U une loi de commande V qui satisfait B en prenant en compte les contraintes décrites par C. Parfois plusieurs solutions existent. Dans ce cas, on peut choisir une loi grâce à un critère J, voir [9].

L'occurrence d'une défaillance cause un changement de la structure du système. Grâce aux modules de diagnostic et d'identification, il est possible d'avoir à chaque instant une estimation $\hat{S}$ de la structure et $\hat{\theta}$ des paramètres. Dans ce cas, il est nécessaire de veiller à un respect des objectifs dégradés de fonctionnement.

Dans ce cas, nous proposons une définition de la problématique de génération de commande tolérante au fautes comme suit :

Problématique 1 *Le problème de commande tolérante aux fautes est donné par* $< B', \hat{S}, \hat{\theta}, U >$ *avec* $(\hat{S}, \hat{\Theta})$ *est l'ensemble de structures et paramètres possibles du système défaillant retournés grâce au diagnostic.* B' *est l'ensemble d'objectifs de fonctionnement dégradés. Enfin,* U *est l'ensemble de signaux de commande (pouvant être des fonctions en boucle fermée).*

Cette définition est différente de celle donnée dans [9] puisqu'elle englobe le cas de l'adoption d'objectifs dégradés.

Dans le cas des systèmes dynamiques, l'état du système à chaque instant peut être quantifié grâce à un certain nombre de variables dynamiques. Ainsi, les objectifs de fonctionnement sont en général spécifiés par rapport à l'évolution de ces variables dynamiques. Notons par $M(V)$ l'ensemble d'évolutions possibles des variables dynamiques dans le cas d'une commande dans $V \subset U$, et $M = M(U)$. Ainsi, la problématique de commande tolérante aux fautes se réduit à trouver V pour s'assurer que $M(V) \subset B'$.

La Figure 1.4 résume la problématique de reconfiguration dans le cadre de cette thèse de doctorat.

1.3.3 La problématique de reconfiguration dans le cas discret

1.3.3.1 Introduction à la modélisation discrète

Un système à événement discret est un système dynamique qui évolue selon des événements générés à des intervalles irrégulières [60]. Un des avantages d'une telle modélisation est que des systèmes larges avec un nombre conséquent de composants peuvent être réduits à des modèles abstraits plus simples selon les propriétés qu'on veut analyser ou assurer. Deux grands paradigmes de modélisation peuvent être avancés : les automates à états finis [35] et les réseaux de Petri [57]. C'est la modélisation avec des automates à états finis qui nous intéresse par la suite. Ce choix est motivé par l'aisance avec laquelle il est possible de réaliser des calculs lorsque le nombre d'états et de transitions sont finies.

La problématique exposée dans le cas discret est inspirée de [60] et [35].

Deux grandes classes peuvent être distinguées dans le cas des systèmes à événements discrets. Les systèmes à événements discrets non temporels et

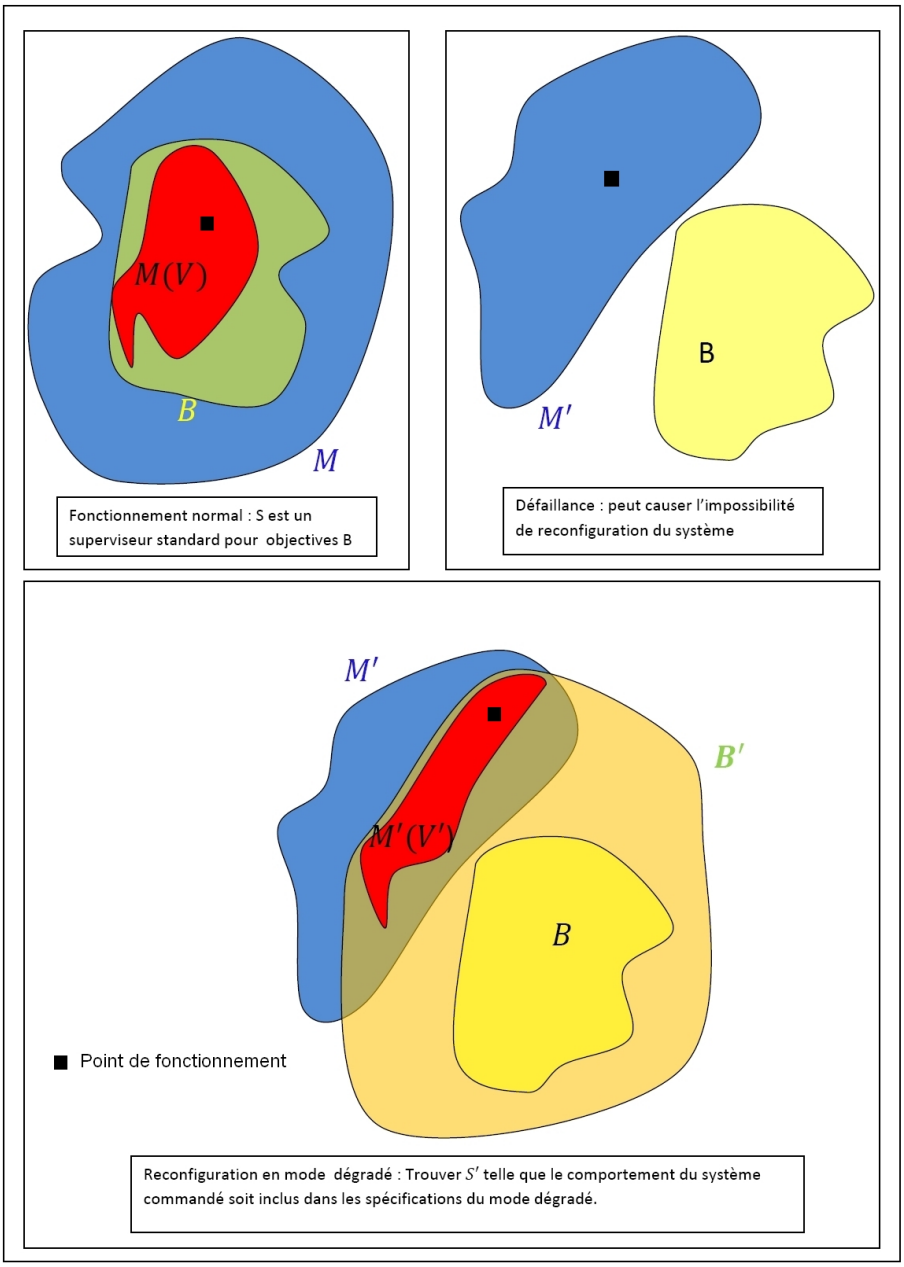

FIG. 1.4 – *Problématique générale de la reconfiguration*

les systèmes à événements discrets temporels.

Dans le cas général d'une modélisation discrète, nous nous intéressons à la séquence d'événements générés par le système. Soit Σ l'ensemble fini d'événements possibles, $\epsilon \in \Sigma$ l'événement silencieux et Σ^* l'ensemble de tous les *mots* avec une longueur finie de Σ. Nous définissons l'opérateur $pre(s)$ comme étant l'ensemble de préfixes du mot s. Soit :

$$pre(s) = \bigcup\{u \in \Sigma^* / \exists v \in \Sigma^*, uv = s\}$$

En général, un sous ensemble $L \subset \Sigma^*$ est appelé un *langage*. L'ensemble de comportements possibles d'un système dynamique peut être décrit par un langage M. Si un mot $s = \sigma_1\sigma_2...\sigma_n$ est dans M, ceci veut dire que l'événement σ_i est déclenché à un instant $\tau_i, \tau_i \leq \tau_{i+1}$ par le système (lorsqu'il s'agit d'un événement non commandable issu du fonctionnement interne du système) ou par l'architecture de commande (il s'agit alors d'un événement commandable servant à modifier le fonctionnement interne du système). Si on modélise le fonctionnement interne d'un système par un langage M, il est naturel d'exiger que ce langage accepte les préfixes des mots d'exécutions possible de M.

On définit l'ensemble préfixe d'un langage comme étant :

$$pre(L) = \bigcup\{pre(s), s \in L\} \tag{1.1}$$

Lorsque $pre(L) = L$, on dit que L est *fermé*. Les langages utilisés pour la modélisation discrète des systèmes dynamiques sont fermés.

Un formalisme important de modélisation des systèmes à événements discrets est celui des automates à états finis.

Definition 7 *Un automate à états finis est un uplet* $A = (Q, \Sigma, \delta, q_0, F)$ *avec :*

– Q *: Un ensemble fini d'*états.

- Σ *: Un ensemble fini de symboles (ou événements).*
- δ *: Une fonction de transition sur* $Q \times \Sigma - \{\epsilon\} \rightarrow Q$.
- $q_0 \in Q$ *: L'état initial du système.*
- F *: Ensemble d'états finaux ou d'acceptation.*

La fonction de transition δ est étendue sur $Q \times \Sigma^* \rightarrow Q$ comme étant :

$$\forall q \in Q, \forall w \in \Sigma^*, \sigma \in \Sigma, \delta(q,\epsilon) = q, \delta(q,w\sigma) = \delta(\delta(q,w),\sigma) \tag{1.2}$$

On note $\delta(q,\sigma)!$ lorsque $\exists q'/\delta(q,\sigma) = q'$. De façon informelle, $\delta(q,\sigma)!$ veut dire qu'un successeur de l'état q existe lorsque l'événement σ est déclenché.

Un automate à états finis est dit *déterministe* lorsque pour tout état source q et un événement σ, $\delta(q,\sigma)!$ implique que δ associe un seul état destination. Par contre, il est *complet* lorsque pour tout $q \in Q, \sigma \in \Sigma$, on a toujours $\delta(q,\sigma)!$.

Dorénavant, le mot automate désigne implicitement un automate à états finis déterministe qui n'est pas nécessairement complet.

Definition 8 *Un mot* $s = \sigma_1\sigma_2...\sigma_n \in \Sigma^*$ *est accepté par* $A = (Q,\Sigma,\delta,q_0,F)$ *si* $\delta(q_0,\sigma_1...\sigma_n)!$.

On définit le langage *accepté* par un automate à états finis A comme étant l'ensemble de mots acceptés par l'automate. Ce langage sera noté $L(A)$.

Une classe importante des langages est celle des langages *réguliers* : c'est la classe de langages qui peuvent être acceptés par des automates à états finis. Notons que la classe de langages acceptés par des réseaux de Petri ou par des machines de Turing est strictement plus grande et englobe la classe des langages réguliers.

Lorsqu'on décrit le fonctionnement d'un système dynamique à l'aide d'un modèle discret, on peut distinguer les événements régissant l'évolution du

système en des événement commandables Σ_c ou non commandables Σ_{nc}, observables Σ_o ou non observables Σ_{no} avec :

$$\Sigma = \Sigma_c \cup \Sigma_{nc} = \Sigma_o \cup \Sigma_{no} \tag{1.3}$$

1.3.3.2 Supervision des systèmes à événements discrets

Les systèmes à événements discrets sont souvent une représentation réduite de haut niveau d'un fonctionnement plus complexe. Ceci présente l'avantage très important de permettre l'analyse, la commande et la supervision du système de façon automatisée et simplifiée. Par exemple, le fonctionnement d'un réseau informatique peut être vu en tant qu'assemblage de câbles, routeurs, terminaux. Cependant, les protocoles (TCP, ICMP...) utilisant cette infrastructure envisagent cet assemblage en tant que système discret éventuellement temporel. Une telle utilisation n'est possible que grâce à des *primitives* de service permettant le passage du niveau abstrait (couches supérieurs du modèle physique) aux couches de bas niveau tel que structurel (couche physique). Nous reviendrons sur ces notions dans le chapitre dédié à l'abstraction.

Interprétons la Figure 1.4 dans le cas d'une modélisation discrète non temporelle. Soient M,B,B' : les langages réguliers sur l'alphabet Σ décrivant respectivement le comportement du système, les objectifs standards et dégradés. L'objectif de supervision est de restreindre le fonctionnement du système dans B. Par contre, l'objectif de reconfiguration est de garder le fonctionnement du système B' lorsque le fonctionnement du système est modifié à cause d'une défaillance.

Afin de trouver des solutions aux problèmes de supervision ou de reconfiguration, le superviseur doit décider selon l'état du système quels sont

les événements commandables qui doivent être désactivés afin d'empêcher la possibilité des comportements interdits par B. L'introduction des automates à états finis simplifie le processus de construction du superviseur puisque le choix des événements à désactiver dépend de l'état des automates correspondants à B et M à un instant donné.

Plusieurs formulations ont été proposées afin de réaliser la supervision des systèmes à événements discrets, voir [60]. En général, on peut modéliser un superviseur comme étant une fonction $\gamma : \Sigma^* \rightarrow 2^{\Sigma_c}$ qui associe à chaque mot généré $s \in \Sigma^*$ par le système un ensemble d'événements commandables $f(s)$ autorisés.

Une façon pratique de réaliser un superviseur S pour le système discret commandé est de construire un automate à états finis $A(S) = (Q_S,\Sigma,\delta_S,s_0,Q_S)$ décrivant le niveau de commande. Dans ce cas, la supervision est réalisée de façon implicite. En effet, pour $s \in \Sigma^*$, $\sigma \in \gamma(s)$ si et seulement si $s \in \Sigma_c$ et $\delta(\delta(s_0,s),\sigma)!$. De façon informelle : lorsque la séquence s est déclenchée par le système commandé, σ est un événement de commande non désactivé par l'automate de supervision $A(S)$ lorsque l'exécution $s\sigma$ existe dans $A(S)$. C'est cette implémentation qui sera adoptée par la suite.

Une question se pose alors : y'a-t-il une garantie pour l'existence d'un superviseur? Pour cela, nous définissons la notion fondamentale de commandabilité dans la définition 9. Soit la fonction *pre* qui associe à chaque mot ou langage l'ensemble de préfixes correspondants.

Definition 9 *Un langage $K \subseteq \Sigma^*$ est commandable par rapport à un langage L si :*

$$pre(K)\Sigma_u \cap L \subseteq pre(K)$$

La condition de commandabilité garantit qu'il n'y a pas de comportements possibles du système qui pourraient forcer le fonctionnement du système en dehors de K.

On définit aussi une condition de commandabilité équivalente et étendue d'après [14] :

Definition 10 *Un langage $K \subseteq \Sigma^*$ est commandable par rapport à un langage L si :*

$$pre(K)\Sigma_u^* \cap L \subseteq pre(K)$$

Ainsi, la problématique de supervision dans le cas discret peut être interprétée comme étant trouver un langage commandable dans $B \cap M$. Il a été démontré que l'ensemble de langages qui vérifient ces conditions $supC(B)$ est non vide (car $\emptyset \in supC(B)$ est commandable) et fermé par union (c'est à dire : si L,$L' \in C(B)$,$L \cup L' \in C(B)$). Ainsi, il existe toujours un langage commandable *maximal* qui est la réunion de tous les langages commandables dans B.

Le langage $supC(B)$ présente un intérêt essentiel pour la commande du système discret de langage M. En effet, $supC(B)$ est un langage commandable permettant de respecter les spécifications B tout en étant le langage le moins restrictif. Ainsi, plusieurs algorithmes ont été avancés pour la construction de ce langage ou de l'automate qui lui correspond. Un premier algorithme récursif a été proposé dans [67]. Cependant, nous nous intéressons à l'algorithme démontré dans [14] se basant sur la formule donnée par Eq. (1.4). Une condition initiale admise par les auteurs cités est que B est inclus dans M. Étant donnée que nous ne retenons pas cette hypothèse, et qu'il est donc possible que des spécifications d'objectifs ne soient pas possibles pour un système décrit par M, nous remplacerons à chaque fois B par $M \cap B$.

L'opérateur D_{uc} est une fonction qui associe à chaque mot $s \in \Sigma^*$ le plus grand préfixe de s dont le dernier symbole est commandable. Remarquons que $M - (B \cap M) = M - B$. Ainsi, l'équation définie dans [14] $supC(B) = B \cap M - D_{uc}(M - (B \cap M))\Sigma^*$ devient :

$$supC(B) = M \cap B - D_{uc}(M - B)\Sigma^* \tag{1.4}$$

Eq. (1.4) stipule que le plus grand langage commandable n'est autre que le langage de spécification dans le cas du système traité $(M \cap B)$ duquel on enlève tous les mots qui peuvent conduire à un comportement inconsistant avec B $(B-M)$. Un algorithme de conception de $supC(B)$ est donné à partir de Eq. (1.4) dans l'algorithme 1 en s'appuyant sur les opérations usuelles sur les automates à états finis : l'union, l'intersection, la complémentation (voir [35] pour plus de détails).

Algorithme 1 *Données :*

- *Langage régulier des spécifications standards de fonctionnement : B et l'automate à états finis déterministe $A(B)$ correspondant.*
- *Langage régulier du comportement normal du système : M et l'automate à états finis déterministe $A(M)$ correspondant.*

Résultat : Langage commandable maximal dans B par rapport à M : $supC(B)$ et l'automate A correspondant. Début

1. *Réaliser les automates $A(B \cap M)$ et $A(M - B) = A(M \cap \bar{B})$.*
2. *Réaliser $A(D_{uc}(M - B)\Sigma^*)$*
3. *Réaliser $A(M \cap B - D_{uc}(M - B)\Sigma^*)$*

Fin

En pratique, la réalisation de $A(B \cap M)$ et $A(M - B)$ est presque similaire. En effet, une première étape serait de *compléter* l'automate $A(B)$ en

ajoutant un état *puit* non final auquel on associe les transitions non définies initialement dans $A(B)$ (plus formellement, on modifie la structure de $A(B)$ en associant pour chaque q,σ où $\delta(q,\sigma)$ est non définie une nouvelle transition vers q_{puit}, où q_{puit} n'est pas final). L'automate $A(B \cap M)$ est réalisé comme étant le *produit* des deux automates duquel on enlève toutes les transitions qui ne paratagent pas le même événement. Un état produit est alors final lorsque les états correspondants le sont dans $A(B)$ et $A(M)$. $A(M-B)$ peut aussi être réalisé de la même façon en complémentant l'automate complet associé à B (ceci peut être fait en inversant les états finaux et non finaux) et en construisant $A(M \cap \bar{B})$ de la même façon.

$A(D_{uc}(M-B)\Sigma^*)$ peut être réalisé en interdisant les séquences de transitions qui mènent vers un état final, et en ajoutant un état puit auquel sera associé toutes les transitions sortantes des états finaux de $A(D_{uc}(M-B))$. Enfin, l'automate $A(M \cap B - D_{uc}(M-B)\Sigma^*)$ peut être construit en tant que l'automate correspondant à l'intersection $A(M \cap B \cap \overline{D_{uc}(M-B)\Sigma^*})$.

En pratique, il est parfois impossible de détecter instantanément l'occurrence de certains événements discrets. Aussi, on introduit une nouvelle dissociation des événements en des événements observables Σ_o et non observables Σ_{no}. On a alors : $\Sigma = \Sigma_o \cup \Sigma_{no} = \Sigma_c \cup \Sigma_{nc}$. Ainsi, la tâche de construction d'un superviseur doit être faite en prenant en compte l'existence d'événements non observables. Une notion d'observabilité a été définie dans le cas des systèmes à événements discrets. Cependant, cette notion s'est avérée assez difficile à vérifier et exploiter puisque l'existence d'un sous-langage commandable et observable maximal n'est pas toujours garantie. Ainsi, on a défini dans la littérature une autre condition plus restrictive, mais plus pratique, qui est la normalité (voir [48], [47] et [60] pour plus de détails).

Afin de décrire un système à événements discrets avec des observations

partielles, nous définissons la fonction projection $P : \Sigma \rightarrow (\Sigma_o \cup \{\epsilon\})$ selon :

$$P(\sigma) = \begin{cases} \sigma, \text{ if } \sigma \in \Sigma_o \\ \epsilon, \text{ if } \sigma \notin \Sigma_o \end{cases} \tag{1.5}$$

On étend la définition de P sur Σ^* par:

$$P(\epsilon) = \epsilon, P(s\sigma) = P(s)P(\sigma), \forall s \in \Sigma^*, \sigma \in \Sigma$$

À partir de cette définition, on peut définir une relation d'équivalence :

$$s \equiv s' \Leftrightarrow P(s) \equiv P(s')$$

Lorsque Σ_o est l'ensemble d'événements observables, le comportement M du système est vu en tant que $P(M) \in \Sigma_o^*$. Dans ce cas, le superviseur est un automate à états finis sur l'alphabet Σ_o.

La condition de normalité est donnée dans la Définition 11.

Definition 11 *Par rapport à un langage de comportement M, un langage fermé K est dit* P-normal *lorsque :*

$$K = M \cap P^{-1}(P(K))$$

Un langage est *P-normal* lorsqu'on est capable de prévoir si le mot d'exécution concret est dans K à partir de sa projection $P(K)$. Étant donné l'intérêt que présente le concept de normalité, on définit l'ensemble de sous langages normaux $N(B)$. Il a été démontré dans [48] que le plus grand langage normal dans B qu'on appelle $supN(B)$ existe toujours. De plus, la condition de normalité implique celle de l'observabilité.

Une formule de calcul de $supN(B)$ a été donnée dans [14] :

$$supN(B) = B - P^{-1}P(M - B)\Sigma^* \tag{1.6}$$

À la différence de $supC(B)$, la construction de l'automate $A(supN(B))$ est exponentielle par rapport au produit de nombres d'états de B et M [14].

Ainsi, réduire le comportement du système dans $supN(B)$ permet de restreindre le fonctionnement du système dans un sous langage de B en commandant à partir des observations partielles données par la projection P. Le langage commandable maximal pour B dans le cas d'une observation partielle par P a été démontré dans [14] :

$$supCN(B) = M \cap P^{-1} supC_P(PsupN(B)) \tag{1.7}$$

avec pour $D \subseteq PM$, $C_P(D)$ est défini par $C_P(D) = \{L \subseteq D :$ L est commandable par rapport à $PM\}$. Étant donnée que la construction de $supCN(B)$ repose sur celle de $supN(B)$, la construction est aussi exponentielle par rapport au produit de nombres d'états de B et M.

1.3.3.3 Formulation de la problématique de reconfiguration des systèmes à événements discrets

À partir du superviseur conçu dans le cas des spécifications B pour un système M, un contrôleur discret peut être construit et couplé au processus commandé. Si une défaillance arrive lors du fonctionnement du système, cette couche de commande doit être adaptée selon la situation structurelle (Modèles de fonctionnement des composants) et dynamique du système. Cette modification peut être illustrée par la Figure 1.5.

Durant le fonctionnement du système en mode standard, le système discret observé évolue de façon normale (selon le modèle discret initial) jusqu'à la détection et diagnostic d'une *anomalie*. Ainsi, le processus de diagnostic est démarré pendant une certaine durée permettant alors la modification progressive du modèle interne du système.

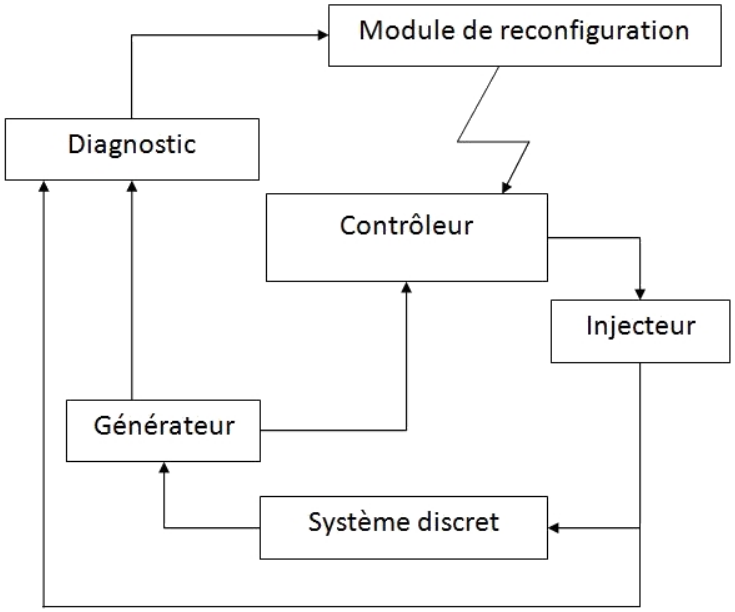

FIG. 1.5 – *Architecture globale de reconfiguration dans le cas d'un système à événements discrets*

Lorsque le système devient défaillant, des approches préventives doivent être déclenchées afin de prévenir des conséquences désastreuses, même si le diagnostic de la défaillance n'est pas fini. Par exemple, couper l'alimentation lorsqu'un appareil électrique est endommagé pour prévenir des accidents. Étant donnée que notre problématique s'intéresse à la reconfiguration en temps réel, c'est à dire sans arrêt de fonctionnement, le système doit continuer à fonctionner selon l'état courant et le système de commande standard jusqu'à la génération de nouvelles décisions. Dans ce cas, la reconfiguration doit s'appuyer sur le nouveau modèle du système et les objectifs dégradés minimaux qui lui sont associés pour adapter le fonctionnement du système à partir de son état courant.

Dans le cas d'un système discret M défini selon le paragraphe 1.3.3.2, soient Σ, Σ_o, Σ_c, Σ_{no}, Σ_{nc}, la projection P définie conformément au para-

graphe 1.3.3.2, un superviseur construit sous la forme d'un langage Π avec la réalisation en tant qu'automate $A(\Pi)$, des objectifs standards B et dégradés B'.

Supposons qu'un échec est détecté à un instant t. Les modules de diagnostic doivent permettre une mise à jour progressive du modèle perçu du système en réduisant l'incertitude. Puisque le modèle est discret, cette mise à jour est réalisée en des instants τ_i, avec $\tau_i < \tau_{i+1}$. À chaque instant τ_i, le diagnostic retourne un nouveau modèle plus *affiné* ($M(\tau_i) \subseteq M(\tau_{i-1})$). À la fin du diagnostic, on dispose du nouveau modèle de fonctionnement défaillant $M' = M(\tau_{|\tau|})$.

D'ou la formulation de la problématique de reconfiguration dans le cas discret :

Problématique 2 *La problématique de reconfiguration est de trouver à un instant $t' > t$ un langage commandable et normal par rapport à M' et B' tel que l'état du système à t' soit reconnu par le superviseur.*

1.3.4 Tolérance aux fautes des systèmes continus

1.3.4.1 Une brève introduction aux systèmes continus

La modélisation continue peut être utilisée dans le cas de systèmes complexes contrôlés de façon continue durant le temps. Ce type de modélisation est adapté pour la commande locale de composants artificiels, en contraste avec la modélisation discrète plus pratique pour une commande de haut niveau.

Souvent, un modèle continu est décrit à l'aide d'équations algébro-différentielles ou à différences. Une description algébro-différentielle d'un système continu est sous la forme :

$$\begin{pmatrix} \dot{x}_1 \\ . \\ . \\ . \\ \dot{x}_q \end{pmatrix} = \begin{pmatrix} f_1(x_1,...,x_{|x|},u_1,...,u_{|u|}) \\ . \\ . \\ . \\ f_{|f|}(x_1,...,x_{|x|},u_1,...,u_{|u|}) \end{pmatrix} \tag{1.8}$$

$x_1,...,x_{|x|}$ sont les variables d'état continus, $u_1,...,u_{|p|}$ sont les variables correspondants à l'entrée de commande. L'état perçu du système est souvent donné sous la forme d'un vecteur y dépendant de l'état continu et des entrées. Souvent, les entrées u_i n'interviennent pas dans la spécification de y. Ainsi, on a la forme usuelle :

$$\begin{pmatrix} y_1 \\ . \\ . \\ . \\ y_{|y|} \end{pmatrix} = \begin{pmatrix} g_1(x_1,...,x_{|x|}) \\ . \\ . \\ . \\ g_{|y|}(x_1,...,x_{|x|}) \end{pmatrix} \tag{1.9}$$

Afin d'assurer l'existence d'une solution pour un système décrit selon Eq. (1.8), une condition nécessaire et suffisante est que les fonctions f_i soient Lipchitziennes (Théorème de Cauchy-Lipschitz). Lorsque cette condition est vérifiée, une solution $\Phi(x_0,t)$ existe et est dépendante de l'état initial x_0 et du temps t. Un cas particulier intéressant est celui d'un système linéaire (f, g fonctions linéaires). Une introduction à la modélisation continue et aux théories de commande est donnée dans [13], [37], [9] et plusieurs autres ouvrages.

En réalité, l'évolution d'un système continu n'est pas indépendante de l'environnement voisin. En effet, un expert est souvent incapable de prendre en compte tous les paramètres entrant en jeu dans l'évolution du système

lors de l'étape de modélisation. Une solution est d'ajouter des entrées de perturbation γ_i (éventuellement bornées) selon :

$$\begin{pmatrix} \dot{x}_1 \\ . \\ . \\ . \\ \dot{x}_q \end{pmatrix} = \begin{pmatrix} f_1(x_1,...,x_{|x|},u_1,...,u_{|u|},\gamma_1,...,\gamma_{|\gamma|}) \\ . \\ . \\ . \\ f_{|f|}(x_1,...,x_{|x|},u_1,...,u_{|u|},\gamma_1,...,\gamma_{|\gamma|}) \end{pmatrix} \tag{1.10}$$

1.3.4.2 Défaillances dans le cas continu

On peut classer les défaillances en trois grandes familles :

- Défauts d'instruments : Ces défauts concernent une modification de la structure de l'injecteur (c'est à dire des actionneurs), induisant une modification de l'effet d'une commande sur l'évolution du système.
- Défauts de composants : Ceux ci concernent des modifications des propriétés physiques d'un sous ensemble des composants du système concerné. Dans le cas de Eq. (1.10), un défaut de capteur se traduit par le changement des fonctions f_i en $\hat{f}_i$.
- Défauts de capteurs : Ils se traduisent par des changements des caractéristiques des capteurs. Dans le cas de Eq. (1.9), un défaut de composant se traduit par le changement des fonctions g_i en $\hat{g}_i$.

1.3.4.3 Cas linéaire

Dans le cas linéaire :

$$\begin{aligned} \dot{x} &= Ax + Bu + E\gamma \\ y &= Cx \end{aligned} \tag{1.11}$$

A, B, C et E sont des matrices constantes réelles. Afin de pouvoir réaliser la commande/supervision d'un tel système, des observateurs doivent être implémentés afin de disposer d'une estimation $\hat{x}$. lorsqu'une loi de commande est définie par $u : t \rightarrow u(t) = K\hat{x}(t)$. Un observateur possible alors est défini par Eq. (1.12).

$$\begin{aligned} \dot{\hat{x}} &= A\hat{x}(t) + Bu(t) + L\left[y(t) - \hat{y}(t)\right] \\ \hat{y}(t) &= C\hat{x}(t) \end{aligned} \tag{1.12}$$

Il a été démontré qu'un tel observateur ($\hat{x}$,$\hat{y}$) converge vers (x,y) lorsque $A - LC$ est une matrice Hurwitz, c'est-à-dire que ses valeurs propres sont à parties réelles négatives.

Dans le cas continu, les objectifs de fonctionnement sont souvent quantifiés en tant que :

- Poursuite de trajectoire : la distance entre le vecteur des variables d'états x et la trajectoire $z(t)$ injectée au fur et à mesure du temps doit converger vers 0.
- Stabilité : Le système doit être robuste par rapport aux perturbations induites par l'environnement.

Le problème de reconfiguration conformément à la problématique 1 peut alors être résumé par:

Problématique 3 *La problématique de commande tolérante aux fautes dans le cas d'un système décrit selon Eq.* (1.10) *et disposant d'estimation d'état $\hat{x}$ est de trouver une famille de commandes U permettant de satisfaire les objectifs de fonctionnement lorsque les matrices A, B, C et E sont modifiées.*

Des méthodologies pour assurer la tolérance des fautes dans le cas linéaire sont données dans [26] et [12].

1.3.5 La problématique de reconfiguration des systèmes dynamiques hybrides

La reconfiguration des systèmes hybrides peut être considérée comme étant une généralisation du problème de reconfiguration dans le cas ou des dynamiques continues et discrètes sont présentes. Nous développerons cette partie dans le chapitre suivant.

1.4 Approches existantes pour la reconfiguration

Diverses approches sont proposées pour la reconfiguration des systèmes à événements discrets selon le formalisme de modélisation utilisé. Dans cette section, nous en donnons un bref résumé.

L'approche traditionnelle pour la reconfiguration fait appel à des opérateurs humains pour modifier le fonctionnement ou la structure du système selon les informations retournées par des modules de diagnostic (c'est ainsi que sont supervisés les systèmes de transport, les systèmes de commande de vol...). Cette approche permet à l'opérateur humain de garder un contrôle sur le fonctionnement du système en situations à risque. Cependant, la reconfiguration peut être automatique lorsque la complexité des situations appréhendées est énorme et le délai de prise de décision est court. Un cas typique est celui des systèmes de transport intelligents caractérisés par un grand nombre de véhicules et infrastructures.

Dans le cas des systèmes complexes, la reconfiguration peut être vue comme étant un changement de mode de fonctionnement en temps réel selon la *configuration* perçue du système. Ceci implique toujours que la phase de

reconfiguration doit être préparée hors ligne afin de permettre une implémentation efficiente et aisée. D'un point de vue général, les méthodologies de reconfiguration peuvent être classées selon le modèle utilisé (discret, continu, hybride), le niveau (de haut niveau : systèmes larges et très larges telle que les autoroutes automatisés ou les systèmes de commande de vol, systèmes réduits tels que des modèles à états ou des équations algébro-différentielles à nombre d'états ou de variables d'état réduits), l'intervention humaine (excluant ou non l'intervention humaine du processus de reconfiguration) et la méthodologie globale de reconfiguration (Prévision détaillée ou non détaillée des configurations et des décisions à prendre).

Par la suite, nous donnerons un bref aperçu de diverses approches pour la reconfiguration des systèmes complexes.

1.4.1 Approches de reconfiguration incluant l'intervention humaine

Les approches de reconfiguration incluant l'opérateur humain concernent souvent la supervision de larges systèmes complexes. Ceci est dû au fait que les approches actuelles d'analyse sont incapables de permettre un traitement efficace de l'explosion combinatoire induite par l'assemblage de différents composants.

1.4.1.1 Système Manufacturier Reconfigurable (SMR)

Le concept de système manufacturier reconfigurable [39, 46] a été développé dans l'université de Michigan en 1999 afin de permettre l'adaptation des systèmes manufacturiers à des changements rapides de fonctionnalités et capacités de production requises par des changements du marché, du système, de la réglementation... Cette adaptation se concrétise par des changements

de structure, matérielles et logicielles. Aujourd'hui, la plupart des entreprises industrielles utilisent un ensemble de lignes de production dédiées (ce sont des lignes de production fixes produisant des grandes quantités de pièces) et des systèmes flexibles de production manufacturière (plus couteux, mais pouvant construire différents types de produits à des fréquences et caractéristiques variées).

Le processus de reconfiguration doit être prévu lors de la construction du système manufacturier par l'utilisation de ressources réglables et avec une grande flexibilité. Les caractéristiques d'un SMR sont résumés par : Modularité, Intégrabilité, Adaptabilité, Scalabilité, Convertibilité, Diagnosticabilité (Voir [39, 46] pour une explication précise de la signification de ces caractéristiques).

Lors du fonctionnement d'un SMR, des défaillances ou réparation constituent des événements susceptibles de nécessiter une reconfiguration. Lorsqu'une ressource est défaillante, une approche possible est de la substituer par une autre ressource. Ceci n'est possible que lorsque la redondance matérielle a été prévue lors de la conception. Une approche par analyse analytique (consistant à modifier la commande des composants au lieu de changer de structure) est aussi possible. Elle présente l'avantage de permettre une plus grande flexibilité par rapport à la redondance matérielle, mais elle a pour inconvénient de ne pas être simplement implémentée et vérifiée. C'est pour cette raison que des industriels rechignent à l'implémentation d'approches par redondance analytique dans le cas de systèmes critiques.

Définissons d'abord le concept de configuration. une configuration est un uplet (R,P) ou R est l'ensemble des ressources disponibles du point de vue du diagnostic, P estl'ensemble d'objectifs de fonctionnement (ensemble de biens à produire sur l'horizon de production). La reconfiguration est déclenchée dès

la détection de l'existence d'une défaillance et est réalisée en faisant migrer la configuration du système de l'état courant inacceptable vers un état objectif. Notons que les exigences des produits à fabriquer peuvent être dégradées, ce qui permet plus de flexibilité au processus de reconfiguration.

La conception des SMR souffre des problèmes de manque de standardisation/modularité des composants et interfaces. Aussi, la construction des SMR se base largement sur le savoir faire et expertise des ingénieurs plutôt que sur des approches méthodologiques précises. De telles méthodes sont appelées méthodes par ingénierie de la commande. Évoquons par exemple :

- La méthodologie CASPAIM (Conception Assistée de Système de Production Automatisé dans l'industrie Manufacturière, Figure 1.6). Elle a été développée au LAGIS (voir [22]) et a pour objectif de proposer une architecture de contrôle/commande permettant le fonctionnement du système en mode nominal et dégradé. Le module de gestion de modes/pilotage permet la supervision et paramétrage de la fonction de commande et est conçu hors ligne. Lors du fonctionnement en ligne, la commande séquentielle permet l'implémentation de la stratégie de commande permettant de réaliser la production des produits finis. Lorsque des défaillances sont détectées, elles sont prises en compte et traitées par la fonction de recouvrement si cette défaillance a été prévue hors ligne éventuellement couplé avec un basculement en mode dégradé. Sinon, l'intervention de l'opérateur humain est nécessaire pour réaliser la maintenance du système. En plus, la fonction de détection est positionnée en tant que filtre entre la commande séquentielle et le procédé pour assurer la cohérence entre les commandes et les comptes rendus.

D'autres approches implémentées dans l'industrie seront évoquées dans l'annexe A.

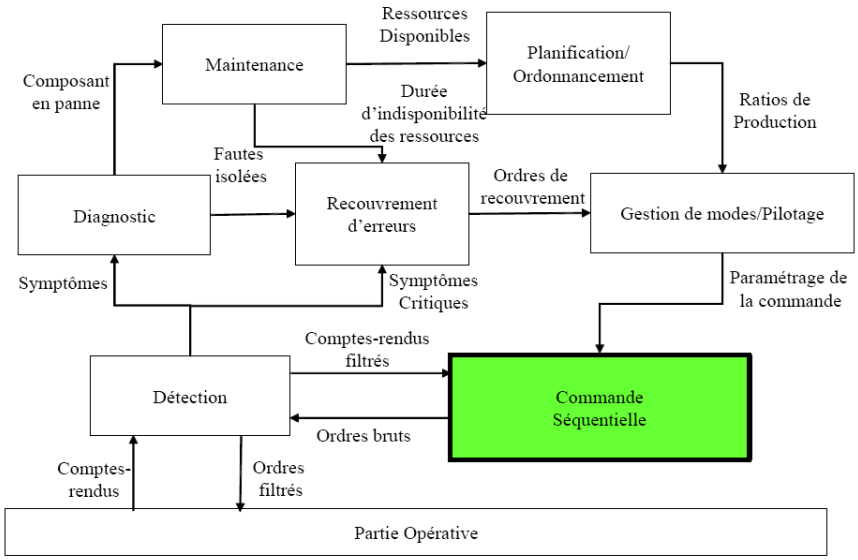

FIG. 1.6 – *Architecture générale de la méthode CASPAIM*

1.4.1.2 Systèmes de transport intelligents

Le terme de Systèmes de Transport Intelligents (STI) (en anglais Intelligent Transportation Systems (ITS)) désigne l'intégration des nouvelles technologies de la communication, de la simulation et du contrôle en temps réel dans les domaines du transport. Cette intégration a pour objectif d'améliorer la sécurité, optimiser l'utilisation des infrastructures et permettre une intégration au développement durable. Les technologies utilisées incluent les carrefours à feux, les panneaux à messages variables, les radars automatiques ou la vidéosurveillance aux applications plus avancées qui intègrent des données en temps-réel avec retours d'informations de nombreuses sources, les systèmes de dégivrage des ponts, les systèmes de navigation embarqués informant des temps de parcours en temps réel, les systèmes des autoroutes automatisés...

Les technologies utilisées dans les systèmes de transport intelligents peuvent

être décomposés essentiellement en des moyens de communications filaires ou sans fil, des véhicules et infrastructures (routes, rails...) adaptés, des calculateurs pour l'aide à la décision, des générateurs d'information telle que des capteurs et ordinateurs adaptés à l'analyse etc... Le facteur humain est cependant conservé pour la surveillance et la décision surtout au niveau stratégique (les humains sont souvent intégrés au niveau tactique aussi, par exemple la conduite de véhicules).

Parmi les différentes applications possibles, citons par exemple les systèmes des autoroutes automatisés. Ces systèmes présentent un stade avancé et ambitieux des systèmes de transport intelligents puisqu'ils excluent l'intervention humaine au niveau de la décision. Ceci est achevé par l'introduction de véhicules autonomes. L'augmentation du pourcentage d'automatisation du système permet d'accroitre la sûreté de fonctionnement puisque 90% des failles sont provoquées par des décisions humaines (estimation selon [66]). Cependant, elle implique de nouvelles exigences (de sûreté de fonctionnement, de coût, de performance...) pour les composants électroniques et informatiques embarqués dans leurs conditions de fonctionnement correspondants.

La commande de bas niveau du véhicule est assurée grâce à des contrôleurs embarqués, alors que les décisions de navigation et de communication sont assurées en collaboration avec les véhicules voisins, le meneur du peloton et éventuellement les centres de commande. Les spécifications et problèmes posés par la conception dans un tel système sont exposés dans [28].

Dans le cas des STI, la reconfiguration est une décision de haut niveau qui peut inclure ou non la décision humaine.

Des architectures pour les autoroutes automatisées tolérantes aux fautes sont proposées dans [36] et [51]. Citons aussi le projet PATH (Partners for Advanced Transit and Highways) administré par Institute of Transportation

Studies (ITS) de UCB (University of California, Berkeley). L'objectif du projet PATH est de permettre une meilleure sûreté et qualité de fonctionnement, de diminuer les embouteillages, la pollution et la consommation de l'énergie.

1.4.2 Approches pour la reconfiguration totalement automatisée

Dans cette section, nous exposons des méthodologies de reconfiguration où l'intervention humaine n'est pas essentielle. Notons que ces méthodologies n'excluent pas l'humain de la tâche de commande. Elles l'excluent surtout de la tâche de reconfiguration à la suite de défaillances.

La stratégie de reconfiguration des systèmes complexes dépend de la criticité du système considéré. Dans le cas de systèmes autonomes hautement critiques, il est courant de recourir à des méthodologies jugées fiables et facilement vérifiables. Une approche très appréciée est de recourir à la redondance matérielle. Ceci a pour effet de limiter les risques d'erreurs de conception ou des effets imprévus. Dans certains cas, le cout additionnel nécessité par une telle approche est acceptable. Une construction hiérarchisée permet aussi d'achever cet objectif en limitant les conséquences d'une défaillance à un niveau local. De telles méthodologies sont citées dans l'annexe A.

Parmi les approches de reconfiguration totalement automatisées formalisées dans le cas des systèmes dynamiques, on peut citer la reconfiguration par commutation de contrôleurs. La réalisation de la commutation peut être déclenchée grâce à un critère J dans le cas continu, ou par un module discret (Voir par exemple la proposition de [56]).

D'autres approches pour la construction d'architectures tolérantes aux fautes ont été développé. Un bref résumé est disponible dans l'annexe A.

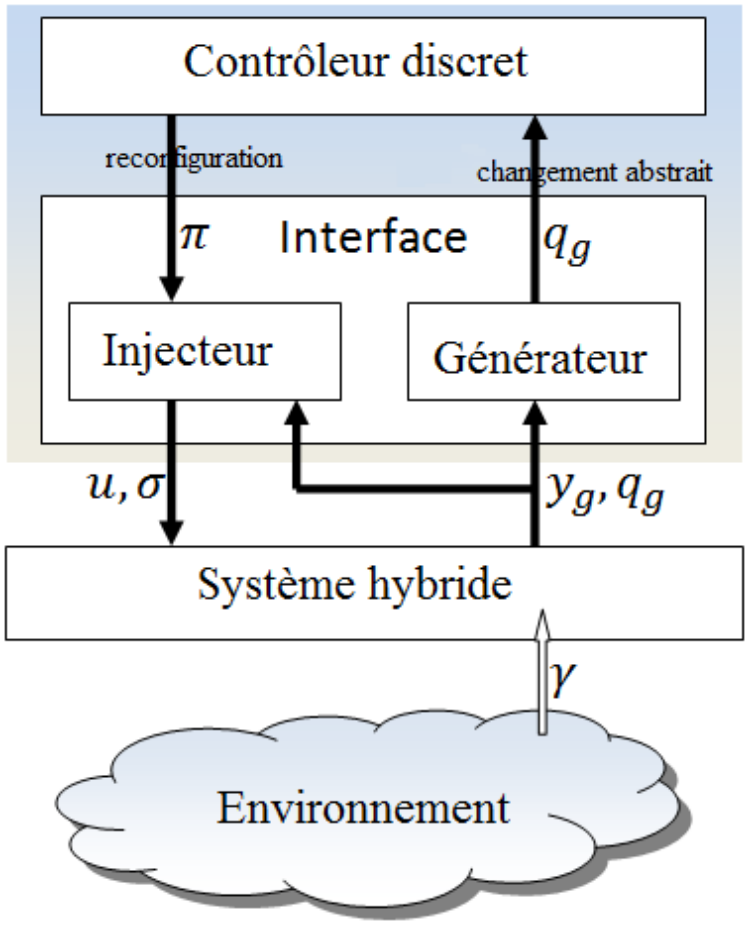

FIG. 1.7 – *Architecture globale proposée*

1.5 Architecture proposée pour la supervision et reconfiguration des systèmes complexes

Notre contribution peut être résumée comme étant la construction d'une architecture de commande tolérante aux fautes en se basant sur un modèle préliminaire hybride des différents composants du système.

L'architecture globale est resumée dans Figure 1.7.

Trois grands niveaux peuvent être distingués dans la Figure 1.7 : un bas

FIG. 1.8 – *Les étapes de construction de l'architecture proposée*

niveau contenant un modèle exhaustif du système traité, en intéraction avec l'environnement et les entrées de commande. Un niveau de commande qui est mis à jour dans le cas d'une défaillance grâce aux algorithmes d'abstraction. Enfin, un niveau de supervision implémentant les algorithmes de reconfiguration.

Les étapes globales de construction de la couche de commande sont resumées dans la Figure 1.8.

Hors ligne, il est nécessaire de réaliser une abstraction du modèle initial en un modèle discret à états finis qui puisse être exploité pour la construction d'une couche de commande (voir la section 1.3.3.2 pour un résumé de méthodologies de construction de la couche de commande discrète). Si une telle couche de commande existe, elle est mise en œuvre. Ceci permet d'assurer un fonctionnement selon les objectifs de fonctionnement standards jusqu'à l'occurrence d'une défaillance.

Une défaillance qui change le fonctionnement d'un sous ensemble de composants peut résulter en un échec de fonctionnement. Notre stratégie de reconfiguration est amorcée dès la détection de la défaillance et en fonction des informations disponibles grâce aux modules de diagnostic et d'identification. Pour celà, des algorithmes pour le calcul d'atteignabilité et d'abstraction sont implémentés dans le niveau supérieur de supervision. Les différents modules et algorithmes de calcul d'atteignabilité, d'abstraction et de reconfiguration seront détaillés dans les chapitres suivants.

1.6 Conclusion

Dans ce chapitre, nous avons exposé les principaux concepts de la discipline de la sûreté de fonctionnement et les différentes approches pour la construction d'architectures tolérantes aux fautes dans le cas discret, continu et hybride. Nous avons parcouru aussi les principales applications de la tolérance aux fautes.

En tant que procédé permettant d'assurer la tolérance aux fautes en temps réel, la reconfiguration a été envisagée comme procédure à réaliser au niveau discret (souvent le niveau discret correspond à une abstraction de haut niveau). L'adoption d'une méthodologie de modélisation continue ou hybride pour la modélisation de bas niveau est une solution pratique puisqu'elle peut être réalisée de façon simplifiée à partir de bibliothèques de modèles détaillés. Mais, un système continu ou hybride est un système à nombre d'états infinis. Et en tant que tel, il ne peut être exploité en l'état afin de prévoir toutes les évolutions possibles et choisir la meilleure solution telle que c'est le cas souvent pour les approches discrètes.

En plus, l'occurrence de défaillances s'accompagne souvent d'une incerti-

tude au niveau du fonctionnement et état interne du système. Par conséquent, la problématique de reconfiguration en temps réel dans le cas hybride ne peut être résolue qu'en répondant aux questions suivantes :

1. Quel type de modélisation peut on adopter pour la modélisation exhaustive des systèmes complexes sujets à des défaillances partielles ou totales?
2. Comment peut on approximer l'évolution d'un système à partir d'une région de l'espace de phase?
3. Comment peut on construire un modèle abstrait simple et exploitable pour la supervision et la reconfiguration de haut niveau? Comment peut on le mettre à jour lorsqu'une défaillance est suivie par un diagnostic retournant un modèle modifié, éventuellement avec des incertitudes.
4. Comment peut on mettre à jour l'architecture de commande à partir du nouveau modèle abstrait représentant le fonctionnement discret perçu après défaillance.

Par la suite, on essaiera d'apporter une réponse à chacune de ces questions dans les différentes parties de la thèse.

Chapitre 2

Modélisation Hybride des Systèmes Complexes

2.1 Introduction

Plusieurs paradigmes de modélisation ont été développés afin de permettre une analyse formelle des propriétés et possibilités des systèmes complexes dynamiques. En effet, chaque paradigme de modélisation met en évidence des propriétés et aspects particuliers qui sont profitables à une utilisation (commande, vérification, suivi de trajectoire...) prévue par les ingénieurs.

Dans le contexte de la construction de l'architecture proposée dans le chapitre précédent, il est nécessaire de représenter le fonctionnement et les défaillances du système complexe de façon exhaustive. Ceci permet de préserver un maximum d'informations pour la tâche de reconfiguration. Pour celà, une modélisation avec des équations algébro-différentielle est particulièrement adaptée pour la réprésentation de possibilités de bas niveau. Dans ce cas, une défaillance se traduit par un changement du modèle mathématique. Un exposé de la modélisation des systèmes linéaires est donné dans [73].

Étant donnée que la tâche de supervision est de haut niveau, elle nécessite un modèle discret abstrait. Souvent, une telle formalisation peut apparître selon une représentation événementielle (réseau de Petri, Automates à états finis...). Dans ce cas, une défaillance peut être représentée parfois en tant que changement de la structure discrète du modèle (changement d'état associé à la défaillance, adoption d'un nouveau modèle). Voir par exemple [69].

Les paradigmes continus et discrets sont une bonne alternative pour la représentation de la dynamique du système selon un contexte bas ou haut niveau. Cependant, Ils ne donnent qu'un aperçu limité de la structure et état du système complexe étudié. Par conséquent, l'adoption d'un formalisme *hybride* (combinant des évolutions continues et événementielles) est nécessaire. En contre partie, la supervision et reconfiguration du système nécessite la manipulation d'un modèle réduit de haut niveau *équivalent*. Par conséquent, nous développons des algorithmes d'*abstraction* pour transformer un modèle exhaustif de bas niveau en un modèle discret réduit. Voir chapitre 4.

Une autre dimension importante pour la modélisation exhaustive est la *modularité*. En effet, les systèmes complexes artificiels sont souvent construits en tant qu'assemblage de blocs élémentaires communicants. Cette modularité permet une construction rapide de modèles correspondants à des systèmes larges (C'est l'approche adoptée dans les logiciels MATLAB/SIMULINK et MODELICA/DYMOLA par exemple). Elle permet aussi une flexibilité accrue dans l'analyse et la construction des systèmes tolérants aux fautes. La construction modulaire a été proposée dans le cas discret [68]. Pour le cas hybride, on peut citer les systèmes hybrides modulaires [3], les statecharts hybrides [38] et les réseaux de Petri hybrides [24]. une modélisation modulaire des systèmes complexes est proposée dans l'annexe E.

En plus du modèle destiné à la description du système complexe, il est

nécessaire de considérer la construction de la couche de commande (voir Fig. 1.3). L'intéraction de la couche de commande doit être étudiée afin d'assurer la cohérence du modèle global. La couche de supervision sera considérée dans les chapitres suivants, et peut être ignorée à ce stade.

Dans ce chapitre, nous commençons par la définition d'un *benchmark* sur lequel nous allons nous baser par la suite. Ensuite, nous définissons des formalismes de description des systèmes complexes incorporant les trois dimensions : modularité, exhaustivité de description et description des défaillances possibles. Enfin, nous définirons le concept plus global de système à transition. Chaque formalisme proposé sera discuté et illustré grâce à l'exemple adopté.

2.2 Benchmark : le système à deux réservoirs

Le benchmark adopté pour cette thèse est une adaptation du système à trois réservoirs proposé dans [8]. Le benchmark proposé ici est composé de deux réservoirs $T1$ et $T2$, deux vannes $V1$ et $V2$ et une pompe P. voir Fig. 2.1

Le réservoir $T1$ est alimenté avec du fluide provenant de la pompe P. Deux vannes $V1$ et $V2$ permettent l'acheminement du fluide de $T1$ vers $T2$. Le système à deux réservoirs délivre un flux QN à un système en aval qui n'est pas représenté dans la figure. Les variables continues $h1$ et $h2$ correspondent aux niveaux de fluide dans les réservoirs $T1$ et $T2$.

Le flux de fluide Q_P délivré par P est géré par un régulateur PI

$$Q_P = K_P \cdot (w1 - h1(t)) + K_I \cdot \int_0^t (w1(\tau) - h1(t))dt$$

Afin de permettre une modélisation canonique, soit la variable dynamique

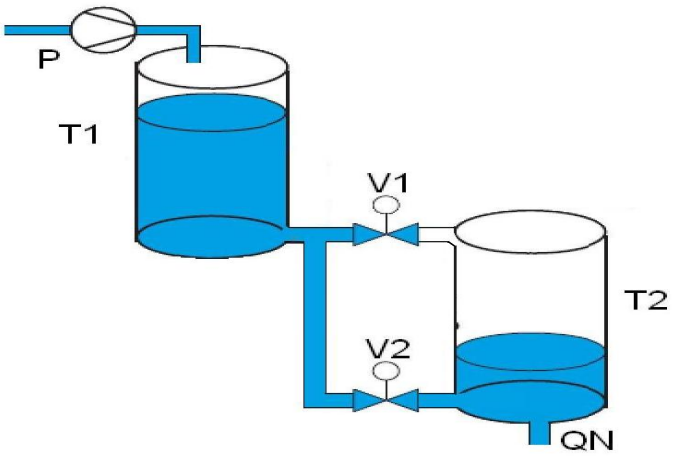

FIG. 2.1 – *Le système à deux réservoirs*

décrivant l'état interne de la pompe $Ip = K_I \int_0^t (w1(t) - h1(t))dt$.

Le niveau de fluide dans chaque réservoir évolue selon :

$$\dot{h}_i = \frac{1}{A}(\sum Q_{ini} - \sum Q_{outi}), hi \leq 0.6$$

La consommation du flux QN est définie selon : $QN = a_Z \cdot S \cdot S\sqrt{2gh_2}$.

Alors que la vanne $V2$ est raccordée à la base de $T2$, la vanne $V1$ est raccordé à une hauteur de $H = 0.3$ au réservoir $T2$. Une vanne Vi $(i = 1,2)$ est forcée en ouverture par l'événement de commande $Vi+$ et en fermeture par $Vi-$.

Les constantes utilisées dans la définition du modèle sont les mêmes dans [8]:

$$
\begin{aligned}
a_Z &= 1 \\
A &= 0.0154\ m^2 \\
g &= 9.81\ m/s^2 \\
S &= 3.6 \cdot 10^{-5}\ m^2 \\
Q_{max} &= 0.0001\ m^3/s \\
K_P &= 0.001\ 1/m \\
K_I &= 5 \cdot 10^{-6}\ 1/m\ s
\end{aligned}
$$

L'objectif assigné au système à deux réservoirs est de fournir un flux de fluide régulé au système en aval. Pour celà, nous voulons faire évoluer le niveau de fluide dans $T2$ jusqu'à l'intervalle [0.09,0.11] et le stabiliser dedans. Afin de simplifier, nous supposons que toutes les variables dynamiques sont observables. Aussi, nous supposerons que les objectifs dégradés sont les mêmes que les objectifs standards : $B' = B$.

2.3 La modélisation hybride

Dans cette section, nous introduisons de façon progressive notre approche de modélisation des systèmes complexes. Pour celà, on commence par un bref exposé de l'automate hybride. Ce formalisme a été introduit dans [32] et sert de base à la méthode de modélisation exhaustive adoptée ici.

De façon globale, un système hybride est un système qui contient des comportements continus et événementiels. Un formalisme très général pour la modélisation des systèmes hybrides a été proposé [15]. Cependant, le caractère très général du modèle proposé est peu pratique pour les tâches d'analyse

qu'on se propose de faire. Un modèle plus réduit, mais qui englobe une grande majorité des systèmes dynamiques a été proposé dans [32].

Par la suite, nous abordons l'exécution de l'automate hybride. Nous finissons par une proposition de formalisme de modélisation adapté à notre problématique.

2.3.1 L'automate hybride

Definition 12 *Un automate hybride est un uplet* $A = (Q,X,\Sigma,inv,A,f,U,E,q_0,x_0)$ *avec:*

- Q *: Un ensemble dénombrable d'états. Un cas particulièrement important est celui de* Q *contenant un nombre fini d'éléments. Soit* q *la variable qui décrit l'état du système à chaque instant.*
- X *: L'espace d'état continu avec un nombre fini de dimensions. Soit* x *le vecteur correspondant.*
- Σ *: L'ensemble fini d'événements.* $\epsilon \in \Sigma$ *est l'événement silencieux.*
- inv *: Associe à chaque état* $q \in Q$ *une* région $q.inv \subset X$.
- $A = \{q.A\}$ *: Associe une partition finie pour chaque partition d'état* $q.inv$.
- $f = \{q.f\}_{q \in Q}$ *: Pour chaque état* q. $q.f(x,u)$ *est l'ensemble de dérivées possible de* x *lorsque la commande* u. *On parle d'une* inclusion différentielle *lorsque* $q.f(x,u)$ *puisse définir plusieurs dérivées.*
- U *: entrée continue pour* A
- E *: est l'ensemble fini de transitions. Un élément* e *de* E *est de la forme* $e = (q,q',\sigma,g,reinit)$. *Il est caractérisé par une* source q, *une* destination q', *une région de garde* g *et un événement* $\sigma \in \Sigma$ *pour déclencher le franchissement de la transition.* $reinit()$ *est une fonction*

qui associe au système un ensemble de mises à jour possibles lors du franchissement.

- q_0 *: États initiaux possibles dans* Q*.*
- x_0 *: Valeurs initiales possibles de* x*.*

À la différence du modèle de Branicky, l'automate hybride utilise un espace d'état fixe lors de l'évolution dynamique, et ne met pas en oeuvre des délais de franchissement des transitions discrètes. Cependant, son expressivité est suffisante pour englober les automates temporels, les systèmes continus... Plusieurs résultats ont été avancés dans le cas des systèmes hybrides. Voir par exemple [5], [1], [30] et [33].

2.3.2 Exécution du système hybride

2.3.2.1 Le temps hybride

L'évolution des systèmes dynamiques a été souvent décrite selon un axe temporel (avec t en secondes par exemple) pour les systèmes continus ou par ordonnancement des événements (grâce à un index n) pour les systèmes à événements discrets.

Le temps événementiel référence l'index de l'étape courante par rapport à l'état qualitatif du système. En pratique, chaque hypersurface détectée, ou chaque événement $\sigma \neq \epsilon$ injecté provoque un passage de l'étape courant n à l'étape $n+1$. Par la suite, nous noterons n comme étant l'instant du début de l'étape n alors que $n(t)$ n'est autre que l'étape courante à l'instant t, lorsque t n'est pas l'instant d'une transition.

Lorsqu'on a commencé à étudier les systèmes hybrides, on s'est aperçu que certains phénomènes ne peuvent être représentés selon le temps ou l'index événementiel. Soit par exemple l'automate hybride décrit par la Figure 2.2.

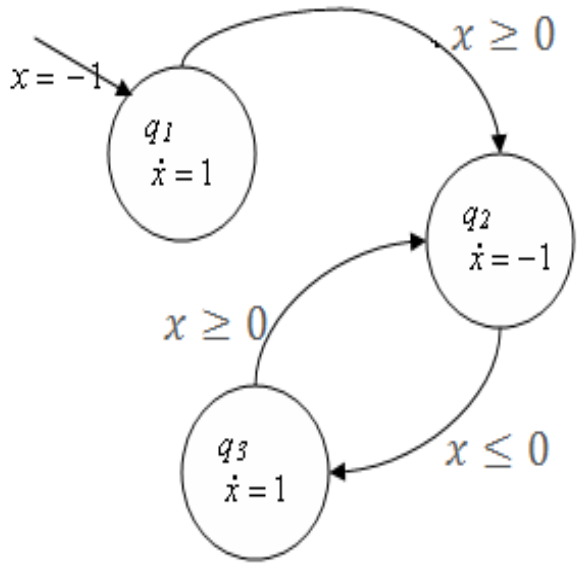

FIG. 2.2 – *Un exemple du phénomène « chattering »*

Ce système est initialisé à $t = 0$ par $x = -1$, il évolue selon la dérivée $\dot{x} = 1$ jusqu'à $x \geq 0$ à $t = 1$. À cet instant, un phénomène de « chattering »: une oscillation instantanée entre les états q_2 et q_3 est déclenchée. Ce type d'évolution ne peut être supporté selon t ou n.

Une solution qui a été proposée dans la littérature est de combiner les deux types de représentation en un temps *hybride* (voir [27], [52] et [50]). Un instant du temps hybride est un couple (t,n) avec t et n les variables correspondant au temps continu et à l'index événementiel. Dans ce cas, on parvient à représenter l'évolution dynamique du système en tant que fonction du temps hybride. Dans le cas de la Figure 2.2, le système reste à q_1 pendant l'étape $n = 1$ $(0 < t \leq 1)$. Le premier franchissement provoque l'évolution de q_1 l'instant $(t = 1,n = 1)$ vers q_2 à $(t = 1,n = 2)$ puis vers q_3 à $(t = 1,n = 3)$, ensuite vers q_2 à $(t = 1,n = 4)$... Cette représentation peut être résumée par la Figure 2.3.

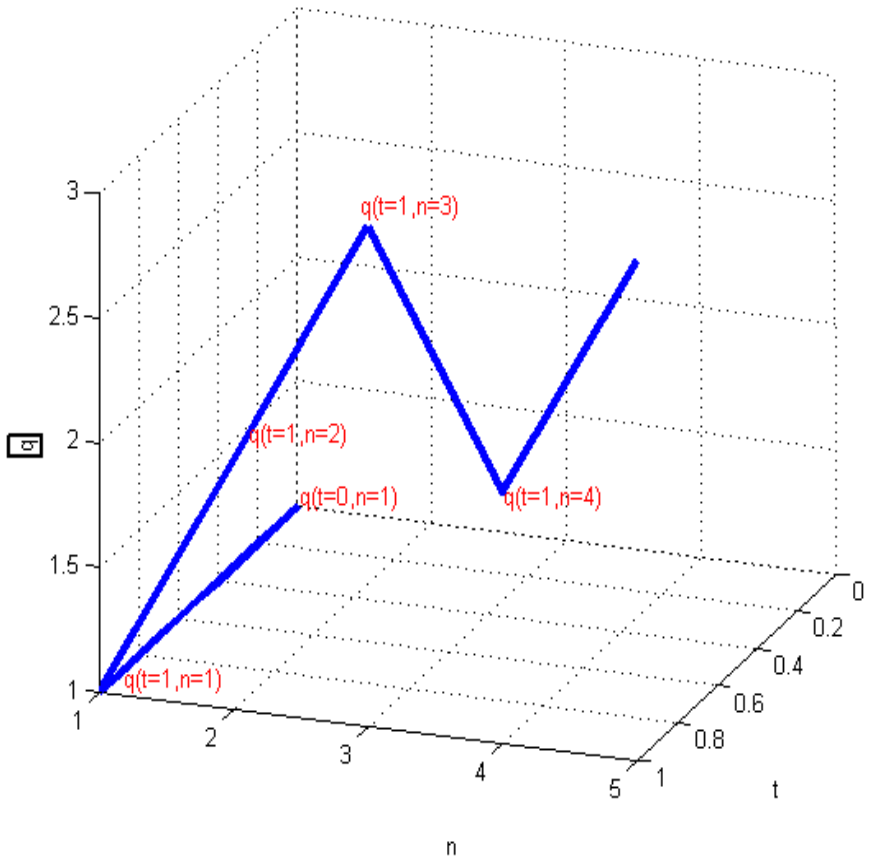

FIG. 2.3 – *Simulation en temps hybride de l'exemple de la figure 2.2*

Par la suite, nous utiliserons un temps approprié pour chaque variable utilisée:

- Le temps t peut être utilisé pour décrire l'évolution des variables pourvu qu'il n'y ait pas de transitions franchies dans le même instant.
- L'index événementiel n peut être utilisé lorsque la variable ne varie pas entre le déclenchement de deux événements. Cet index est donc particulièrement utile pour la représentation de variables du niveau discret de haut niveau telle que q.
- Si les deux représentations précédentes ne peuvent être utilisées, le temps hybride (t,n) peut être utilisé.

En plus de permettre une représentation cohérente de l'évolution des systèmes hybrides, le temps hybride a pour avantage de conserver la notion d'ordre.

Soit ϕ une fonction dépendante du temps. ϕ peut être décrite selon l'index n, t ou t,n selon la notation $\phi[n]$, $\phi(t)$ ou $\phi(t,n)$ selon les conditions citées précédemment.

2.3.2.2 L'exécution

L'exécution des automates hybrides a été traitée dans [50] et [52]. Ici, nous présentons quelques concepts.

Pour assurer l'existence et unicité de la solution continue, il est nécessaire d'avoir l'hypothèse suivante qui découle du théorème de Cauchy-Lipchitz:

Hypothèse 1 *Pour chaque $q \in Q$, la fonction correspondante $q.f$ est continue et Lipchitzienne.*

Définissons maintenant la notion de trajectoire dans le temps hybride:

Definition 13 *Une trajectoire dans le temps hybride est une séquence finie*

ou infinie d'intervalles $\tau = \{I_i\}_{i=0}^{N}$ *telle que:*

- $I_i = [\tau_i,\tau_i'], \forall i < N$;
- *Si* $N < +\infty$, *alors on a* $I_N = [\tau_N,\tau_N']$ *ou* $I_N = [\tau_N,\tau_N'[$;
- $\tau_i \leq \tau_i' = \tau_{i+1}\ \forall i$.

$\{\tau_i\}_i$ sont les instants ou des transitions discrètes sont franchies. Lorsque $\tau \neq \tau_i'$ aucune transition n'est franchie dans $]\tau_i,\tau_i'[$. Ainsi, le système est continu sur $]\tau_i,\tau_i'[$. En plus, chaque intervalle $]\tau_i,\tau_i'[$ correspond à une valeur de l'index $n = i$. Par la suite, nous notons $< I_i >=< [\tau_i,\tau_i'] >= i$, $< \tau >= \cup_i < I_i >$. Nous définissons la notion de durée d'exécution comme étant $|\tau| = \sum_{0<i\leq N} \tau_i' - \tau_i$.

Définissons maintenant l'exécution d'un automate hybride:

Definition 14 *Soit* $A = (Q,X,\Sigma,inv,A,f,U,E,q_0,x_0)$. *Une exécution de* A *est une collection* $\chi = (\tau,q,x)$ *avec* τ *étant une trajectoire en temps hybride.* $q :< \tau >\rightarrow Q$ *est une fonction constante et continue par morceaux et* $x = \{x^i : i \in < \tau >\}$ *est la collection de fonctions différentiables* $x^i : I_i \rightarrow X$ *avec:*

- $(q(0,0),x(0,0)) = (q[0],x^0(0)) \in (q_0,x_0)$;
- $\forall t \in [\tau_i,\tau_i'[,\ \dot{x}^i(t) \in f(q(i),x^i(t))$ *et* $x^i(t) \in q[i].inv$;
- $\forall i \in < \tau > -\{N\}$, *il existe une transition* $e = (q,q',\sigma,a,reinit)$, *avec* $q[i] = q, q[i+1] = q'$, $x(\tau_i',i) \in a$ *et* $x(\tau_{i+1}',i+1) \in reinit(x(\tau_i',i))$;

Par la suite, on note par $\chi_A(\hat{q},\hat{x})$ l'ensemble d'exécutions de A selon les conditions de la Définition 14 pour une initialisation $q[0] \in \hat{q}, x(0,0) \subset \hat{x}$. χ_A^* est l'ensemble d'exécutions avec une durée finie. χ_A^∞ est l'ensemble A. Grâce à cette définition de l'exécution, il est possible d'avoir la notion d'incertitude en définissant plusieurs dérivées, gardes et réinitialisations possibles. Ceci est très pratique pour modéliser la description d'un système défaillant.

2.3.2.3 Exemple

Dans le cas du système à deux réservoirs, un automate hybride peut être proposé selon la Figure 2.4. Dans ce modèle, trois variables h_1,h_2,I_P sont utilisés pour décrire l'état continu du système. En contrepartie, l'état discret est donné par un symbole de trois lettres $l_1 l_2$ avec:

- l_1 état de la vanne $V1$: O pour vanne ouverte, F pour vanne fermée.
- l_2 état de la vanne $V2$: O pour vanne ouverte, F pour vanne fermée.

a_{min} et a_{max} correspondent aux bornes maximales et minimales de la section de la vanne $V1$. Ainsi, $V1+$ (respectivement $V1-$) provoque le basculement de la section de la vanne vers a_{max} (respectivement a_{min}). Initialement, on a $a_{min} = 0$,$a_{max} = 1$.

2.3.3 Modélisation avec des défaillances

La modélisation des défaillances doit préserver les caractéristiques discrètes (défaillances complètes ou prévues lors de la modélisation discrète) et continues (défaillances partielles ou progressives). Ceci peut être réalisé en combinant les deux types de représentations : en ajoutant des états défaillants Q_F, des événements de défaillance Σ_F et des paramètres continues θ. Notre modèle peut être approché de la représentation proposée dans [41] en ajoutant à l'automate hybride dans 12 un automate de défaillances:

Definition 15 *A_Φ : l'automate de défaillances défini par $A_\Phi = (\Phi,\Phi_0,\delta_\Phi,\Sigma_\Phi,EN,DI)$ avec:*

- *Θ : l'espace des paramètres constantes. θ est le vecteur correspondant. Les fonctions continues f,g,$reinit$ dépendent de θ.*
- *Φ : l'ensemble fini de modes de défaillances.*

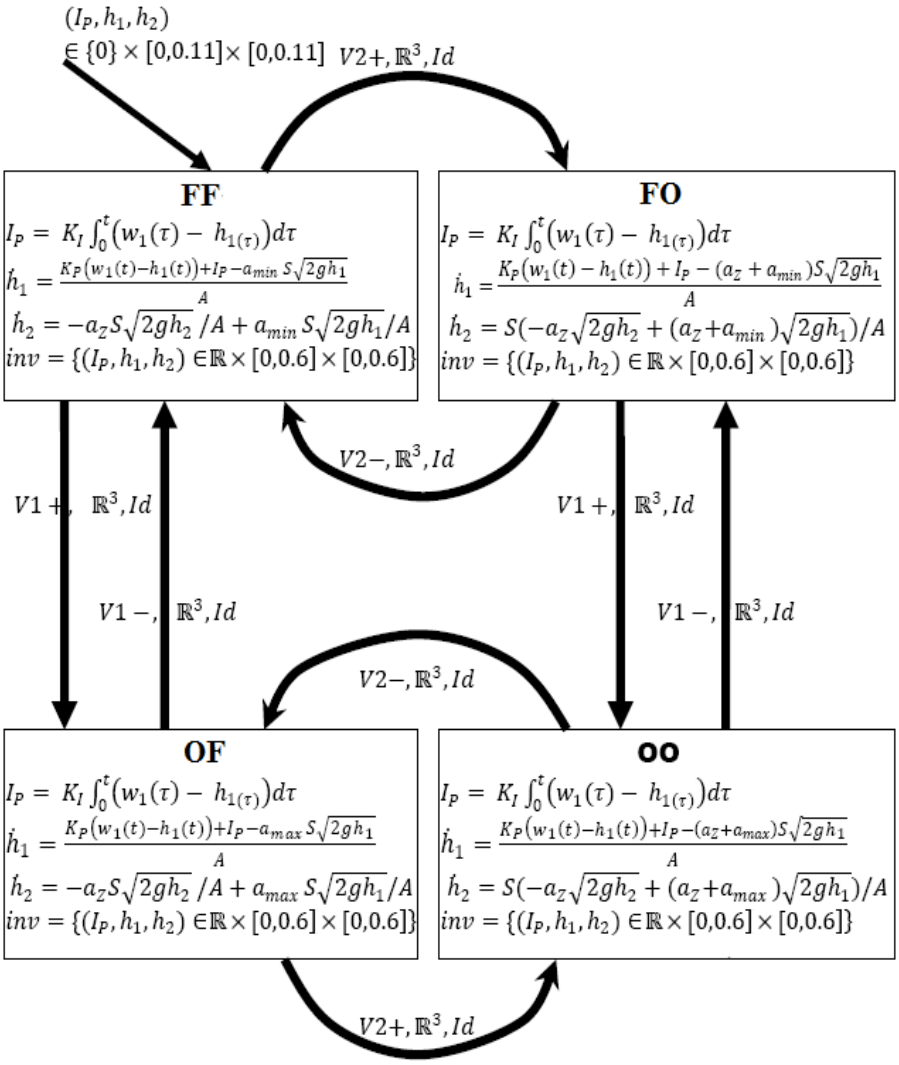

FIG. 2.4 – *L'automate hybride du comportement standard du système à deux réservoirs*

- Φ_0 *: mode initial de l'automate de défaillances correspondant à un état sain initial.*
- δ_Φ *: la fonction de transition*
- Σ_Φ *: ensemble d'événements générés par les défaillances.*
- *EN : fonction qui associe à chaque état discret de défaillance* $\phi \in \Phi$ *un ensemble de transitions activées* $EN(\phi)$ *de l'automate hybride.*
- *DI : fonction qui associe à chaque état discret de défaillance* $\phi \in \Phi$ *un ensemble de transitions* désactivées $DI(\phi)$ *de l'automate hybride.*

Σ_Φ est un ensemble d'événements générés par des modules de diagnostic. Par contre le vecteur de paramètres θ est défini par des modules spécialisés d'identification et de filtrage. Puisque les modules d'identification et diagnostic ne retournent pas nécessairement d'informations précises, il est pratique d'assigner des intervalles à θ plutôt que des points fixes de l'espace Θ. L'exécution de l'automate hybride est analogue à celle définie dans 2.3.2.2, à l'exception de l'évolution de l'automate de défaillances et des paramètres θ. Ceux-ci sont mis à jour selon des informations externes à l'automate hybride.

2.3.3.1 Atteignabilité et incertitude

Ayant défini le modèle et l'exécution de l'automate hybride avec défaillances, nous abordons le calcul d'atteignabilité et le support pour l'incertitude (qui peut être causée par des défaillances). L'état du système hybride prenant en compte les défaillances est donné par l'uplet (q,ϕ,x,θ).

En pratique un degré d'incertitude est inévitable lors de la mise à jour du modèle par les modules de diagnostic. En effet, il est plus réaliste de supposer un processus itératif de diagnostic et d'identification qui retourne un modèle structurel et états de plus en plus restreints et précis. Ainsi, (q,ϕ,x,θ) ne

désigne pas un point dans $Q \times \Phi \times X \times \Theta$, mais plutôt une région dans $2^Q \times 2^\Phi \times 2^X \times 2^\Theta$.

La modélisation orientée composants constitue une extension importante pour la description des systèmes hybrides. Étant donnée que les approches développées dans le cadre de la thèse n'exploitent pas cette extension, elle est proposée en annexe E.

2.3.3.2 Exemple

Dans le cas du système à deux réservoirs, nous supposons trois types globaux de défaillances:

- BO : Blocage de la vanne $V1$ en état ouverture, éventuellement de façon partielle.
- BF :Blocage de la vanne $V1$ en état fermeture, éventuellement de façon partielle.
- Fuite du réservoir $T1$.

Afin de pouvoir modéliser des défaillances partielles, a_{min} et a_{max} ne seront plus considérées en tant que constantes mais plutôt en tant que paramètres de défaillance.

Les défaillances ainsi prévues peuvent être décrites par la Figure 2.5. Les événements BO, BF et R correspondent respectivement à la vanne $V1$ bloquée à l'état ouverture, fermeture ou réparée.

Dans la Figure 2.5, chaque état décrit une configuration du système d'un point de vue qualitatif. En effet, chaque mode de défaillances correspond à la désactivation ou activation d'un certain nombre de transitions dans E. Notons que pour un mode de défaillance $\phi \in \Phi$, il n'est pas nécessaire d'avoir $DI(\phi) \cup EN(\phi) = E$. Dans la Figure 2.5, $DI = \{(*, *, V1-, *, *)\}$ est une

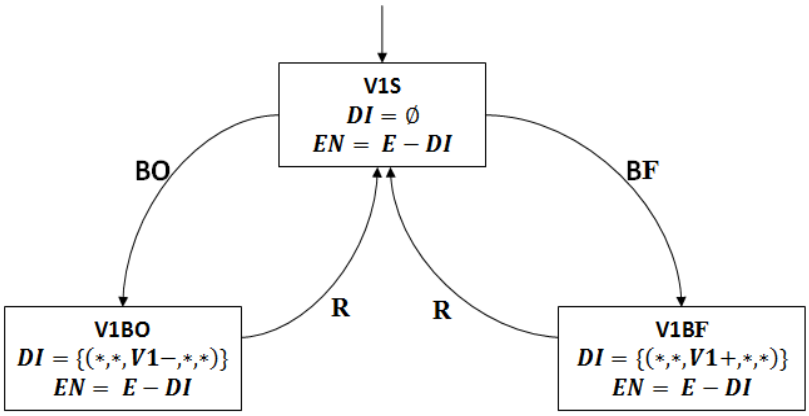

FIG. 2.5 – *L'automate de défaillances correspondant au système à deux réservoirs*

notation qui veut dire que DI est l'ensemble de transitions qui peuvent être franchies par $V1-$.

Considérons le cas des fautes partielles. Supposons que la vanne $V1$ est bloquée dans un niveau entre 0% et 40% à l'instant t_f. En plus, $h_1 = 0.5$ et $h_2 \in [0{,}0.09]$, l'état discret est FFL et $\phi = V1S$ (Aucune faute complète n'a été détectée). Dans ce cas, l'état du système est donné par:

$$(q,\phi,x,\theta) = (q,\phi,\begin{pmatrix} h_1 \\ h_2 \end{pmatrix},\begin{pmatrix} a_{max} \\ a_{min} \end{pmatrix}) = (FFL,V1S,\begin{pmatrix} \{0.5\} \\ [0{,}0.09] \end{pmatrix},\begin{pmatrix} 0 \\ 0.4 \end{pmatrix})$$

2.4 Le niveau de commande

Afin de décrire l'architecture globale de fonctionnement selon la Figure 1.7, il est nécessaire de décrire le niveau de commande et son interaction avec le système complexe. Il est aussi important d'inclure des entrées de perturbation de la part de l'environnement externe. Dans notre architecture, le niveau de commande est composé d'un contrôleur discret et d'une interface.

Le contrôleur discret implémente la stratégie de commande sous une forme symbolique. Par contre, l'interface permet l'interaction du contrôleur avec le système hybride commandé. La spécification formelle de l'architecture générale est donnée dans l'annexe E dans un cadre plus général.

L'interface se compose d'un *générateur* (permettant l'abstraction du comportement hybride en des informations qualitatives) et d'un *injecteur* (interprétant l'état courant du contrôleur discret afin de choisir une entrée de commande pour le système hybride). Une telle modélisation a été proposée dans plusieurs publications afin de permettre un contrôle discret des systèmes continus et hybride. Elle a pour avantage de proposer une passerelle entre l'automatique discrète (plutôt adaptée pour des problématiques de haut niveau, global, stratégiques avec une représentation abstraite) et continue (très pratique pour une commande ou observation de bas niveau, locale avec une représentation précise et exhaustive). Citons par exemple [63], [40]). L'architecture de commande est donnée par la Figure 2.6.

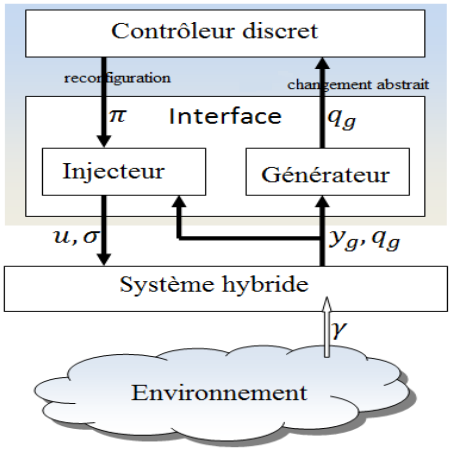

FIG. 2.6 – *L'architecture gloable de commande*

2.4.1 Le contrôleur

Definition 16 *Le contrôleur discret est un automate à états finis* $A_\Pi = (\Pi, R, \delta_\Pi, \pi_0)$ *avec:*

- Π *est l'ensemble des états de l'automate.*
- $R = 2^Q$ *est l'ensemble de parties de* Q*, avec* Q*: ensemble des états discrets du système hybride (voir les Définitions 15 et 28).* R *est l'alphabet utilisé pour la définition des transitions.*
- δ_Π *est la fonction de transition.*
- π_0 *est l'état initial du contrôleur.*

Notons que l'état du contrôleur est constant durant chaque étape n, puisque les changements de ce module sont conditionnés par des changements sur l'état du système $q[n]$. Par la suite, nous utilisons $\pi[n]$ comme étant l'état du contrôleur à l'étape n.

2.4.2 L'interface

L'interface est composé d'un *injecteur* et d'un *générateur*.

2.4.2.1 Le générateur

Le générateur a accès aux informations observables qui peuvent être délivrées par les capteurs ou les modules d'identification. Cette information *partielle* est abstraite sous forme symbolique pour le module de commande. En pratique, l'implémentation du générateur est basée sur deux ensembles importants : $Q_g \subset Q$ (états discrets observables directement) and $Y_g \subset Y$ (variables continues observables directement).

2.4.2.2 L'injecteur

Ce module interprète l'état discret du contrôleur afin de choisir une entrée hybride au système commandé. Pour cela, nous admettons l'existence d'une base de fonctions préconçues $W = (w_1,...,w_{|W|})$. Cette combinaison est réalisée selon une fonction $\alpha(\pi) = (\alpha_1(\pi),...,\alpha_{|W|}(\pi))$. L'entrée continue est définie selon Eq. (2.1).

$$\text{Quand } (t,n)\text{est définie, } u(t,n) = \sum_{i=1}^{|W|} \alpha_i(\pi(t,n))w(y_g(t,n)) \tag{2.1}$$

L'événement qui est généré pour le système commandé est donné par $\sigma(t,n)$. Si aucune transition n'est franchie à (t,n), on a $\sigma(t,n) = \epsilon$. Par contre, un passage du contrôleur discret de (t,n) vers $(t,n+1)$ se traduit par le déclenchement d'un événement selon Eq. (2.2).

$$\sigma(t,n+1) = \sigma(\pi(t,n)) \tag{2.2}$$

2.5 Conclusion

Dans ce chapitre, nous avons présenté, formalisé et illustré un formalisme de modélisation des systèmes hybrides qui supporte la description de défaillances partielles ou totales incertaines. En plus, ce modèle permet la description d'une observabilité partielle grâce au générateur. Enfin, les liens avec la partie discrète (commande et supervision de haut niveau) sont spécifiés.

Par la suite, nous utiliserons le modèle hybride de la section 2.3 pour la modélisation des systèmes hybrides. Une défaillance se traduirait par un changement de l'état et structure du système de façon analogue au traitement réalisé dans la paragraphe 2.3.3.2.

Chapitre 3

Calcul Numérique et Vérification d'Atteignabilité

3.1 Introduction

Dans ce chapitre nous abordons et proposons des méthodologies du calcul d'atteignabilité. Cette problématique est essentielle afin de pouvoir réaliser l'architecture proposée dans la Figure 1.7 et afin de répondre à la question posée à la fin du chapitre 1 : Comment peut on approximer l'évolution d'un système à partir d'une région de l'espace de phase?

Le calcul de l'espace atteignable constitue un enjeu essentiel pour l'analyse, la vérification et l'abstraction des systèmes hybrides. Elle sert à trouver la région que peut atteindre un système complexe (autonome ou commandé avec une commande fixe) à un instant précis, et à exploiter cette information pour trouver des propriétés intéressantes à propos de la sécurité d'un système ou pour réaliser une abstraction du système hybride en vue de sa commande ou de sa vérification algorithmique. En pratique, il est souvent impossible de donner une caractérisation exacte de la région atteignable par une formule

mathématique lorsqu'on a affaire à un système continu.

Cette problématique peut être résolue de façon assez simple dans le cas d'un système à états fini en explorant de façon exhaustive les états successeurs à un état donné. Par conséquent, le cas d'un système à états finis n'est pas traité ici.

Dans le cas continu, l'élaboration d'une méthode universelle nécessite d'avoir recours à des méthodes d'approximation. Ceci présente l'avantage de :

- Réaliser des calculs dans un temps fini
- Pouvoir stocker le résultat de façon informatique (Sous forme de polygone, ou assemblage de formes élémentaires telle que les ellipsoides, polytopes, zonotopes, hypercubes...).
- Présenter un résultat avec une précision et efficacité éventuellement suffisantes pour les tâches de vérification ou d'abstraction.

Diverses méthodes d'approximations de l'espace atteignable ont été présentées dans la littérature. ces techniques peuvenet être classés en :

- Des techniques exactes de calcul d'atteignabilité pour des systèmes continues ou hybrides dont les fonctions d'évolution appartiennent à des classes de fonctions linéaires. Citons par exemple [65].
- Des techniques d'approximation pour le calcul d'atteignabilité. Ces méthodologies peuvent être appliqués pour une partie plus grande des systèmes complexes. Citons par exemple l'utilisation des inclusions différentielles pour approximer l'évolution d'un système dont la fonction d'évolution est soumise à des perturbations [59], [30]. L'approximation des tubes de flux (flow pipe) a été proposée dans [20]. D'autres méthodes d'approximation sont données dans [23], [42]

Citons par exemple l'utilisation des inclusions différentielles pour approximer l'évolution d'un système dont la fonction d'évolution est soumise à des perturbations [59], [30]. L'approximation des tubes de flux (flow pipe) a été proposée dans [20]. D'autres méthodes d'approximation sont données dans [65], [23], [42]...

Ces techniques peuvent être exploitées afin de réaliser la vérification de certaines propriétés désirées (Par exemple la propriété de sécurité : le système n'évolue pas vers une région interdite) ou une abstraction (transformation du modèle hybride à dimension infinie en un modèle de dimension finie plus exploitable pour la vérification, commande et supervision). On peut trouver plus de détails et propositions d'approches dans [2], [6], [19], [21], [31], [40], [53].

Notons que le développement de méthodologies d'approximations discrètes et continues suggère la possibilité d'approximation dans le cas général hybride. Cette généralisation se base sur les techniques développées dans ce chapitre et sera réalisée dans le chapitre 5.

Dans ce chapitre, nous commençons par exposer le calcul d'atteignabilité dans le cas global (celui d'un système à transition). Après, nous donnerons une méthode de sur-approximation de l'espace atteignable selon une tolérance α voulue dans le cas continu. Nous analyserons l'efficacité de la méthode proposée et proposerons une approche permettant d'adresser le problème de l'explosion exponentielle de l'erreur. Enfin, le cas d'une mise à jour discrète sera traité. Pour implémenter les stratégies proposés dans ce chapitre, nous utilisons le logiciel MATLAB/SIMULINK.

3.2 Préliminaires

3.2.1 Le calcul d'atteignabilité des systèmes à transitions

Nous commençons par formaliser l'atteignabilité dans le cas global des *systèmes à transitions*. Ce modèle a pour avantage d'englober tous les modèles à états décrits dans la thèse. Il nous permet par la suite de définir des tâches telle que l'atteignabilité ou l'abstraction.

Definition 17 *Un système à transitions est un uplet* $(Q, \rightarrow, Q_0)$ *avec:*

- Q *: ensemble fini d'états.*
- $\rightarrow$ *: une relation qui associe à chaque élément de* Q *un ou plusieurs éléments de* $q \in Q$*.*
- Q_0 *: ensemble d'états initiaux.*

Afin d'inclure la réactivité du système par rapport à des commandes ou des observations, le concept de système à transitions étiquetées a été proposé.

Definition 18 *Un système à transitions étiquetées est un uplet* $(Q, \Sigma, \rightarrow, Q_0)$ *tel que:*

- Q *: ensemble fini d'états.*
- Σ *: ensemble d'étiquettes.*
- $\rightarrow$ *: une relation qui associe à chaque élément de* E *et à chaque événement* σ *l'ensemble vide, un ou plusieurs éléments de* Q*.*
- Q_0 *: ensemble d'états initiaux.*

À partir de cette définition, on peut définir l'ensemble E d'arcs en tant que $E = \{(q, \sigma, q'), q, \sigma \rightarrow q'\}$. Par la suite, on désigne par un système à transitions un système à transitions étiquetées $(Q, \Sigma, \rightarrow, Q_0)$.

Cette définition de système à transitions n'a pas de restrictions par rapport aux ensembles Q et Σ qui peuvent être non dénombrables. En plus, l'évolution décrite par la relation $\rightarrow$ n'est pas nécessairement déterministe.

Les opérateurs *exacts* du calcul d'atteignabilité peuvent alors être définis dans le cas global d'un système à transition.

Definition 19 *Soit le système à transitions étiqueté* $(Q,\Sigma, \rightarrow ,Q_0)$. *On définit les opérateurs exacts du calcul d'atteignabilité de la façon suivante:*

- $pre(q,\sigma) = \{q' \in Q/(q',\sigma) \rightarrow q\}$, $pre(q) = \cup_{\sigma\in\Sigma} pre(q,\sigma)$.
- $post(q,\sigma) = \{q' \in Q/(q,\sigma) \rightarrow q'\}$, $post(q) = \cup_{\sigma\in\Sigma} post(q,\sigma)$.

On désigne par *région* un ensemble de points de l'espace d'état. Les opérateurs d'atteignabilité peuvent être étendus pour des régions :

Definition 20 *Les opérateurs d'atteignabilité sont définis par:*

- $r \subset Q$, $pre(r,\sigma) = \{q' \in Q/(q',\sigma) \rightarrow q, q \in r\}$, $pre(r) = \cup pre(r,\sigma)$.
- $post(q,\sigma) = \{q' \in Q/(q,\sigma) \rightarrow q'\}$, $post(q) = \cup post(q,\sigma)$.

Dans le cas ou Q contient un nombre fini d'éléments (cas des automates à états finis), le calcul d'atteignabilité est trivial. Par contre, le cas des systèmes à transitions avec un nombre d'états infini pose un problème pratique puisqu'on ne possède plus la garantie d'un temps fini lors du parcours exhaustif du graphe. Par conséquent, nous introduisons les *opérateurs d'atteignabilité approximative*. Pour celà, il est nécessaire de définir une *distance*. Celle-ci est définie en choisissant une fonction $d : Q \times Q \rightarrow \mathbb{R}^+$ qui est conforme avec la définition mathématique d'une distance:

Definition 21 *Une distance d sur un ensemble E est une fonction sur* $E \times E$ *qui vérifie:*

- $\forall x,y \in E, d(x,y) = d(y,x)$

- $\forall x,y \in E, d(x,y) = 0 \Longleftrightarrow x = Y$
- $\forall x,y,z \in E, d(x,z) \leq d(x,y) + d(y,z)$

Soit alors les opérateurs d'atteignabilité approximative:

Definition 22 *Soit le système à transitions étiqueté* $(Q,\Sigma, \rightarrow ,Q_0)$, $\alpha \in \mathbb{R}^+$ *et* d *une fonction distance. Soient les opérateurs d'atteignabilité pre et post comme définis dans les définitions 20 et 19. On définit les ensembles de fonctions* PRE_α *et* $POST_\alpha$*:*

- $PRE_\alpha = \{pre'/\forall (q,\sigma) \in Q \times \Sigma, p' \in pre'(q,\sigma), p \in pre(q,\sigma) \Rightarrow d(p,p') \leq \alpha\}$
- $POST_\alpha = \{post'/\forall (q,\sigma) \in Q \times \Sigma, p' \in post'(q,\sigma), p \in post(q,\sigma) \Rightarrow d(p,p') \leq \alpha\}$

On peut alors étendre la définition de PRE_α et $POST_\alpha$ dans le cas des régions. En plus, nous exigerons que PRE_α et $POST_\alpha$ soient des sur-approximations, ce qui veut dire que $PRE_\alpha \subset PRE_\eta, POST_\alpha \subset POST_\eta, \forall$ $0 < \alpha < \eta$. Soit pre' (respectivement $post'$) une fonction d'approximation de PRE_α. pre' peut être étendue dans le cas d'une région $r \subset Q$ et $\sigma \in \Sigma$ telle que : $pre'(r,\sigma) = \cup\{pre'(q,\sigma), q \in r\}$.

Par la suite, nous nous fixons comme objectif de trouver des techniques pour la génération de fonctions dans $POST_\alpha$. Elles seront exploitées pour la tâche d'abstraction abordée dans le chapitre 5.

3.2.2 Problématique

Etant donnée la difficulté de réaliser un calcul exact du calcul d'atteignabilité, la problématique d'atteignabilité sera désormais de trouver une procédure de sur-approximation de l'espace atteignable.

Problématique 4 *La problématique d'atteignabilité approximative est de*

trouver un algorithme permettant de générer une fonction post $\in POST_\alpha$ *pour n'importe quel* $\alpha > 0$ *dans un temps (nombre d'itérations) fini.*

En pratique, exiger un nombre d'étapes fini pour l'approximation d'atteignabilité n'est pas suffisant. Il faut veiller aussi à la faisabilité pratique de la mise en œuvre des algorithmes mis au point. Ceci nécessite un examen approfondi de la complexité.

3.3 Le calcul d'atteignabilité continue

Dans le paragraphe 1.3.4.1 du chapitre 1, nous avons donné une brève introduction à la modélisation des systèmes continus. Par la suite, un système continu sera décrit sous la forme suivante:

$$\begin{pmatrix} \dot{x}_1 \\ . \\ . \\ . \\ \dot{x}_q \end{pmatrix} = \begin{pmatrix} f_1(x_1,...,x_{|x|},u_1,...,u_{|u|},\gamma_1,...,\gamma_{|\gamma|}) \\ . \\ . \\ . \\ f_{|f|}(x_1,...,x_{|x|},u_1,...,u_{|u|},\gamma_1,...,\gamma_{|\gamma|}) \end{pmatrix} \tag{3.1}$$

avec f : continue lipchitzienne. X est l'espace d'état associé à x (X_0 est l'ensemble de valeurs initiales possibles). U est l'ensemble de fonctions qui peuvent être associées à l'entrée de commande. Enfin, Γ est l'espace borné de paramètres possibles de perturbations.

Un système décrit par Eq. (3.1) peut être transformé sous la forme d'un système à transitions étiquetées (Q,Σ, $\rightarrow$,Q_0) avec $Q = X$ l'espace d'état, $\Sigma = U$: l'espace des fonctions de commande qui peuvent être assignées au vecteur de commande u. Soit $\phi(t,t_0,x,u)$ la fonction de flot définissant la solution au système traité à l'instant t à partir d'un état x_0 à l'instant t_0

pour une fonction de commande u.

Nous définissons alors la relation $\rightarrow$ telle que : pour $x,u,x' \in X \times U \times X$, $(x,u) \rightarrow x' \Leftrightarrow$ il existe une fonction $\gamma(t)$ dans Γ et une durée t telle que $\phi(t,0,x,u) = x'$. On peut aussi définir ϕ dans le cas négatif comme étant : $\phi(-t,0,x,u) = x'/\phi(t,0,x',u) = x$.

La problématique de calcul d'atteignabilité approximative devient : trouver un algorithme permettant de sur-approximer à α près la région atteignable sous une commande u à partir d'une région $r \subset X$. Cette région sera désignée $\phi(r,u) = \{\cup\phi(t,0,x,u),x \in r,t > 0\}$.

Cependant, nous nous sommes aperçus lors de notre recherche qu'il peut être impossible de réaliser cette tâche dans un horizon infini. Par conséquent, nous établissons un horizon maximal $T > 0$. La région atteignable à approximer devient : $post(r,u,T) = \{\cup\phi(t,0,x,u),x \in r,0 \leq t \leq T\}$ et $pre(r,u,T) = \{\cup\phi(-t,0,x,u),x \in r,0 \leq -t \leq T\}$

3.3.1 Analyse d'atteignabilité

Un algorithme d'approximation doit permettre pour une durée T, une région d'origine r, une commande v et une tolérance α une région r' telle que $\forall x \in r',\exists y \in post(r,v,T)$, $\|x - y\| \leq \alpha$. la Figure 3.1 illustre ce principe : un découpage de la région d'origine doit être réalisé avec une finesse suffisante pour obtenir la minimisation voulue de l'erreur.

Une hypothèse essentielle est d'avoir des fonctions f d'évolution Lipchitziennes par rapport à x. Ceci veut dire que pour une norme $\|\|$ on a :

$$\forall x,y,v, \|f(x,v) - f(y,v)\| \leq k \left\| \begin{pmatrix} x - y \end{pmatrix} \right\|, k > 0 \tag{3.2}$$

Une deuxième hypothèse concerne les régions étudiées lors du découpage de l'espace de phase. Dans le cadre de notre étude, ces régions doivent être

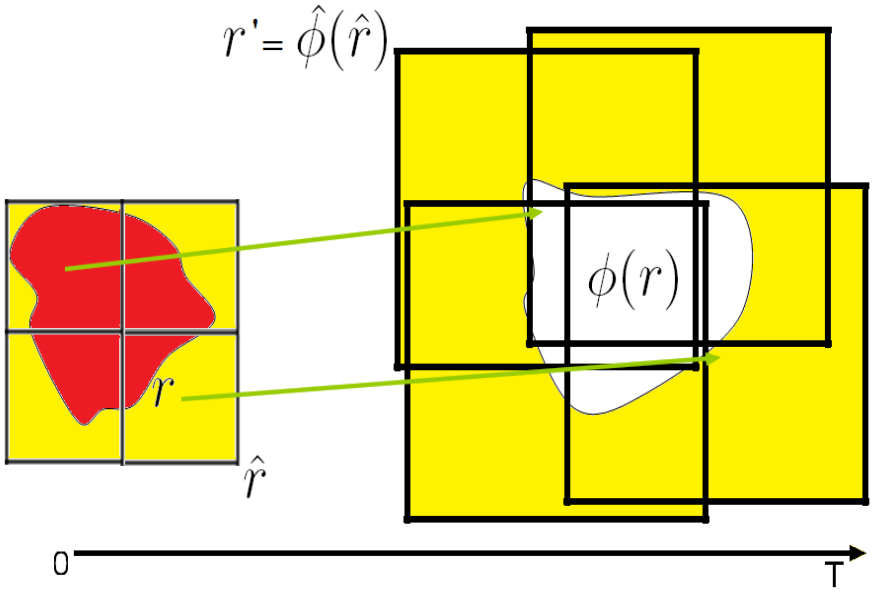

FIG. 3.1 – *Approximation à α près de l'espace atteignable*

bornées et connexes. L'hypothèse de convexité n'est pas nécessaire. Le choix de manipuler des régions bornées est justifié par notre volonté d'aboutir à un découpage en un nombre fini de *cellules*. Par contre, nous nous limitons au cas des régions connexes pour manipuler des ensembles dont l'exploitation est aisée par les algorithmes de découpage.

Notons enfin que la méthodologie d'approximation utilisée ne tient pas compte de l'erreur induite par les méthodes d'approximation numérique (cette erreur est inhérente aux logiciels informatiques de simulation). Ceci n'est pas gênant puisque les méthodes d'intégration numérique classiques sont capables de garantir une erreur inférieure à n'importe quelle tolérance voulue sur une durée donnée.

Notre méthodologie d'approximation est réalisée grâce à un découpage de la région source en des cellules de rayon maximal ϵ. C'est grâce à un choix judicieux de ϵ que l'on pourra estimer de façon assez précise la région

atteignable.

Soit l'algorithme global de sur-approximation:

Algorithme 2 *1. Calculer le rayon maximal de découpage $\hat{\epsilon}$ selon l'erreur α voulue à l'instant t.*

2. Découper la région d'origine selon $\hat{\epsilon}$ (voir par exemple l'algorithme 3).

3. Pour chaque cellule, simuler l'évolution du milieu de la cellule jusqu'à t. Soit: $x_i(t)$ le résultat de simulation du centre de la ieme cellule à l'instant t.

4. Calculer l'erreur d'approximation des cellules $r(t)$.

5. La sur approximation de la région atteignable à l'instant t est l'ensemble

$$\hat{P} = \bigcup_{y \in X} \{\exists i / \|x_i - y\| \leq r(t)\}$$

Dans ce qui suit, nous détaillerons et justifierons chaque étape dans le cas d'une évolution continue et discrète.

3.3.1.1 Découpage d'une région

Dans ce paragraphe nous présentons un algorithme pour le découpage d'une région. D'autres techniques peuvent être développées pour des régions d'une forme particulière (polyèdres, ellipsoide...).

Notons par argmin la fonction qui retourne l'élément dont l'évaluation est minimale. Par exemple, pour $f(x) = (x+1)^2$, $argmin\{f(x), x \in \Re\} = -1$, car c'est la valeur -1 qui retourne la plus petite valeur.

L'existence d'un vecteur de paramètres de perturbation γ a pour conséquence la nécessité de faire un découpage de $r \times \Gamma$ plutôt que de r. Par la suite, nous notons

$$z = \begin{pmatrix} x \\ \gamma \end{pmatrix}$$

Toutes les régions considérées intégreront les paramètres de perturbation dans leurs définitions.

Definition 23 *Nous définissons par une cellule de rayon ϵ selon une norme quelconque et centrée sur $x \in Z$ l'ensemble :*

$$c = c(x,\epsilon) = \cup\{y, \|x - y\| \leq \epsilon\}$$

Soit une région bornée $r = \{h(z) \leq 0\}$. Étant donné que nous utiliserons toujours des techniques numériques approximatives pour l'analyse des systèmes complexes étudiés, nous jugeons qu'il n'est pas important de traiter la distinction entre région ouverte et fermée. Par la suite nous utiliserons toujours des régions bornées de la forme $r = \{h(z) \leq 0\}$ de l'espace Z.

Le découpage selon un rayon ϵ de r retourne un ensemble de cellules $\{c_i(z_i,\epsilon)\}_i$ telle que $r \subset \cup\{c_i \in D\}$.

Lorsqu' il existe une fonction continue h telle que $r = \{h(z) \leq 0\} \neq \emptyset$, on a $argmin\{h(z)\} \in r$. On pourra donc utiliser un point de $argmin\{h(z)\}$ comme étant un point de départ pour trouver l'ensemble de toutes les cellules. Appelons ce point z_0.

Le découpage d'une région r en des cellules de rayon maximal ϵ peut être réalisé selon la nature des régions considérées. Par la suite nous donnons une méthode de découpage globale, itérative avec un nombre d'itérations fini pour toute région bornée $r \subset Z$.

Dans le cas d'un espace d'état à n dimensions, nous notons :

$$e_i = \begin{pmatrix} \delta_{i1} \\ . \\ . \\ . \\ \delta_{in} \end{pmatrix}$$

avec δ_{ij} le symbole de Kronecker pour la fonction qui retourne 1 lorsque $i = j$ et 0 lorsque $i \neq j$. e_i n'est autre que le vecteur unitaire selon la i^{eme} composante de l'espace X. Dans l'algorithme suivant, D contient les centres de cellules trouvées lors du découpage et $\hat{r}$ la région couverte par les cellules centrées sur les éléments de D. M est l'ensemble des sommets marqués par la procédure, c'est à dire dont on n'a pas encore exploré le voisinage.

D'une façon informelle, l'algorithme suivant construit à partir du point initial de façon récursive un découpage en cellules de rayon maximal ϵ de la région étudiée. L'algorithme explore pour chaque point s'il y'a des cellules non ajoutées dans son voisinage direct selon les différentes directions possibles (grâce aux vecteurs e_i). En effet, $z - \epsilon e_i$ est le point obtenu par décalage de ϵ selon le vecteur unitaire e_i. Par contre $z + \epsilon e_i$ correspond à un décalage positif.

Algorithme 3 *Données:*

- *Région à découper* r *avec* $r = \{h \leq 0\}$
- *Rayon maximal de découpage* ϵ
- *Point initial* z_0

Initialisation: $D = \{z_0\}$ *et* $\hat{r} = \{c(z_0,\epsilon))\}$.

Appeler Découper(r,D,$\hat{r}$*,*z_0*)*

Procédure Découper(r,D,$\hat{r}$*,z)*

Variables locales : M : ensemble de points marqués initalisé à $M = \emptyset$.
Début

1. *Répéter pour chaque dimension $i = 1..n$*
   (a) $z + \epsilon e_i \in r - \hat{r} \Rightarrow$ *ajouter* $z + \epsilon e_i$ *aux ensembles* M *et* D *si* $h(z + \epsilon e_i) \leq 0$. *Sinon, ajouter la cellule centrée sur* $\underset{\|y-z-\epsilon e_i\| \leq \epsilon}{argmin} \{\{h(y) \leq 0\} - \hat{r}\}$ *à* M *et* D *si différent de* $\emptyset$.
   (b) $z - \epsilon e_i \in r - \hat{r} \Rightarrow$ *ajouter* $z - \epsilon e_i$ *aux ensembles* M *et* D *si* $h(z + \epsilon e_i) \leq 0$. *Sinon, ajouter un point de* $\underset{\|y-z+\epsilon e_i\| \leq \epsilon}{min} \{\{h(y) \leq 0\} - \hat{r}\}$ *à* M *et* D *si différent de* $\emptyset$.
2. *Pour chaque point y de M, exécuter Découper(P,D,$\hat{P}$,y).*

Fin procédure.

3.3.1.2 Calcul d'Atteignabilité approximative continue

Dans cette partie, nous donnons les conditions suffisantes pour la résolution de la problématique énoncée précédemment grâce à une technique de simulation exhaustive. Puisque la fonction f est lipchitzienne, il existe k selon (3.2).

Proposition 1 *Dans le cas d'une évolution continue de la forme* (3.1) *et vérifiant la condition lipchitzienne selon* (3.2) *et pour toute commande $v \in V$ et région $r \in Z$. $\forall \alpha$: approximation voulue de l'espace atteignable et $t > 0$, il existe toujours un rayon maximal $\hat{\epsilon}$ tel que pour tout $\epsilon < \hat{\epsilon}$, et pour tout découpage D de r, de rayon maximal ϵ selon l'algorithme 3, on a :*

$$\forall c \in C, \text{ avec } z(0) \text{ centre de } c, \forall y(0) \in c,$$
$$\forall \tau \leq t, \|\Phi(z(0),v,\tau) - \Phi(y(0),v,\tau)\| \leq \alpha$$

Cette proposition veut dire qu'on peut toujours trouver une approximation de l'espace atteignable pour toute tolérance voulue α, en jouant sur la finesse de découpage des paramètres d'entrées et de l'espace d'origine. La preuve que nous donnons par la suite constitue aussi une procédure pour la déduction de $\hat{\epsilon}$.

Preuve. Soient les découpages C et D de l'espace d'origine avec des cellules de rayon maximal ϵ selon l'algorithme 3. Par la suite, nous caractériserons la tolérance obtenue selon l'horizon temporelle T, et la tolérance maximale voulue α. Pour cela, nous commençons en analysant l'évolution du système en fonction des découpages initiaux.

Rappelons l'inégalité triangulaire pour les normes : Quelle que soit la norme utilisée, quels que soit les vecteurs x,y, on a :

$$\|x\| - \|y\| \leq \|x - y\| \leq \|x\| + \|y\| \tag{3.3}$$

Pour une cellule $c \in C$ de centre $z(0)$ et $v \in U$, soit $y(0) \in c$. On a :

$\|z(0) - y(0)\| \leq \epsilon$

On a : $\|z(t) - y(t)\|'$

$= \lim_{dt \to 0} \frac{\|z(t+dt) - y(t+dt)\| - \|z(t) - y(t)\|}{dt}$

$\leq \lim_{dt \to 0} \frac{\|z(t+dt) - y(t+dt) - z(t) + y(t)\|}{dt}$ (Inégalité triangulaire, voir Eq. (3.3))

La fonction norme étant continue, on peut faire passer la limite à l'intérieur de la norme:

$\|z(t) - y(t)\|' \leq \left\| \lim_{dt \to 0} \frac{z(t+dt) - z(t)}{dt} - \lim_{dt \to 0} \frac{y(t+dt) - y(t)}{dt} \right\|$

$\leq \|\dot{z}(t) - \dot{y}(t)\|$

$\leq \|f(z(t),v(t)) - f(y(t),v(t))\|$ (D'après l'équation (3.1))

$\leq k \|z(t) - y(t)\|$ (D'après l'hypothèse de l'équation (3.2))

Soit l'équation différentielle définie par $r(0) = \epsilon$ et $\dot{r}(t) = kr(t)$. On a alors :

$$\forall t, \|x(t) - y(t)\| \leq r(t) \tag{3.4}$$

En effet, soit $e(t) = r(t) - \|x(t) - y(t)\|$. On a :

$$\begin{aligned}
&\dot{e}(t) = \dot{r}(t) - \|x(t) - y(t)\|' \\
&\text{Or } \|x(t) - y(t)\|' \leq \|\dot{x}(t) - \dot{y}(t)\| \\
&\text{Donc, } \dot{e}(t) \geq kr(t) - k\|x(t) - y(t)\| \\
&\Rightarrow \dot{e}(t) \geq ke(t) \\
&\Rightarrow e(t) \geq e(0)e^{kt}.\text{Or } e(0) = \epsilon - \|x(0) - y(0)\| \geq 0 \\
&\Rightarrow e(t) \geq 0
\end{aligned}$$

Donc, on obtient :

$$\forall t, \|x(t) - y(t)\| \leq \epsilon e^{kt} \tag{3.5}$$

Interprétons maintenant l'approximation donnée par l'équation (3.5). Remarquons d'abord que $r(t) = \epsilon e^{kt}$ est une fonction croissante par rapport à ϵ (Car sa dérivée est positive). Cette propriété signifie le résultat intuitif que plus le découpage (de rayon ϵ) est fin, plus l'erreur $r(t)$ est minime. En plus, pour $\epsilon = 0$, $r(t) = 0$. Remarquons aussi que $r(t)$ est croissante par rapport à t. Par conséquent:

$$\forall \tau < t, r(\tau) \leq r(t) \tag{3.6}$$

Supposons que la tolérance voulue soit α jusqu'à t. Puisque $r(t)$ est croissante par rapport à ϵ et t, la borne $\hat{\epsilon}$ peut être définie pour une approximation α par:

$$\begin{aligned} r(t) &= \alpha \\ &\Rightarrow \hat{\epsilon} = \frac{\alpha}{e^{kt}} \end{aligned}$$

D'ou

$$\hat{\epsilon} = \frac{\alpha}{e^{kt}} \tag{3.7}$$

Ainsi, on a pu trouver une borne supérieure $\hat{\epsilon}$ de l'ensemble de rayons maximaux du découpage permettant d'avoir une approximation selon un maximum réglable sur α selon la formule:

$$\hat{\epsilon} = \frac{\alpha}{e^{kt}} \tag{3.8}$$

■

3.3.2 Exemple 1

Considérons un système continu du type :

$$\begin{pmatrix} \dot{h} \\ \dot{Q} \end{pmatrix} = f(h,Q) = \begin{pmatrix} 0.1(Q-h) \\ 0.1h \end{pmatrix} \tag{3.9}$$

Dans ce qui suit nous adoptons la norme infinie :

$$\|x\| = max\{|x_i|\}$$

Montrons d'abord le caractère Lipchitzien.

On a :

$$\|f(h_1,Q_1) - f(h_2,Q_2)\| = 0.1 \left\| \begin{pmatrix} Q_1 - h_1 - Q_2 + h_2 \\ h_1 - h_2 \end{pmatrix} \right\|$$

(Inégalité Triangulaire, voir 3.3)

$$\begin{aligned} &\leq 0.1 \left\| \begin{pmatrix} Q_1 - Q_2 \\ 0 \end{pmatrix} \right\| + 0.1 \left\| \begin{pmatrix} -h_1 + h_2 \\ h_1 - h_2 \end{pmatrix} \right\| \\ &\leq 0.2 max\{|Q_1 - Q_2|,|h_1 - h_2|\} \end{aligned}$$

D'ou :

$$\|f(h_1,Q_1) - f(h_2,Q_2)\| \leq 0.2 \left\| \begin{pmatrix} h_1 - h_2 \\ Q_1 - Q_2 \end{pmatrix} \right\| \tag{3.10}$$

Soit une région origine $0 \leq h \leq 0.1, 0 \leq Q \leq 0.4$ Supposons qu'on veuille estimer la région atteignable entre 0 et 5 secondes avec une tolérance de 0.01. Soit :

$$\begin{aligned} \alpha &= 0.01 \\ k &= 0.2 \\ Tmin &= 0 \\ Tmax &= 5 \end{aligned}$$

D'après l'équation définissant $\hat{\epsilon}$ (voir Eq. 3.7), on doit choisir un découpage avec un rayon maximal de

$$\hat{\epsilon} = \frac{\alpha}{e^{k \cdot Tmax}} = \frac{0.01}{e^{0.2*5}} = 0.0037 \tag{3.11}$$

3.4 Approche itérative pour la simulation exhaustive

3.4.1 Limites de l'approche proposée

L'approche proposée dans la section 3.3 possède l'inconvénient majeur de ne pas être pratique dans des cas plus élaborés à cause de la divergence exponentielle de l'estimation d'erreur. Dans le cas de l'exemple 1 du paragraphe 3.3.2, il aurait suffit de choisir $Tmax = 100$ au lieu de $Tmax = 5$ pour trouver un rayon maximal trop petit (Pour $Tmax = 40$, on a $\hat{\epsilon} = 3.35 \times 10^{-6}$. Pour $Tmax = 100$, on a $\hat{\epsilon} = 2.06 \times 10^{-11}$).

3.4.2 Approche itérative

Dans ce qui suit, nous essayerons d'énoncer une méthode pour traiter ce problème. Remarquons d'abord que le terme $e(t) = \epsilon e^{kt}$ est dépendant linéairement en ϵ et exponentiellement en t. Ainsi, la réduction de $e(t)$ pourrait être réalisée de façon plus « rapide »en réduisant les durées pour l'estimation de l'espace atteignable qu'en affinant ϵ.

Soit m le nombre d'itérations d'approximations, T : l'horizon d'approximation final. Soit $T_i = i * T/m$. Supposons un découpage initial avec un rayon maximal ϵ. Pour la première période [0,T_1], on a l'erreur maximale : $r(T_1) = \epsilon e^{kT_1}$. Soit r_1 la région trouvée par la première approximation.

En itérant pour chaque i :

1. On choisit un découpage D_{i-1} de rayon maximal ϵ de P_{i-1}
2. On construit une approximation à ϵ près de $\Phi(P_{i-1},T_{i+1} - T_i)$ grâce à D_{i-1}.

Théoriquement, on ne peut pas avoir la garantie d'avoir une meilleure to-

lérance que la méthode non itérative. Cependant, itérer la simulation permet de limiter l'explosion de l'erreur à chaque étape puisqu'elle ne sera jamais plus grande que le seuil $r(T/m)$ à partir de l'approximation précédente. Dans la pratique, ceci permet de réduire considérablement l'explosion de l'erreur.

3.4.3 Application sur l'exemple 1

Considérons l'exemple du paragraphe 3.3.2 de nouveau et supposons que $T_{max} = 20$. Pour $\epsilon = 0.037$, on a $r(20) = \epsilon e^{0.2*20} = 2.02$. Voir les Figures 3.2, 3.3 pour l'évolution de l'approximation de l'espace atteignable dans le cas simple. Ces figures représentent les différentes trajectoires à partir de chaque cellule avec une couleur différente, et précise aussi l'approximation de l'espace atteignable à $t = 10$ et $t = 20$ secondes.

Supposons qu'on utilise une borne $\epsilon = 0.037$ selon 2 périodes avec $T_1 = 10$,$T_2 = 20$.

- Période 1 : Entre 0 et $T_1 = 10$, L'erreur initiale est de $\alpha_1 = 0.27$.
- Période 2 : Entre $T_1 = 10$ et $T_2 = 20$, L'erreur est de $\alpha_1 = 0.27$ (Puisque le deuxième découpage utilise le même ϵ comme rayon maximal d'une cellule).

D'ou la surface atteignable représentée par les Figures 3.4, 3.5.

Ces figures montrent différentes vues de l'évolution de la région atteignable approximée à 10 secondes et 20 secondes. En comparant l'approximation simple sur la figure 3.3 et l'approximation itérative de la figure 3.5, on remarque que l'approximation itérative permet, à partir d'un même rayon maximal de découpage ϵ, d'aboutir à une approximation sensiblement plus restreinte, donc plus précise.

La méthode itérative permet un gain en efficacité d'approximation en

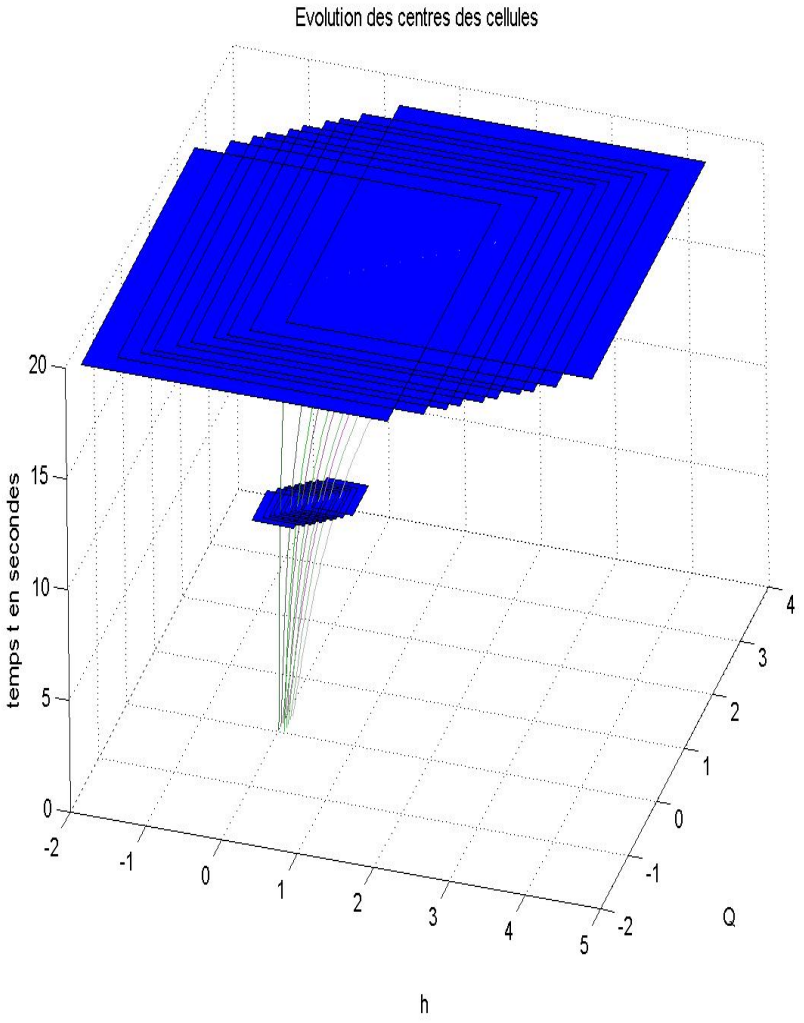

FIG. 3.2 – *Vue 1 de la génération simple de la région atteignable au bout de 20 secondes*

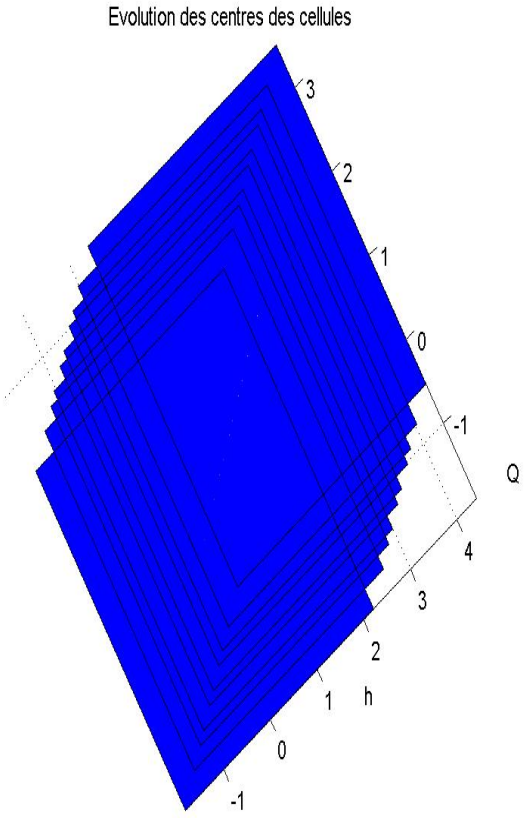

FIG. 3.3 – *Vue 3 de la génération simple de la région atteignable au bout de 20 secondes*

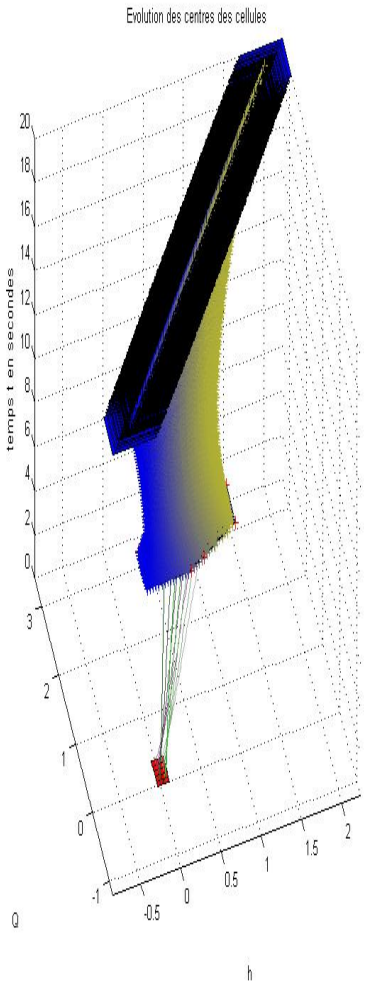

FIG. 3.4 – *Vue 1 de la génération itérative à 2 étapes de la région atteignable au bout de 20 secondes*

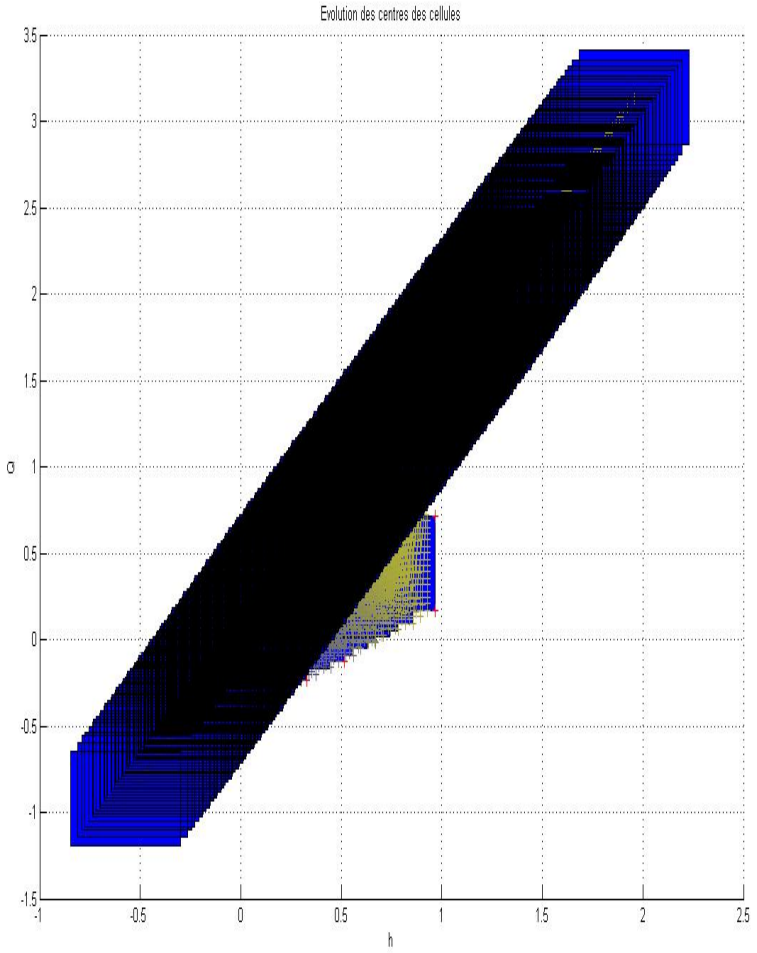

FIG. 3.5 – *Vue 3 de la génération itérative à 2 étapes de la région atteignable au bout de 20 secondes*

interdisant la divergence exponentielle au delà de la durée utilisée pour le découpage de l'horizon temporel de simulation. L'inconvénient majeur de cette méthode est de ne pas présenter de formule permettant la génération de $\hat{\epsilon}$ de façon automatique.

Une autre ilustration de l'approche d'atteignabilité est réalisée dans l'annexe B pour l'exemple du pendule inversé.

3.5 Calcul d'atteignabilité approximative dans le cas de mise à jour discrète

Nous appelons une mise à jour discrète la réinitialisation des variables d'état d'un système hybride lorsqu'une transition $e = (q,q',\sigma,g,reinit)$ est franchie. Les variables sont alors réinitialisés selon $x := reinit(x)$.

À la différence de l'ensemble de fonctions d'atteignabilité continues approximatives, le calcul d'atteignabilité lors d'une mise à jour discrète peut être construit de la façon illustrée dans l'algorithme 4. Nous avons seulement besoin d'une constante k telle que *reinit* soit k-Lipchitzienne:

$$\forall x,y \in X \times X, \|reinit(x) - reinit(y)\| \leq k\|x - y\| \tag{3.12}$$

La condition décrite par l'équation (3.12) est une hypothèse fondamentale et qui n'est pas trop restrictive. Pour faire en sorte que l'erreur soit inférieure à α, il suffit d'imposer un découpage avec un rayon maximal ϵ tel que:

$$\forall x,y / \|x - y\| \leq \epsilon, \|reinit(x) - reinit(y)\| \leq \alpha \tag{3.13}$$

Eq. 3.12 et Eq. 3.13 impliquent la condition nécessaire suivante:

$$k\epsilon \leq \alpha \Rightarrow \epsilon \leq \alpha/k \tag{3.14}$$

D'ou le choix de la tolérance maximale $\hat{\epsilon} = \alpha/k$

Algorithme 4 *fonction* $\phi_D(A,q,x,t)$ *Début*

1. $\forall e = (q,q',g,\sigma,reinit), x \in g$:
   - *découper* x *avec des cellules à rayons maximales* $\hat{\epsilon} = \alpha/k$
   - *Calculer pour chaque cellule l'image de son centre* $\{x_i\}$ *par reinit.*
   - *L'approximation de l'espace atteignable est* $(q', \cup_i \{x \in X / \|x - reinit(x_i)\| \leq \alpha\})$. *Nous définissons la région succ :*

$$succ := \bigcup (q', \cup_i \{x \in X / \|x - reinit(x_i)\| \leq \epsilon\})$$

2. *retourner* $\phi_D(H,q,x,t) := succ$

Fin

3.6 Conclusion

Nous avons présenté deux méthodes d'approximation de l'espace atteignable dans le cas continu et discret. Nous avons montré qu'on peut trouver un découpage minimal de la région origine permettant de construire une approximation qui soit proche de la région atteignable réelle, quelle que soit la précision voulue. Cette garantie théorique n'est cependant pas une solution pratique pour l'approximation du cas continu à cause de la divergence exponentielle du terme de l'erreur.

Pour remédier à cette limitation, nous avons présenté une méthode itérative qui limite cette divergence en réalisant un découpage et simulation périodiques. Cette méthode peut être applicable même dans le cas de grands

horizons temporels, mais présente l'inconvénient de ne pas donner de garantie théorique quant à la croissance de l'erreur.

La combinaison de l'atteignabilité continue et discrète approximative garantit la possibilité de calcul approximatif de la région atteignable dans le cas d'un horizon temporel et d'un nombre de sauts finis pour les systèmes hybrides dont les fonctions sont Lipchitziennes. Ce calcul d'atteignabilité est aussi possible lorsqu'on inclut une dimension incertaine en enrichissant les régions r dans X en $r \times \Gamma \in X \times \Gamma$. Par la suite, nous utiliserons les techniques développées dans le cadre de ce chapitre afin de réaliser des tâches de haut niveau.

Chapitre 4

Génération et Vérification Automatique de Stabilité

4.1 Introduction

La génération de commande stabilisatrice par rapport à une région est un sujet important puisqu'elle permet de restreindre le fonctionnement du système dans un ensemble de comportements désirables, ce qui est nécessaire pour garantir un fonctionnement satisfaisant du système. Une telle propriété est essentielle afin de limiter les risques de comportements erratiques dus à des perturbations externes.

Disposer d'un outil permettant de réaliser une telle propriété est essentiel dans notre thèse. En effet, la reconfiguration d'un contrôleur nécessite de pouvoir *stabiliser* le fonctionnement du système dans une configuration précise. Dans ce cadre, ce chapitre peut être considéré comme un prolongement du chapitre précédent, puisqu'il fournit les outils de bas niveau (calcul d'atteignabilité et vérification de stabilité) qui pourront être exploités par les tâches de haut niveau (abstraction et reconfiguration).

De façon classique, la théorie de la stabilité a été développée dans le cas des systèmes continus grâce aux théorèmes de Lyapunov. La stabilité d'un système continu peut être garantie en trouvant une fonction de Lyapunov (deuxième théorème de Lyapunov), ce qui permet de prouver de façon formelle les différents types de stabilité (asymptotique, globale, locale, exponentielle...). Voir par exemple [61].

Plusieurs méthodologies pour la vérification de stabilité ont été développées pour des cas plus généraux tels que les systèmes hybrides (par exemple grâce aux fonctions multiples de Lyapunov [17]). D'autres méthodologies concernent des cas particuliers: Par exemple [34] pour les systèmes à commutations...

Un inconvénient majeur de ces méthodologies est qu'elles nécessitent de trouver la fonction de Lyapunov et la fonction de commande stabilisatrice. Cette tâche ne peut être conduite de façon systématique. En effet, elle dépend largement de l'habileté de l'expert et nécessite souvent un long temps pour l'analyse.

Dans ce chapitre, nous développons une méthodologie pour la vérification et la génération de commande pour la stabilisation dans une région de l'espace de phase de façon automatique. En pratique, la génération de commande permet d'assurer l'invariance de la région considérée en s'assurant que tous les points de fonctionnement de la frontière ne peuvent quitter la région sous l'action de la commande stabilisatrice. Enfin, nous illustrerons cette méthodologie sur un exemple.

4.2 Problématique

4.2.1 Problématique générale de stabilisation d'un système à transitions

Commençons par une formulation globale du problème de stabilisation dans une région d'un système à transitions. En général, stabiliser le fonctionnement d'un système dans une région veut dire que le point de fonctionnement finit par entrer et rester dans cette région sous l'effet de la commande stabilisatrice. Dans le cadre de notre thèse nous adoptons une définition plus restrictive de la stabilisation: stabiliser le fonctionnement dans une région veut dire que le point de fonctionnement ne peut jamais quitter la région sous l'effet de la commande stabilisatrice. Plus formellement:

Definition 24 *Soit* $(Q,E,\Sigma, \rightarrow ,Q_0)$ *un système à transitions étiqueté et* $u : Q \rightarrow \Sigma$ *une fonction associant une loi de commande dans* Σ *au système selon le point de fonctionnement actuel. Soit* $r \subset Q$ *une région de* Q*. On dit que* u *stabilise le fonctionnement du système dans* r *lorsque* $\forall q \in r, \forall q' \in Q, (q,u(q)) \rightarrow q' \Rightarrow q' \in r$*.*

Ce choix a été motivé par la nécessité d'exploiter les propriétés définies en bas niveau lors de l'abstraction. En effet, adopter la définition générale de la stabilité ne nous permet pas d'interdire certaines régions de l'espace de phase (puisque le point de fonctionnement peut osciller dans tout l'espace de phase avant de se stabiliser dans une région précise).

4.2.2 Problématique de stabilisation continue

Un système continu est défini sous la forme:

$$\dot{x}(t) = f(x(t),u(t)) \tag{4.1}$$

$x(t)$ est la variable d'état continue, $u(t)$ est l'entrée de commande. f est une fonction retournant un ensemble de dérivées possible (Ce qui permet de prendre en compte des incertitudes ou perturbations).

Soit une région $r = \{h(x) \leq 0\}$. r est invariante sous une commande u si toute trajectoire commençant dans r reste toujours dans r.

Proposition 2 *Une fonction de commande u stabilise le fonctionnement du système dans $r = \{h(x) \leq 0\}$ si pour chaque point x vérifiant $h(x) = 0$, on a $\nabla h(x) \times f(x,u(x)) < 0$.*

la Figure 4.1 illustre le principe de la proposition 2. $h(x) = 0$ veut dire que le point x appartient à la surface délimitant la région $r = \{h(x) \leq 0\}$. Dans ce cas, $\nabla h(x)$ est un vecteur qui est normal à la tangente à la surface en x.

$\nabla h(x) \times f(x,u) < 0$ implique que pour $x(t) = x, x(t+dt) \approx x(t) + f(x(t),u(t))dt$. Donc, $h(x(t+dt)) = h(x(t) + f(x(t),u(t))dt) = h(x(t)) + \nabla h(x) \times f(x(t),u(t))dt$. Or $h(x(t)) = 0$ d'ou :

$$h(x(t+dt)) = \nabla h(x) \times f(x(t),u(t))dt < 0$$

Ainsi, le problème de stabilisation du système dans r se réduit à trouver une commande u vérifiant la condition de Prop. 2.

Pour réaliser la stabilisation dans une région de l'espace d'état continu, nous supposons l'existence d'une famille finie de commandes possibles à injecter $U = \{u_i, 1 \leq i \leq |U|\}$.

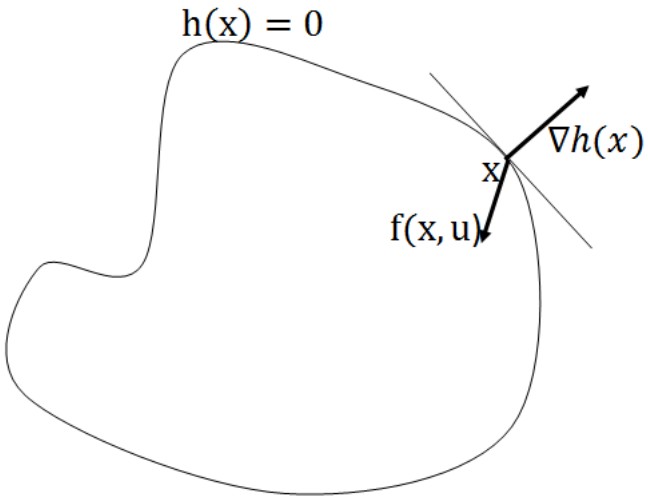

FIG. 4.1 – *Illustration de la vérification de stabilité dans le voisinage d'un point x selon Prop. 2*

4.2.3 Problématique de stabilisation d'un système hybride

Considérons un système hybride conforme à la Définition 15. L'espace de phase étant $Q \times X$, une région de l'espace de phase peut être décrite sous la forme $r = \cup(q_i,r_i)$, où q_i est l'état discret et r_i la région de X associée par r à l'état q.

Dans le cas d'un système hybride, il peut être impossible de trouver une commande stabilisatrice dans r à cause de la possibilité de transitions autonomes qui mettent à jour le point de fonctionnement du système en dehors de r. Aussi la stabilisation dans une région du système hybride revient à:

1. Trouver une sous région $r' = \cup(q_i,r'_i)$ de r telle que toute transition autonome dans $r' =$ provoque une évolution dans r'.
2. Pour chaque état q_i, trouver la commande continue permettant de sta-

biliser l'évolution continue dans r'_i.

4.2.4 Hypothèses

Puisqu'on travaille dans le contexte incertain nous utiliserons désormais l'espace $Z = X \times \Gamma$.

Les hypothèses dont nous avons besoin sont essentiellement le caractère Lipchitzien et borné des fonctions f et ∇h. Ces hypothèses permettent de limiter le changement de l'expression $\nabla h(z) \times f(z,u(z))$ dans le voisinage de z (voir la section suivante pour plus de détails).

4.3 Vérification de l'invariance continue

Une région r est dite *invariante* avec une commande u lorsque le point de fonctionnement ne quitte pas r lorsque le système est commandé par u.

Afin de réaliser une approche automatique de vérification d'invariance, nous nous basons sur la proposition 2. Dans ce paragraphe, nous montrerons que notre méthodologie de vérification de la propriété $\nabla h(z) \times f(z,u(z)) < 0$ peut être exécutée de façon automatique en un nombre fini d'opérations lorsque le caractère Lipchitzien et borné des fonctions continues utilisées est assuré.

Soit $\|\|$ une norme quelconque. Nous définissons la notion de cellule dans $Z = X \times \Gamma$ $c(z,\epsilon)$ de façon analogue à la Définition 23 du chapitre 3.

Soit $x \in Z$ un point vérifiant $h(x) = 0$ et $\nabla h(x) \times f(x,u(x)) < 0$. Essayons de trouver la cellule autour de x vérifiant la condition de la proposition 2.

Soit la fonction $N : X \to \mathbb{R}$ qui associe à chaque point $z \in X$ la valeur $N(z) = \nabla h(z) \times f(z,u(z))$. Supposons que $N(x,u) < 0$. Soit $y = x + d$ un point de X vérifiant $h(y) = 0$. On a:

$$\|N(y,u) - N(x,u)\| = \|\nabla h(y) \times f(y,u(y)) - \nabla h(x) \times f(x,u(x))\|$$
$$= \|\nabla h(y) \times f(y,u(y)) - \nabla h(y) \times f(x,u(x))$$
$$+\nabla h(y) \times f(x,u(x)) - \nabla h(x) \times f(x,u(x))\|$$
$$\leq \|\nabla h(y) \times f(y,u(y)) - \nabla h(y) \times f(x,u(x))\|$$
$$+ \|\nabla h(y) \times f(x,u(x)) - \nabla h(x) \times f(x,u(x))\|$$
$$\leq \|\nabla h(y)\| \, \|f(y,u(y)) - f(x,u(x))\| + \|f(x,u(x))\| \, \|\nabla h(x) - \nabla h(y)\|$$

D'après les hypothèses, il existe n,k,m,l telle que $\|\nabla h(x)\| \leq l$, $f(x,u) \leq m$, $\|f(x,u) - f(y,u)\| \leq k \, \|x - y\|$ et $\|\nabla h(x) - \nabla h(y)\| \leq l \, \|x - y\|$ $\forall x,y$, on alors:

$$\|N(y,u) - N(x,u)\| \leq n \, \|f(y,u(y)) - f(x,u(x))\| + m \, \|\nabla h(x) - \nabla h(y)\|$$
$$\leq nk \, \|y - x\| + ml \, \|y - x\|$$
$$\leq (nk + ml) \, \|y - x\|$$

Soit $K = (nk + ml)$. Ainsi, la fonction $N(x,u) = \nabla h(x) \times f(x,u(x))$ est K-Lipchitzienne.

Après avoir montré que la fonction N est Lipchitzienne, montrons que pour $N(x,u)$ strictement négative, il existe un voisinage de x vérifiant aussi cette propriété. Afin de pouvoir assurer une stabilité stricte, soit η un réel positif très petit. Notre objectif est de s'assurer que $N(y,u) \leq -\eta$ (au lieu de $N(y,u) < 0$).

Soit $y \in X$. On a:

$$\begin{aligned} N(y,u) &= N(y,u) - N(x,u) + N(x,u) \\ &\leq |N(y,u) - N(x,u)| + N(x,u) \\ &\leq K \|y - x\| + N(x,u) \end{aligned}$$

Ainsi, lorsque $N(x,u) < 0$, $N(y,u) \leq -\eta$ nécessite

$$\begin{aligned} K \|y - x\| + N(x,u) &\leq -\eta \\ \Rightarrow \|y - x\| &\leq (-N(x,u) - \eta)/K \end{aligned}$$

Ainsi, chaque point de la cellule $c(x,(-N(x,u)-\eta)/K)$ vérifie la condition de Prop. 2.

D'ou l'algorithme de vérification :

Algorithme 5 *Données :*

- *Une région* $r = \{h(x) \leq 0\}$, ∇h *étant une fonction l-Lipchitzienne dont la norme est majorée par* N. *Appelons* δr *la frontière de* r ($\delta r = \{x,h(x) = 0\}$). *Soit* x_0 *un point de* δr.
- *Une fonction* $f : X \times U$ *définissant l'évolution du point de fonctionnement. Cette fonction est k-lipchitzienne. En plus, sa norme est majorée par* M.
- *Un réel positif très petit* η.

Résultat : vrai si la région r *est invariante sous* u, *faux sinon.*

Initialisation : Soit S *correspondant à la surface couverte par l'algorithme à chaque itération.* S *est initialisé à* $\emptyset$. *Soit* x *un point de* δr.

Répéter Si $N(x,u(x)) = \nabla h(x) \times f(x,u(x)) < -\eta$ *alors*

- *Calculer* $\epsilon = (-N(x,u) - \eta)/K$.
- $S = S \cup (c(x,\epsilon) \cap \delta r)$

– *Choisir un point x de la frontière de S et passer à l'itération suivante Sinon ECHEC : retourner $faux$ Jusqu'à $\delta r \subseteq S$: SUCCÈS. Retourner vrai.*

L'algorithme 5 peut être implémenté en un nombre fini d'étapes si deux conditions sont vérifiées:

– La surface δr est finie.

– Il existe α telle que $\alpha > \eta$ et $N(x,u(x)) \leq -\alpha$ vérifiée à chaque test.

Nous présentons ici quelques éléments de preuves. Si pour un point x de la frontière δr, on a : $N(x,u(x)) \leq -\alpha$, ce qui veut dire que :

$$\begin{aligned} \epsilon &= (-N(x,u) - \eta)/K \\ &\geq (\alpha - \eta)/K \end{aligned}$$

Ainsi, l'algorithme retourne une cellule avec un rayon supérieur ou égal à $(\alpha - \eta)/K$. La surface étant finie, on est sûr que la frontière serait couverte par un nombre fini de cellules générées par l'algorithme.

4.4 Génération d'une commande stabilisatrice dans le cas d'un système continu

Nous nous basons sur les propriétés définies dans la section précédente afin de présenter un algorithme de génération automatique de commande stabilisatrice. En pratique, notre la génération est réalisée de façon similaire à la vérification.

4.4.1 Algorithme de génération

L'idée centrale de l'algorithme de génération de la commande est de parcourir la surface séparant r de $Z-r$. À chaque fois, une commande est choisie pour le point traité, et un rayon de la cellule centrée sur ce point est calculé.

Algorithme 6 *Données :*

- *Une région* $r = \{h(x) \leq 0, x \in Z\}$, ∇h *étant une fonction l-Lipchitzienne dont la norme est majorée par* N. *Appelons* δr *la frontière de* r ($\delta r = \{x, h(x) = 0\}$). *Soit* x_0 *un point de* δr.
- *Une fonction* $f : X \times U$ *k-lipchitzienne dont la norme est majorée par* M.
- *Un ensemble fini de fonctions de commande* $U = u_1,...,u_{|U|}$.
- *Un réel positif très petit* η.

Résultat : Un ensemble de cellules $C = \{c_1,...,c_{|C|}\}$. *Un ensemble de commandes associées aux cellules* $U_C = \{u_{c_1},...,u_{c_{|C|}}\}$. u_i *est la commande associée à la cellule* c_i.

Initialisation : $C = \emptyset$ *est l'ensemble de cellules générées,* u_C *est la fonction qui associe à chaque cellule* $c \in C$ *une fonction de commande* $u_C(c)$. *Soit* $S = \emptyset$ *correspondant à la surface couverte par l'algorithme à chaque itération.*
Répéter

1. *Parcourir* u *dans* U.
2. *Si* $N(x,u(x)) = \nabla h(x) \times f(x,u(x)) < -\eta$,
   - *Calculer* $\epsilon = (-N(x,u(x)) - \eta)/K$.
   - $C := C \cup \{c(x,\epsilon)\}$, $u_C(c(x,\epsilon)) = u$
   - $S = S \cup c(x,\epsilon)$
   - *Choisir un point* x *de la frontière de* S *et passer à l'itération suivante*

3. *Aucune commande ne vérifie* $N(x,u) = \nabla h(x) \times f(x,u(x)) < -\eta$ *: échec de l'algorithme*

Jusqu'à $\delta r \subseteq S$

L'algorithme 6 parcourt la surface δr. À chaque itération, une commande est associée au point étudié. Ceci permet d'étendre la surface S couverte par la commande de stabilisation. A la fin de l'itération un autre point est choisi de la frontière de S (si un point est dans la frontière de S, il possède un voisinage qui n'est pas dans S). Les conditions de convergence en nombre fini d'étapes sont les mêmes que celles énoncées pour l'algorithme de vérification (L'algorithme 5).

4.4.2 Génération couplée avec la réduction

En pratique, il est souvent difficile de stabiliser un point de fonctionnement dans une région r de la façon décrite dans le paragraphe 4.4. La cause en est que l'inversion d'évolution d'un point de fonctionnement dans l'espace continu peut être impossible dans le cas réel. Trouver une commande stabilisatrice sur la frontière de la région à stabiliser peut être impossible. Notre solution est de limiter la région à stabiliser. En pratique, une telle réduction permet d'éliminer les dynamiques trop *rapides* pouvant causer un fonctionnement trop rapide.

Ainsi, la problématique de stabilisation du point de fonctionnement dans r se réduit à:

- Choisir $r' \subset r$
- trouver la commande stabilisatrice dans r'.

Le choix de la région r' peut être réalisé de diverses façons. Il peut être choisi par l'expert en enlevant par exemple la possibilité de grandes valeurs

pour les diverses dérivées. Cependant, afin de conserver le caractère automatique de la procédure, nous donnons une version générale automatisée avant de donner une application dans un cas particulier. Pour cela, à chaque fois qu'on constate sur un point de la région un échec, on essaie de trouver un autre point du voisinage intérieur permettant la stabilisation de la même façon tout en étant relié aux points déjà explorés.

L'algorithme global de génération couplé à la réduction est donné dans 7. Pour cela, une procédure spéciale de réduction est invoquée. Cette procédure peut être construite de plusieurs façons selon la nature du système. Voir paragraphe 4.4.2.1.

Algorithme 7 *Données :*

- *Une région* $r = \{h(x) \leq 0\}$, ∇h *étant une fonction l-Lipchitzienne dont la norme est majorée par* N. *Appelons* δr *la frontière de* r ($\delta r = \{x,h(x) = 0\}$).
- *Une fonction* $f : X \times U$ *k-lipchitzienne dont la norme est majorée par* M.
- *Un ensemble fini de fonctions de commande* $U = u_1,...,u_{|U|}$.
- *Un réel positif très petit* η.

Résultat : Un ensemble de cellules $C = \{c_1,...,c_{|C|}\}$. *Un ensemble de commandes associées aux cellules* $U_C = \{u_{c_1},...,u_{c_{|C|}}\}$. u_i *est la commande associée à la cellule* c_i.

Initialisation : $C = \emptyset$ *est l'ensemble de cellules générées,* u_C *est la fonction qui associe à chaque cellule* $c \in C$ *une fonction de commande* $u_C(c)$. *Soit* $S = \emptyset$ *correspondant à la surface couverte par l'algorithme à chaque itération. Soit* x *un point de* δr.

Répéter

1. *Parcourir* u *dans* U.

2. *Si* $N(x,u(x)) = \nabla h(x) \times f(x,u(x)) < -\eta$,
   - *Calculer* $\epsilon = (-N(x,u(x)) - \eta)/K$.
   - $C := C \cup \{c(x,\epsilon)\}$, $u_C(c(x,\epsilon)) = u$
   - $S = S \cup c(x,\epsilon)$
   - *Choisir un point* x *de la frontière de* S *et passer à l'itération suivante*
3. *Aucune commande ne vérifie* $N(x,u) = \nabla h(x) \times f(x,u(x)) < -\eta$*: r:=réduire(r). Recommencer l'algorithme.*

Jusqu'à $\delta r \subseteq S$

4.4.2.1 Réduction d'une région

La réduction d'une région à stabiliser peut être réalisée de diverses façons selon les caractéristiques propres au système analysé. Le choix d'une méthode de réduction est très important afin de garantir le succès dans un temps raisonnable de l'algorithme de stabilsation couplée avec une réduction (Algorithme 7).

Prenons par exemple un système avec une région à stabiliser qui est un *hyperrectangle*, c'est à dire un produit cartésien d'intervalles. Dans ce cas, une procédure possible est de réduire l'intervalle correspondant à la cellule ou s'est produit l'échec. Voir la Figure 4.2.

4.4.3 Exemple

4.4.3.1 Le système des deux réservoirs

Dans cette partie, nous appliquerons les méthodologies développées dans la section 4.4 et 4.4.2.

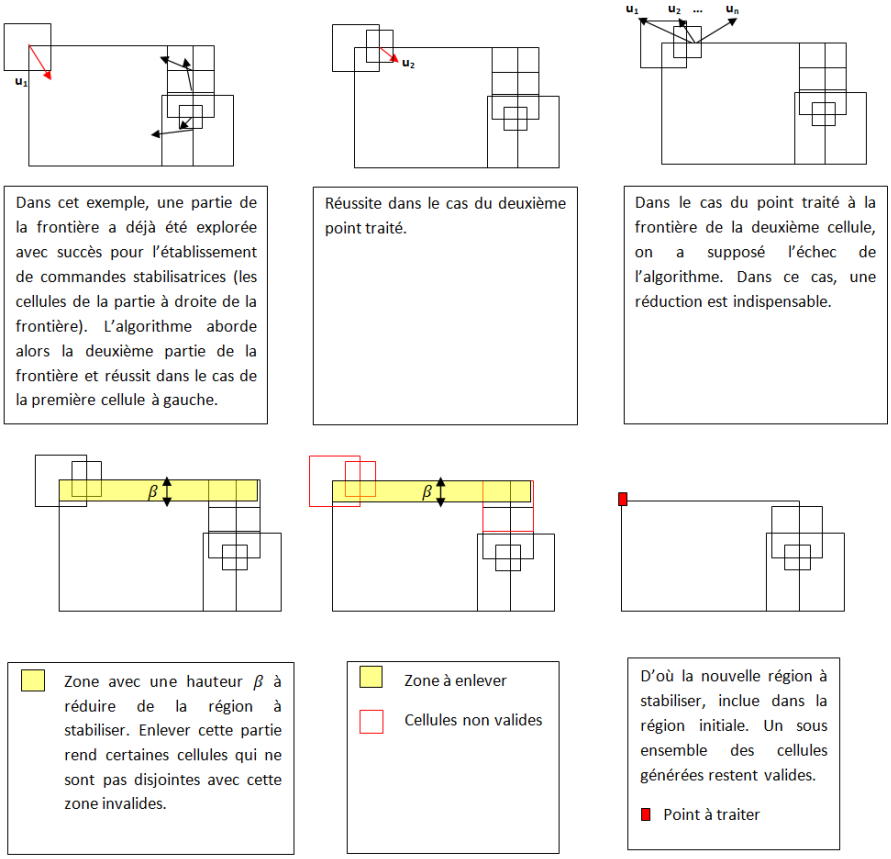

FIG. 4.2 – *Un algorithme pour la réduction d'une région*

Soit l'exemple du système à deux réservoirs de la Figure 2.1 introduit dans le chapitre 2.

Avec $a_{min} = 0$ et $a_{max} = 1$.

Supposons que l'objectif de stabilité soit d'évoluer vers la région $0.3 \leq h_1 \leq 0.6$, $-2 * 10^{-2} \leq I_P \leq 2 * 10^{-2}$,$0.09 \leq h_2 \leq 0.11$ lorsque la vanne $V1$ est ouverte et $V2$ est fermée. Imposer ces conditions a pour objectif de stabiliser le fonctionnement du système dans un intervalle bien déterminé tout en limitant la *vitesse* de changement du point de fonctionnement.

Dans le cas ou la vanne $V1$ est ouverte et $V2$ est fermée, on pose:

$$x = \begin{pmatrix} I_P \\ h_1 \\ h_2 \end{pmatrix}, u = (w_1), x' = \begin{pmatrix} I'_P \\ h'_1 \\ h'_2 \end{pmatrix}$$

on a alors

$$f(x,u) = \begin{pmatrix} I'_P(t) \\ h'_1(t) \\ h'_2(t) \end{pmatrix} = \begin{pmatrix} K_I(w_1(t) - h_1(t)) \\ \frac{K_P(w_1(t)-h_1(t))+I_P-a_{max}S\sqrt{2g(h_1-0.3)}}{A} \\ -a_Z S\sqrt{2gh_2}/A + a_{max}S\sqrt{2g(h_1-0.3)}/A \end{pmatrix} \tag{4.2}$$

Le système à deux réservoirs est commandé à l'aide d'une commande continue w_1, et des deux événements de commande $V1+, V1-, V2+, V2-$. Supposons qu'on quantifie l'espace d'entrée continue en $U = \{0,0.1,0.2,0.3, 0.4,0.5,0.6\}$. Ce sont ces commandes continues qu'on va utiliser pour stabiliser le système grâce à l'algorithme 6.

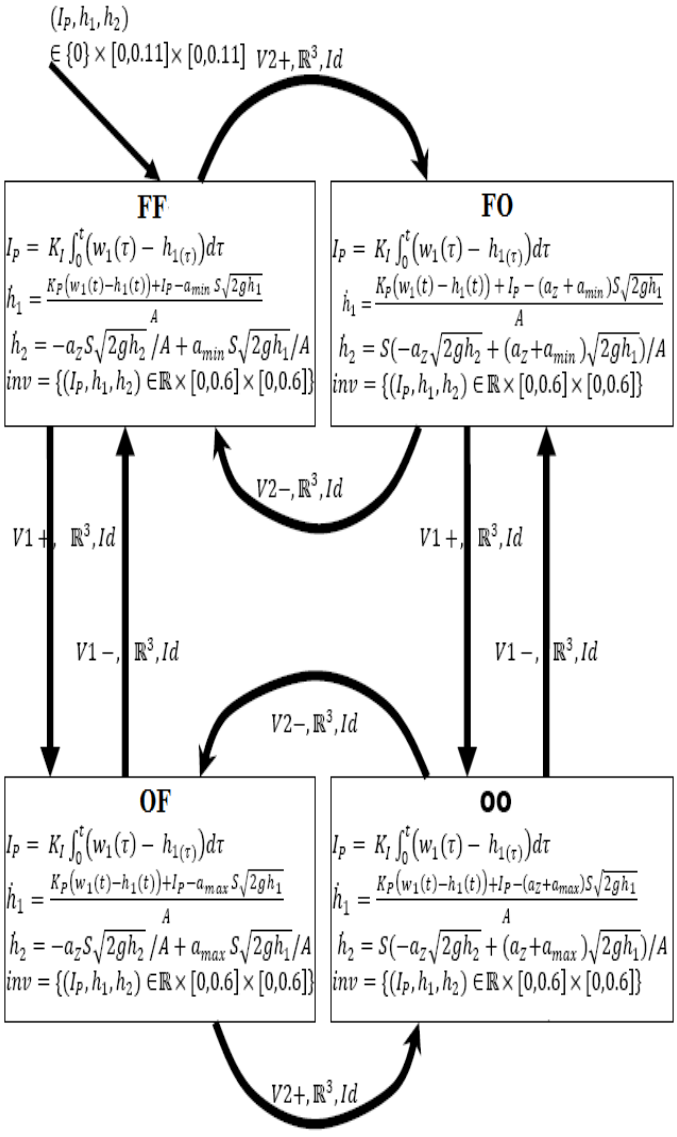

FIG. 4.3 – *Le modèle hybride du système à deux réservoirs*

4.4.3.2 Constante Lipchitzienne

Après calculs (voir annexe D) on a obtenu:

$$\begin{aligned} \|\nabla h\| &\leq N \\ \|f(x,u)\| &\leq M \\ \|\nabla h(x) - \nabla h(y)\| &\leq l\,\|x-y\| \end{aligned}$$

avec $N = 1, l = 1, M = 1$.

4.4.3.3 Application à la stabilisation

Supposons que notre objectif soit de stabiliser le système dans la région $r = \{-2 * 10^{-2} \leq Ip \leq 2 * 10^{-2}, 0.4 \leq h1 \leq 0.6, 0.09 \leq h2 \leq 0.11\}$. Les commandes disponibles sont $U = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$. Soit la fonction h définie par

$$h(h1,h2,Ip) = s(h1,h2,h3)min(dist(Ip,10^{-2}),dist(Ip,-10^{-2}),dist(h1,0.6),$$
$$dist(h1,0.4),dist(h2,0.09),dist(h2,0.11))$$

avec

$$s(Ip,h1,h2) = \begin{cases} -1, \text{ si } (Ip,h1,h2) \in r \\ 1, \text{ sinon} \end{cases}$$

La fonction h a été choisie de telle façon que la norme de son gradient soit égale à 1 sur la frontière.

$\nabla h(x)$ retourne toujours 1 ou -1 dans chaque composante du vecteur résultant (car la fonction h est toujours linéaire de coefficient de linéarité 1).

Soit $N(x,u) = \nabla h(x) \times f(x,u(x))$. Vouloir imposer $N(x,u) < -\eta$ revient à vouloir imposer :

$$h1' > \eta, \text{si } h1 = 0.4$$

$$h1' < -\eta, \text{si } h1 = 0.6$$

$$h2' > \eta, \text{si } h2 = 0.09$$

$$h2' < -\eta, \text{si } h2 = 0.11$$

$$Ip' > \eta, \text{si } Ip = -0.02$$

$$Ip' < -\eta, \text{si } Ip = 0.02$$

Prenons le cas de la frontière $h2 = 0.09$. En appliquant l'algorithme, on obtient une réduction vers la région $r_{red} = 0.3901 \leq h1 \leq 0.4099$, $-1.62 \cdot 10^{-4} \leq Ip \leq 4.62 \cdot 10^{-4}$,$0.09 \leq h2 \leq 0.11$. Ainsi, on dispose de la commande stabilisatrice du système à deux réservoirs tant que le point de fonctionnement est dans r_{red}.

Supposons que le point de fonctionnement du système soit dans r_{red}. Selon les commandes générées pour les cellules de la frontière, une commande stabilisatrice est:

$$u = \begin{cases} u <= 0.4 \text{ lorsque } Ip = 4.62 \cdot 10^{-4} \\ u >= 0.4 \text{ lorsque } Ip = -1.62 \cdot 10^{-4} \\ u >= 0.4 \text{si } h1 = 0.3901 \\ u <= 0.4 < -\eta \text{si } h1 = 0.4099 \end{cases} \tag{4.3}$$

La commande 0.4 permet à coup sûr de stabiliser le système dans la région spécifiée ci dessus. Notons que l'évolution de h2 ne dépend pas directement de la commande u. Elle dépend plutôt des niveaux $h1$ et Ip.

4.5 Génération et vérification de stabilité dans le cas hybride

Dans le cas d'un automate hybride $A = (Q,X,\Sigma,inv,A,f,U,E,q_0,x_0)$ selon la Définition 15, l'espace d'état est $Q \times X$.

Dans ce cas une région peut être vue en tant qu'ensemble $\cup\{(q_i,r_i)\}$, q_i étant un élément de Q, r_i est la région continue de X associée au mode discret q_i. Dans ce cas, la stabilisation dans cette région nécessite une commande permettant :

- d'interdire toutes les transitions autonomes qui aboutiraient à un point de fonctionnement en dehors de r. Ceci peut être fait en éliminant toute possibilité d'évolution vers les gardes des transitions autonomes concernées.
- de stabiliser le fonctionnement continu dans r_i selon la fonction $q_i.f$.

Cette tâche peut être réalisée en exploitant l'algorithme de vérification dans le cas continu (algorithme 5 de ce chapitre) et celui utilisé pour l'approximation de la mise à jour discrète dans (section 3.5 du chapitre 3).

Afin de pouvoir interdire les transitions autonomes illégales, il est nécessaire de réduire la région r à stabiliser en une région r' telle que toutes les transitions autonomes de r' ne provoquent pas une violation de r'. Lorsque cette condition est vérifiée, on a la garantie qu'aucune transition autonome ne risque de provoquer un saut autonome vers une région qui risque de ne pas être stabilisable ou d'être en dehors de r.

Soit l'algorithme de génération de stabilité dans le cas hybride:

Algorithme 8 *Données :*

- *Un système hybride* $A = (Q,X,\Sigma,inv,A,f,U,E,q_0,x_0)$.

- *Une région* $r = \cup\{(q_i,r_i)\}$, ∇h *étant une fonction l-Lipchitzienne dont la norme est majorée par* N.
- *Un ensemble fini de fonctions continues de commande* $U = u_1,...,u_{|U|}$.
- *Un réel positif très petit* η.
- *Une valeur de tolérance maximale* ϵ

Résultat :

- *Une région réduite* $r' = \cup\{(q_i,r'_i)\}$
- *Un découpage* C_i *pour chaque frontière* $\delta r'_i$ *correspondant à une région* r'_i.
- *Une fonction de commande* $u_i : C_i \rightarrow U$ *qui associe à chaque cellule une loi de commande dans* U.

Initialisation : $C_i = \emptyset$ *est l'ensemble de cellules générées dans le cas de* r'_i, u_i *est la fonction qui associe à chaque cellule* $c \in C$ *une fonction de commande* $u_C(c)$. *Soit* $S = \emptyset$ *correspondant à la surface couverte par l'algorithme à chaque itération.*

Début

1. *Réduire La région* $r = \cup(q_i,r_i)$ *: Soit* $r' = r$
   (a) *Pour chaque transition* $e = (q,q',g,\sigma,reinit) \in A$, *si* $reinit(g) \subsetneq r'$, *réduire* r' *à* $r' := r' - reinit^{-1}(Q \times X - r')$. *Cette opération est réalisée grâce à l'algorithme 4 de la section 3.5 du chapitre 3.*
   (b) *Soit la nouvelle région obtenue* $r' = \cup\{(q_i,r'_i)\}$,$q_i \in Q$,$r'_i \subset X$
   (c) *Soit* $\delta r'_i$ *la frontière de* r'_i. *Réaliser la génération de commande stabilisatrice comme cela est détaillé dans l'algorithme 6. Cette génération peut être combinée à une réduction comme c'est le cas de l'algorithme 7.*
   (d) *S'il y a eu réduction, refaire l'étape 1.*

La vérification de stabilité découle naturellement de l'algorithme 8 en réalisant un test dans l'étape 2 au lieu d'une recherche de commande.

4.6 Conclusion

Dans ce chapitre nous avons exposé une méthode numérique pour la vérification et la génération de commande stabilisatrice dans une région précise de l'espace d'état dans le cas d'un système continu. Nous avons ensuite étendu cet algorithme dans le cas ou aucune commande stabilisatrice n'existe pour cette région. Nous avons donné une méthodologie générale pour traiter cette tâche et donné un cas particulier où cette tâche peut être réalisée de façon numérique et aisée. Enfin, nous avons étendu les algorithmes développés dans le cas d'un système hybride.

La méthodologie exposée dans ce travail a pour avantage de pouvoir être implémentée sur un calculateur. La vérification peut être résumée comme étant un parcours de la frontière de la région à stabiliser. Par conséquent, l'efficacité de cette méthode dépend de la taille et du nombre de variables internes du système.

Chapitre 5

Abstraction des Systèmes Complexes

5.1 Introduction

Dans le paragraphe 1.3.3.1 du chapitre 1, nous avons réalisé un exposé à propos de la modélisation événementielle des systèmes complexes à l'aide d'automates à états finis. Ce type de modélisation est souvent une abstraction de systèmes complexes réels. Par exemple, il est possible de modéliser une machine dans un système manufacturier selon trois états : repos, fonctionnement, panne. Il est clair que cette description est une interprétation du fonctionnement du système qui fait abstraction de toutes les dynamiques internes. Un tel modèle discret peut être beaucoup plus facile à exploiter qu'un modèle exhaustif de bas niveau (décrit par un système hybride selon les définitions 28 ou 15).

Les systèmes hybrides, continus ou événementiels peuvent tous être décrits en tant que systèmes à transitions. Ainsi, ce paradigme est adapté pour la conception et le test de méthodologies globales pour les systèmes dyna-

miques.

L'abstraction des systèmes hybrides en des systèmes discrets est une des techniques permettant l'analyse éventuellement automatique des systèmes complexes. L'enjeu de cette technique est de permettre la transformation d'un modèle de bas niveau complexe en un modèle de haut niveau plus maniable pour des taches globales d'analyse, commande, supervision... Cette étape est souvent nécessaire car les analyses réalisées pour les composants ne prennent pas en compte les objectifs de haut niveau et les couplages induits par le système global.

Initialement, la modélisation de bas niveau peut être fournie sous forme de systèmes continus (avec des équations algébro-différentielles), discrets (sous la forme d'automate à états finis ou réseau de Petri, voir [35] et [57]) ou hybride (par exemple : sous la forme du modèle général de Branicky [15], [16], ou l'automate hybride [32]). Cependant, ces modélisations peuvent être unifiées sous la forme d'un système à transitions (voir paragraphe 3.2.1 du chapitre 3). Par conséquent, on peut admettre que le modèle initial qu'on doit transformer lors de l'abstraction est sous la forme générale d'un système à transitions.

En théorie, le résultat de la tâche d'abstraction peut être vu en tant que système général à transitions. Cependant, les approches d'abstraction ont souvent pour objectif de retourner un système discret à états finis. Ceci est dû à plusieurs raisons : l'exécution des algorithmes d'abstraction étant limitée par la vitesse de calcul et la mémoire des ordinateurs utilisés, il est impossible de générer des nouveaux modèles à nombre d'états infinis. En plus, des algorithmes pour l'analyse et la commande automatique des systèmes discrets à états finis existent déjà dans la littérature scientifique (même si l'implémentation de tels algorithmes est limitée par l'explosion combinatoire

dans le cas des systèmes larges).

Souvent, le modèle initial du système permet de présenter de façon exhaustive l'évolution dynamique du système. Par conséquent, ces informations contiennent des détails qui ne sont pas nécessaires pour la tâche d'analyse ou de commande envisagée. Une bonne approche d'abstraction doit permettre de *filtrer* le modèle initial et de préserver les propriétés intéressantes pour la tâche voulue. Une façon d'envisager cette abstraction est la construction d'un modèle qui soit en *bisimulation* avec le système initial. Après l'exécution d'un algorithme d'abstraction, le modèle résultant peut être utilisé pour diverses tâches de haut niveau telles que la vérification. Voir par exemple [6], [4] et [2].

Un algorithme pour la génération d'un système à transitions qui soit en bisimulation avec le système initial a été déjà proposé (voir par exemple [31], [43], [44], et [45]). Étant la base de notre approche d'abstraction, cet algorithme sera introduit par la suite. Un inconvénient majeur de cette approche est que l'exécution peut boucler indéfiniment lorsque le système initial possède un nombre d'états infini. Pour remédier à cet inconvénient, des approches ont été proposées afin de coupler la génération d'une bisimulation avec la tâche envisagée. Citons par exemple la proposition de Chutinan pour la vérification dans [19] et [21]. Cette proposition permet de limiter la complexité de l'algorithme de génération initial, bien qu'il n'y ait pas de garantie que l'algorithme soit exécuté en un nombre fini d'étapes.

Dans ce chapitre, nous proposons une méthodologie de construction d'un automate à états finis qui soit une abstraction d'un système à transitions *contrôlé* selon un *contexte*. Les algorithmes d'abstraction proposés seront moins ambitieux, mais plus efficaces pour des situations précises. Nous proposons aussi des algorithmes de mise à jour en temps réel de la structure de

l'automate discret. Nous montrerons qu'il est possible d'adapter ces méthodes pour la construction d'architectures de commande tolérantes aux fautes pour les systèmes hybrides.

5.2 Préliminaires

Nous commençons d'abord par un exposé succinct de méthodologies d'abstraction qui inspirent notre proposition. Cette section fait suite aux concepts exposés dans les chapitres précédents (voir la section 3.2.1 dans le chapitre 3).

5.2.1 Les systèmes à transitions

Dans la section 3.2.1 du chapitre 3 nous avons introduit le concept des systèmes à transitions. Afin de modéliser la réactivité du système, nous avons donné la définition du système à transitions étiquetées (Définition 17).

5.2.2 Opérateurs d'atteignabilité

Dans les chapitres 3 et 4 nous avons défini différents approches et algorithmes pour :

- Approximer à ϵ près l'évolution d'un système complexe (Algorithmes 2, 4).
- Vérifier la stabilisation dans une région de l'espace de phase du point de fonctionnement (Algorithmes 5 et 8).
- Générer une commande stabilisatrice dans une région impliquant une réduction éventuelle de la région stabilisée.

Ces approches constituent les outils de base qu'on va utiliser pour proposer une approche pratique d'abstraction des systèmes complexes en systèmes avec un nombre d'états fini exploitable pour la supervision ou la reconfiguration.

5.2.3 Concept et Synthèse d'une bisimulation

Dans les travaux cités précédemment, l'objectif de l'abstraction est de construire un nouveau système conservant la totalité des informations voulues [29], [6], [4] et [2]. Dans ce cadre, la construction d'un système *en bisimulation* constitue la réponse théorique idéale pour la problématique d'abstraction générale. En effet, il permet de sauvegarder la totalité des informations relatives aux propriétés de l'évolution du système, sous réserve que l'algorithme de conversion converge dans un temps fini.

Soit une relation $\approx$ dans $Q \times Q$. Alors, on peut définir le système quotient $T/\approx = (Q/\approx, \Sigma, \rightarrow_{\approx}, Q_0/\approx)$ tel que pour $p,p' \in Q/\approx$, $p,\sigma \rightarrow_{\approx} p'$ si et seulement s'il existe des éléments $q \in p$ et $q' \in p'$ telle que $q,\sigma \rightarrow q'$. Les états du système quotient $T/\approx$ sont des états *abstraits*. Par contre, Les états du système initial T sont des états *concrets*.

Ce système quotient peut être d'une très grande utilité si on peut établir de façon déterministe l'évolution des propriétés à partir de celle du système quotient. Dans l'idéal, une approche d'abstraction devrait permettre la construction d'un système quotient telle que la relation R soit une *bisimulation* entre le modèle initial et le modèle abstrait. Soient les opérateurs exacts d'atteignabilité *pre* et *post* définis comme dans le chapitre 3.

De façon informelle, $T/\approx$ et T sont en bisimulation lorsque pour deux états abstraits p,p' et un événement de commande σ, deux cas se présentent :

– Aucun état concret de p ne permet d'atteindre p' sous la commande σ.

Formellement : $p \cap pre(p',\sigma) = \emptyset$. Une autre formulation : $p' \cap post(p,\sigma) = \emptyset$.

- Tous les états concrets de p permettent d'atteindre p' sous la commande σ. Formellement : $p \cap pre(p',\sigma) = p$.

Ainsi, la condition de bisimulation se traduit formellement par Eq. (5.1) :

$$\forall \tilde{q}_1,\tilde{q}_2 \in Q/\approx, \tilde{q}_1 \cap pre(\tilde{q}_2,\sigma) = \emptyset \text{ ou } \tilde{q}_1 \cap pre(\tilde{q}_2,\sigma) = \tilde{q}_1 \qquad (5.1)$$

Pour que le système abstrait discret puisse être exploitée pour la supervision, il est nécessaire qu'il conserve certaines régions lors du découpage de Q. Ces régions contiennent la région initiale Q_0 et l'ensemble de régions qui sont utilisées lors de la définition des spécifications (ou objectifs) de fonctionnement abstraits.

Un algorithme de génération du système quotient en bisimulation avec le système originel a été proposé dans la littérature en se basant sur la condition 5.1. Un résumé de cet algorithme et de sa démonstration peuvent être retrouvés dans [19]. L'algorithme de synthèse de bisimulation est donné dans l'algorithme 9.

Algorithme 9 *Données :*

- *Un système à transitions* $T = (Q,\Sigma, \rightarrow, Q_0)$
- *Une relation* $\approx_0$ *définissant le découpage initial. Ce découpage initial correspond aux régions qui interviennent lors de la définition des objectifs de fonctionnement et de l'état initial.*

Résultats : Relation de bisimulation $\approx \in Q \times Q$ *telle que le système quotient* $T/\approx = (Q/\approx, \Sigma, \rightarrow_{\approx}, Q_0/\approx)$ *soit en relation de bisimulation avec le système* T.

Initialisation : relation d'équivalence initiale $\approx$ *définie telle que* $\approx = \approx_0$.

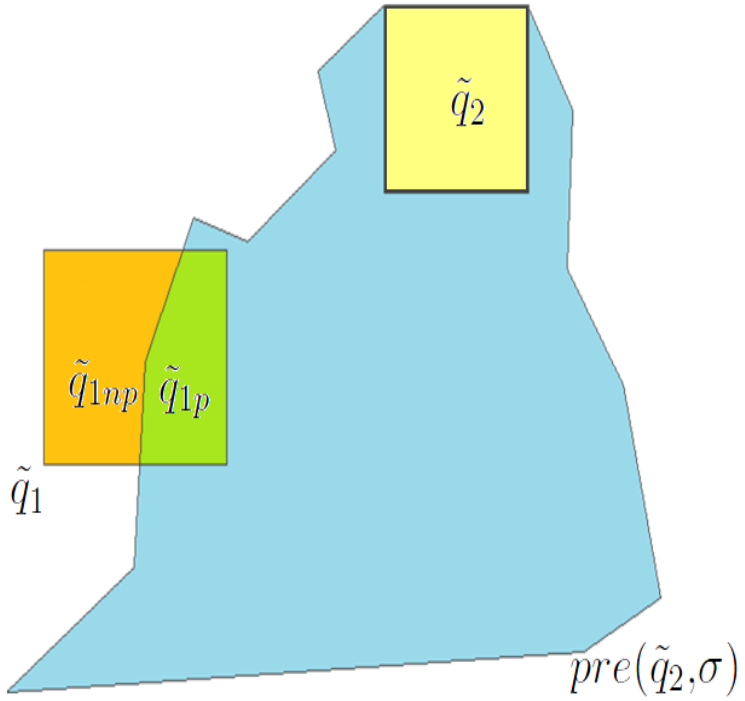

FIG. 5.1 – *Illustration du cas lorsque la condition de bisimulation n'est pas respectée. Il est alors nécessaire d'affiner le partitionnement en découpant* $\tilde{q}_1$

Début

Pour chaque $\tilde{q}_1,\tilde{q}_2 \in Q/\approx$, $\sigma \in \Sigma$

- *Si* $\emptyset \neq \tilde{q}_1 \cap pre(\tilde{q}_2,\sigma) \neq \tilde{q}_1$ *(cas illustré dans la Figure 5.1)*
  - *Diviser la région* $\tilde{q}_1$ *en* $\tilde{q}_{1p} = \tilde{q}_1 \cap pre(\tilde{q}_2,\sigma)$ *et* $\tilde{q}_{1np} = \tilde{q}_1 - pre(\tilde{q}_2,\sigma)$.
  - *Affiner le découpage en régions* $\approx$ *selon :* $P/\approx := (P/\approx - \tilde{q}_1) \cup \{\tilde{q}_{1p}\} \cup \{\tilde{q}_{1np}\}$.
- *Sinon : retourner la relation* $\approx$. *FIN.*

Dans le cas d'un système T avec un nombre d'états Q infini, il n'existe pas de garantie d'arrêt de l'algorithme 9. En plus, cet algorithme peut être qualifié de trop exhaustif : toutes les informations générées lors de l'abstraction ne

sont pas nécessaires pour la tâche d'analyse ou de commande à réaliser.

Par conséquent, des approches plus dédiées ont été proposées. Citons par exemple l'approche de Chutinan [19] pour la vérification intégrée à la génération du système quotient : à chaque itération, la propriété à vérifier est testée par rapport à l'état actuel du système généré. Dans certains cas, la validation de cette propriété au bout d'un nombre fini de boucles permet une exécution dans une durée finie, même lorsque l'algorithme 1 bouclerait à l'infini dans une exécution normale. Une approche analogue pour l'abstraction couplée à la vérification existe dans [2].

Dans la cadre de notre recherche, nous essayerons aussi d'adapter l'algorithme 9 pour l'abstraction selon un *contexte*, celui-ci étant le comportement dont l'observation ou la supervision est désirable.

5.3 Approche d'abstraction selon un contexte

Dans cette section, nous exposons notre méthodologie pour l'abstraction d'un système complexe modélisé en tant que système à transitions en un système à états finis. Cette approche qui se veut efficace, s'inspire de l'algorithme 9 tout en modifiant les conditions d'arrêt pour être moins restrictives. Le système à transitions initial peut être un système hybride (comme c'est le cas dans notre thèse) ou être un système continu ou discret.

5.3.1 La notion de contexte

Afin de pouvoir réaliser une abstraction qui conserve tous les aspects importants pour les tâches de haut niveau, nous devons spécifier quels sont les comportements qualitatifs qui doivent apparaître dans le modèle abstrait. Dans ce paragraphe, nous donnons une approche pour conserver ces diverses

informations grâce à une analyse intelligente du contexte de fonctionnement.

Les principales tâches que nous envisageons pour l'exploitation du système abstrait sont les suivantes :

- La vérification : il faut préserver la capacité de détecter si un ensemble d'états est atteignable ou non à partir d'un état origine et sous une certaine séquence de commandes.
- La commande et la supervision : il faut préserver la capacité de détecter et de commander certaines transitions qualitatives. En particulier, les évolutions utilisés pour la définition des objectifs de fonctionnement ou de la région de fonctionnement initiale.

On peut remarquer que dans les deux cas, un comportement du contexte peut être décrit en tant qu'évolution entre deux régions :

- La problématique de vérification peut être décrite comme étant la recherche, la vérification ou l'interdiction d'une évolution de r_0 vers r, ou r_0 est la région initiale et r est la région à atteindre ou à éviter.
- La problématique de commande et de supervision selon un langage régulier objectif B : B est un langage spécifiant une séquence d'événements observables. Dans ce cas, chaque événement correspond à un passage d'une région de l'espace de phase vers une autre. La problématique d'abstraction dans ce cas est la recherche d'un découpage de l'espace de phase qui puisse exhiber des comportements abstraits et déterministes inclus dans le langage B. Prenons le cas du système à deux réservoirs décrit dans la section 2.2 du chapitre 2. L'objectif de supervision est de générer une évolution de la région initiale $\{h2 \in [0{,}0.09[, h1 = 0, Ip = 0\}$ vers la région $\{h2 \in [0.09{,}0.11]\}$.

Pour résumer, nous envisageons une approche d'abstraction qui parte du même principe que l'algorithme 9. La différence réside dans l'objectif

d'abstraction. Dans notre approche, l'objectif est que le système abstrait contienne les comportements décrits dans le contexte d'abstraction. Afin de simplifier l'exposé, nous essayerons de coupler chaque définition formelle avec une description informelle.

Supposons que le système complexe originel soit modélisé sous la forme d'une système à transitions $T = (Q,\Sigma, \rightarrow ,Q_0)$. Soit R un ensemble de régions dans Q telle que $\forall r,r' \in R$, $r \cap r' = r$ ou $r \cap r' = r'$ ou $r \cap r' = \emptyset$. On dit que R est un découpage exact de l'espace d'états Q. Les ensembles vérifiant de telles conditions seront appelés des ensembles *exacts* (Un découpage exact n'assure pas forcément des régions disjointes). Dorénavant, $\{r\}$ désigne un élément correspondant à la région $r \subset Q$.

Pour illustrer cette notion, soit un ensemble $Q = q_1,q_2,q_3,q_4$. Soit les découpages $R1,R2,R3$ avec $R1 = \{R11 = \{q_1,q_2\},R12 = \{q_3,q_4\}\}$, $R2 = \{R21 = \{q_1,q_2\},R22 = \{q_3,q_4\},R23 = \{q3\}\}$ et $R3 = \{R31 = \{q_1,q_2\},R32 = \{q_3,q_4\},R33 = \{q1,q3\}\}$. Le découpage $R1$ assure des régions disjointes. Par contre, les régions de $R2$ ne sont pas disjointes bien que le découpage soit exact. Enfin, le découpage $R3$ n'est pas exact (car $R31 \cap R33 \notin \{\emptyset,R31,R33\}$).

Définissons maintenant la notion de contexte.

Definition 25 *Soit $T = (Q,\Sigma, \rightarrow ,Q_0)$ un système initial à transitions. Un contexte d'abstractions est un ensemble $C = \{c_i\}_i$, $c_i = (r_{srci},r_{desti},r_{Ii})$ étant un comportement du contexte telle que :*

r_{srci} : région source du comportement désiré.

r_{desti} : région destination du comportement désiré.

r_{Ii} : région interdite lors du comportement.

Interprétons maintenant la description du contexte. Chaque élément du contexte est un comportement qui doit apparaître dans le système abstrait.

En plus de la description des évolutions à vérifier, conserver ou interdire lors de l'abstraction, un contexte a la possibilité de décrire la problématique de stabilité conformément au chapitre 4. Supposons par exemple qu'on veuille stabiliser le fonctionnement dans une région r de l'espace d'état Q. Pour celà, il suffit d'inclure un comportement $c = \{r_{src} = r, r_{dest} = r, r_I = Q - r\}$.

Par la suite, chaque comportement $c = \{r_{src}, r_{dest}, r_I\}$ sera une description d'atteignabilité stricte ($r_{src} \cap r_{dest} = \emptyset$) ou de stabilité ($r_{src} = r_{dest}$).

5.3.2 Abstraction selon un contexte

Soit *post* et *pre* les opérateurs d'atteignabilité approximatifs pour le système à transitions T' (voir chapitre 3).

Definition 26 *Soit $T = (Q, \Sigma, \rightarrow, Q_0)$ un système initial à transitions. Soit un contexte d'abstraction $C = \{c_i\}_i$, $c_i = \{r_{srci}, r_{desti}, r_{Ii}\}$. Enfin, soit un deuxième système à transitions $T' = (P, \Sigma, \rightarrow', P_0)$, P étant un découpage exact de Q. Le système à transitions T' est une abstraction selon le contexte C de T si :*

- *Pour tout comportement $c = \{r_{src}, r_{dest}, r_I\}$, il existe $p, p' \in P$ et une fonction de commande sur r_{src} telle que :*
  - *$r_{src} \subset p, p' \subset r_{dest}$*
  - *l'évolution approximée à partir de p et des séquences de commande aboutit en temps fini à r_{dest} sans toucher r_I.*

Cette définition a pour avantage de définir une abstraction qui est moins exigeante que celle définie dans les autres travaux précédemment cités. Elle est cependant très puissante puisqu'elle peut décrire des exigences de commandabilité et de stabilité. Aussi, elle permet de spécifier la combinaison des deux exigences en une évolution stabilisée.

5.3.2.1 Approche initiale de génération d'abstraction

Notre méthodologie initiale de génération du modèle constituant une abstraction par rapport à un contexte s'appuie sur l'algorithme 9 tout en modifiant la condition d'arrêt et le choix des affinements pour qu'ils soient conformes aux comportements spécifiés dans le contexte.

Algorithme 10 *Données :*

- *Système à transitions initial* $T = (Q,\Sigma, \rightarrow ,Q_0)$.
- *Contexte d'abstraction* $C = \{c_i\}_i$. S
- *Un algorithme d'approximation d'atteignabilité selon* α *et* T. *Ces constantes sont choisies par les experts dans la phase hors ligne.*
- *Un découpage initial* R *de l'espace d'état* Q.

Résultat : Automate $T' = (P \subset 2^Q,\Sigma, \rightarrow' ,P_0 \subset P)$ *qui est une abstraction selon* C *de* T.

Initialisation : Automate $T_{aux} = (\hat{R},\Sigma, \rightarrow' ,r_0)$ *avec* $\rightarrow'$ *étant relation initialisée vide,* $\hat{R} = R$.

Début

Pour chaque comportement $c = (r_{src},r_{dest},r_I) \in C$ *telle que* $r_{src} = r_{dest}$ *(comportement de stabilisation), répéter :*

- *Stabiliser la région* r_{src} *selon l'algorithme 8 du chapitre 4 :*
  - *Trouver un découpage de* $r_{src} = \cup r_i$ *avec une commande stabilisatrice* σ_i *pour chaque* r_i *:* $Post(r_i,\sigma_i) \subset r$.
  - *Si un tel découpage n'existe pas : Echec de l'algorithme.*
  - *Si la stabilisation est réussie : intégrer cette stabilisation dans l'automate abstrait.*

Pour chaque comportement $c = (r_{src},r_{dest},r_I) \in C$ *telle que* $r_{src} \cap r_{dest} = \emptyset$

et $r_I = Q - r_{src}$, *répéter :*

- *Soit att la région atteignable à partir de* r_{src} *sans passer par* r_I. *Initialement* $att = r_{src}$
- *Calculer les régions accessibles à partir de* r_{src} *et d'une commande* σ . $att := att \cup post(att,\sigma)$
- *Á chaque fois qu'on constate que* $att \cap r_{dest} \neq \emptyset$, *on découpe la région d'origine* r_{src} *en* r_{src1} *permettant d'atteindre* r_{dest} *sans passer par* r_I *et* $r_{src2} = r_{src} - r_{src1}$.
- *Si* $r_{src2} \neq \emptyset$, *remplacer* $c = (r_{src},r_{dest},r_I)$ *par* $c = (r_{src2},r_{dest},r_I)$.
- *Si* $r_{src2} = \emptyset$, $C := C - c$.
- *On met à jour l'automate abstrait en y intégrant le comportement constaté. jusqu'à* $C = \emptyset$.

De façon informelle, l'algorithme 10 réalise pour chaque comportement du contexte une analyse d'atteignabilité en se basant sur les opérateurs d'atteignabilité approximative. Á chaque fois qu'une intersection est constatée entre la région atteignable et la région destination r_{dest}, l'automate abstrait est mis à jour afin d'intégrer ce comportement. La partie de r_{src} qui ne permet pas d'atteindre la destination est intégrée de nouveau au contexte.

5.3.2.2 Approche d'abstraction couplée à la réduction

Bien que beaucoup moins restrictif que l'algorithme 9, notre implémentation de l'algorithme 10 pour l'exemple à deux réservoirs cité 2.2 du chapitre 2 a échoué. En effet, nous n'avons pas pu trouver de commande stabilisatrice dans la région $\{h_2 \in [0.09,0.11]\}$. En analysant de plus près les causes de cet échec, il nous a paru évident qu'il était impossible de stabiliser le comportement dans la région $\{h2 \in [0.09,0.11]\}$. En effet, lorsque $h1 < 0.3$

et $h2 = 0.09$, aucun fluide ne peut circuler du réservoir $T1$ vers $T2$. Ceci implique qu'aucune commande ne pourra empêcher le niveau de fluide de descendre sous 0.09 à cet instant précis, c'est à dire de quitter la région à stabiliser.

Pour remédier à cette difficulté, l'algorithme 10 peut être modifié en incluant la possibilité de *réduction* lors de la recherche de commande stabilisatrice. Celle ci permet de réaliser une abstraction encore moins restrictive. La réduction est réalisée selon l'approche donnée dans la section 4.4.2 du chapitre 4.

D'ou l'algorithme suivant :

Algorithme 11 *Données :*

- *Système initial à transitions* $T = (Q,\Sigma, \rightarrow ,Q_0)$.
- *Contexte d'abstraction* $C = \{c_i\}_i$. S
- *Un algorithme d'approximation d'atteignabilité selon* α *et* T. *Celles-ci sont des constantes choisies par les experts dans la phase hors ligne.*

Résultat : Automate $T' = (P \subset 2^Q,\Sigma, \rightarrow' ,P_0)$ *qui est une abstraction couplée avec la réduction selon* C *de* T.

Initialisation : Automate $T_{aux} = (\hat{R},\Sigma, \rightarrow' ,r_0)$ *avec* $\rightarrow'$ *la relation initialisée vide,* $\hat{R} = R$.

Début

- *Pour chaque* $c = (r_{src},r_{dest},r_I) \in C$ *telle que* $r_{src} = r_{dest} = \emptyset$, *répéter :*
  - *Stabiliser et réduire la région* r_{src} *selon l'algorithme 8 du chapitre 4. Soit* r_{red} *la région réduite stabilisée résultat de l'exécution de l'algorithme.*
  - *Inclure le résultat de la réduction dans l'automate abstrait.*

- *Tous les comportements du contexte à abstraire* $c = (r'_{src}, r'_{dest}, r'_I)$ *telle que* $r'_{dest} = r_{src}$ *sont remplacés par* $c = (r'_{src}, r_{red}, r'_I)$.
- *Mettre à jour l'ensemble* $\hat{R}$ *en y ajoutant* r_{red}.

– *Exécuter l'algorithme 10.*

Toute information générée à partir des algorithmes d'abstraction peut être exploitée directement par d'autres processus concurrent à chaque étape de génération. Ceci est particulièrement intéressant pour les algorithmes de reconfiguration (6) qui doivent fonctionner au plus vite pour trouver des solutions afin de gérer l'occurrence des défaillances.

5.3.3 Mise à jour du modèle abstrait

Lors de l'occurrence d'une défaillance, le fonctionnement du système d'origine est altéré en incarnant des nouveaux comportements et des incertitudes. Ces modifications suggèrent que le système abstrait synthétisé par les algorithmes précédents peut ne plus être valide.

En pratique, l'occurrence de défaillances ne se traduit pas par des changements radicaux du fonctionnement. En réalité, ce sont seulement certains comportements de l'espace de phase qui sont affectés par les modifications et incertitudes selon la modélisation de la section 2.3.3 du chapitre 2. Par conséquent, une solution pratique est de détecter les régions de l'espace de phase ou le comportement dynamique aurait changé, d'éliminer les transitions générées sur ces régions de l'automate abstrait (ceci est d'autant plus aisé que les états de l'automate abstraits ne sont autres que des éléments de la partition réalisée sur l'espace d'état) et réexécuter l'algorithme 11 sur les comportements du contexte ou l'évolution de la région source a été modifiée.

Plus de détails sont disponibles dans le chapitre 6.

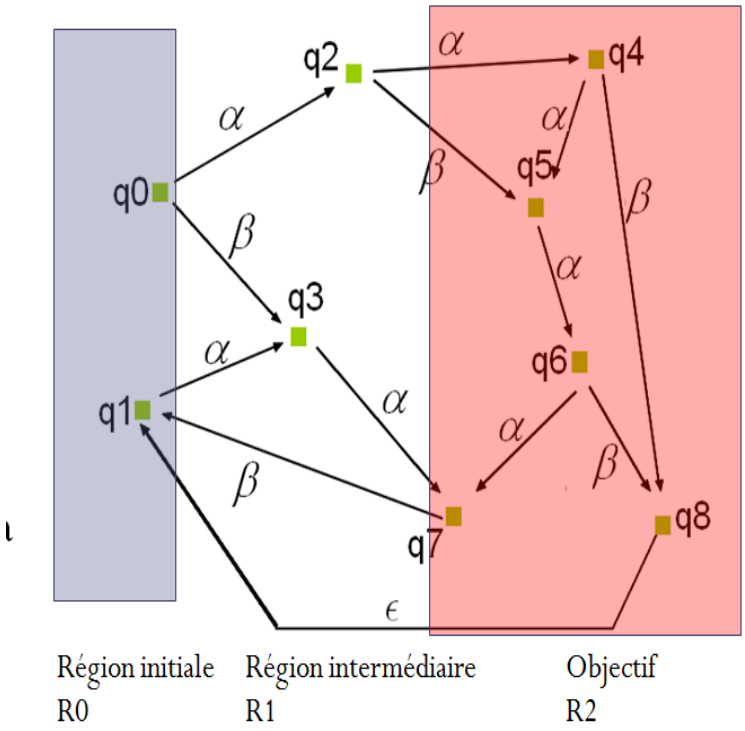

FIG. 5.2 – *Exemple de système à transitions à états finis*

5.4 Application de l'algorithme d'abstraction selon un contexte

5.4.1 Illustration sur un exemple discret

Soit le système discret de la Figure 5.2. Les événements non spécifiés ne changent pas le comportement du système. Ceci veut dire par exemple que $q5,\beta \rightarrow q5$.

Ce système est découpé en trois régions :

- $R0$: région initiale.
- $R1$: région intermédiaire.
- $R2$: région objectif.

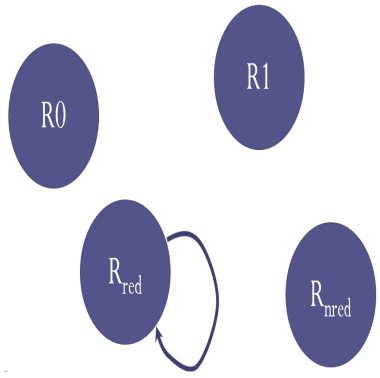

FIG. 5.3 – *Première étape de l'abstraction*

Nous voulons commander le système à partir de $R0$ vers $R2$, puis le stabiliser dans $R2$. Exécutons l'algorithme 11. Le contexte associé à cet objectif est :

$$c1 = (R0,R1,\emptyset) \text{ spécifie une évolution de R0 vers R1}$$
$$c2 = (R1,R2,\emptyset) \text{ spécifie une évolution de R1 vers R2}$$
$$c3 = (R2,R2,Q - R2) \text{ spécifie une stabilisation dans la région R2}$$

La première étape est une stabilisation et une réduction de la région $R2$: Seule une partie de $R2$ peut être stabilisée car pour $q8 \in R2$, $q8,\epsilon \rightarrow q1$. le résultat de la réduction est un découpage de $R2$ en $R_{red} = \{q4,q5,q6,q7\}$ avec $post(R_{red},\alpha) \subseteq R_{red}$ et $R_{nred} = \{q8\}$.

Le résultat est l'automate abstrait donné par la Figure 5.3.

Étant donnée qu'on a réalisé une réduction, une modification du contexte s'impose : $C = \{c1,c2\}$ avec $c1 = (R0,R1,\emptyset)$ et $c2 = (R1,R_{red},\emptyset)$.

La deuxième étape traite la problématique de génération du comportement $c1 = (R0,R1,\emptyset)$. Ceci est résolu en observant que $post(R0,\alpha) = R1$. Le résultat est l'automate abstrait mis à jour de la Figure 5.4.

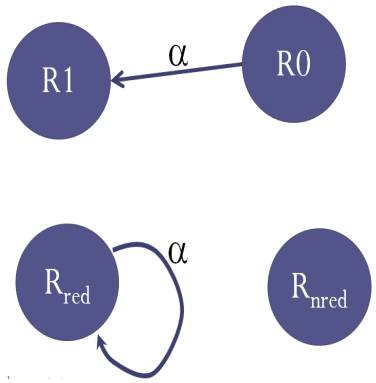

FIG. 5.4 – *Deuxième étape de l'abstraction*

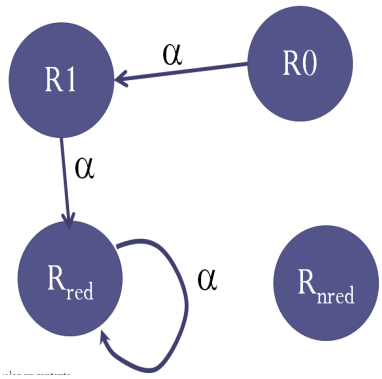

FIG. 5.5 – *Troisième étape de l'abstraction*

L'algorithme d'abstraction finit en traitant le dernier comportement $c2 = (R1,R_{red},\emptyset)$. On a $post(R1,\alpha) = R_{red}$. Ceci nous permet d'avoir l'automate de la Figure 5.5.

Tous les comportements du contexte ont été traités par l'algorithme. Ainsi, on dispose d'une abstraction permettant de réaliser la tâche de supervision.

5.4.2 Application sur l'exemple des deux réservoirs

5.4.2.1 Génération d'un modèle discret abstrait

La problématique de supervision des deux réservoirs a été décrite dans la section 2.2 du chapitre 2. Elle a été utilisée pour illustrer la vérification et la génération de stabilité. Dans cette partie, nous montrerons que l'algorithme 11 permet de combiner toutes ces techniques pour générer un système abstrait exploitable pour la supervision et la reconfiguration.

Supposons que l'état initial du système est caractérisé par $I_P = 0,h1 <= 0.3,h2 <= 0.09$ et les deux vannes fermées. Notre objectif est de faire évoluer le niveau de fluide dans $T2$ jusqu'à l'intervalle [0.09,0.11] et le stabiliser dedans. Le découpage initial est ainsi : $R_0 = \{L = \{h_2 \in [0,0.09[\},M = \{h_2 \in [0.09,0.11]\}$, et $H = \{h_2 \in]0.11,0.6]\}\}$. Q est l'ensemble des états discrets de l'automate hybride. Appellons $Q \times X$ l'ensemble des états hybrides du système à deux réservoirs.

À partir des objectifs on peut définir un contexte d'abstraction $C = \{c1,c2\}$ avec :

- $c1 = (Q \times L,Q \times M,Q \times X - L)$. C'est l'exigence d'une évolution de r_{src1} vers la région objectif.
- $c2 = (Q \times M,Q \times M,Q \times X - M)$. C'est l'exigence d'une stabilisation dans la région moyenne du réservoir $T2$.

Nous rappelons l'automate hybride dans la Figure 5.6.

Les commandes que nous allons utiliser sont : $V1+$ (ouverture de la vanne $V1$), $V1-$ (Fermeture de la vanne $V1$), $V2+$ (Ouverture de la vanne $V2$), $V2-$ (Fermeture de la vanne $V2$) et les commandes continues de régulation de la pompe $w_1 = 0$, $w_1 = 0.1$, $w_1 = 0.2$, $w_1 = 0.3$, $w_1 = 0.4$, $w_1 = 0.5$ et $w_1 = 0.6$.

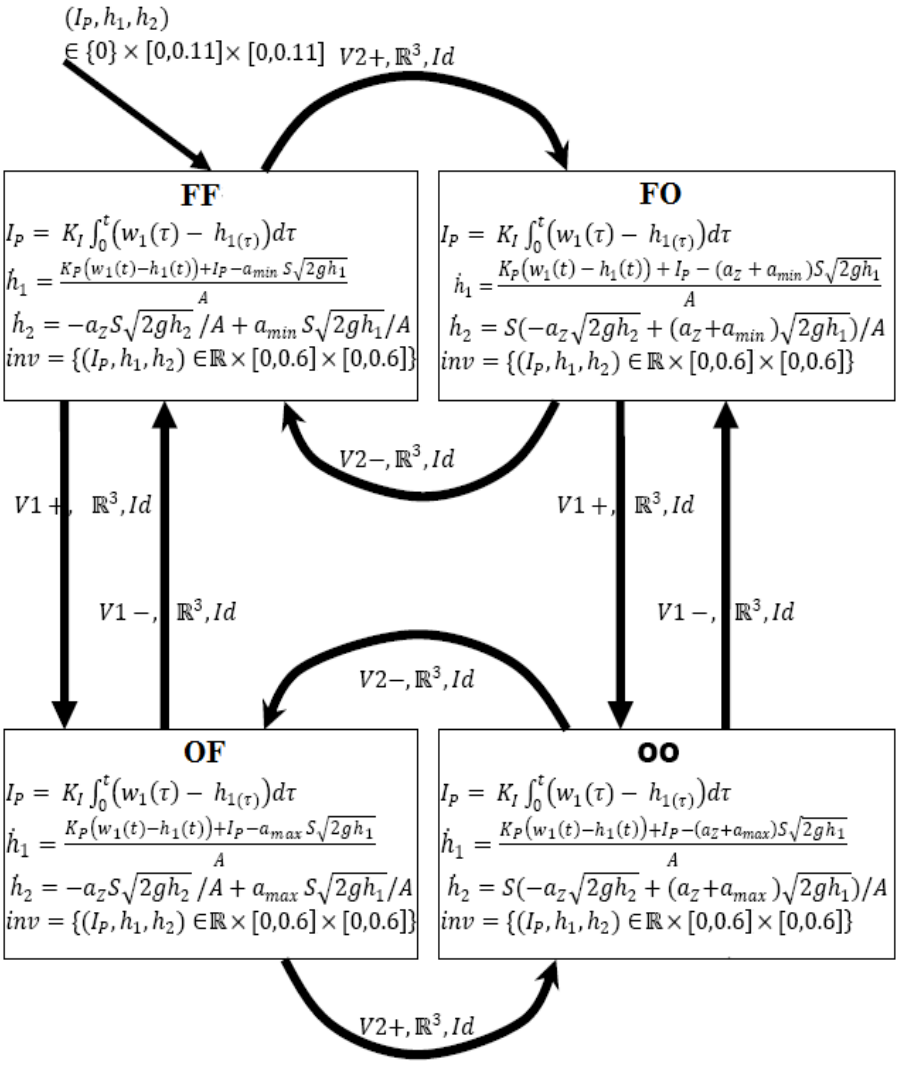

FIG. 5.6 – *L'automate hybride du comportement standard du système à deux réservoirs*

Appliquons l'algorithme 11.

1. Initialisation :
   - Système initial à transitions : le système hybride donné par la Figure 5.6.
   - Contexte C décrit précédemment.
   - Une constante d'approximation α.
   - Un horizon temporel de 10000 secondes.
   - État initial $I_P = 0,h1 <= 0.3,h2 <= 0.09$, $V1$ ouverte et $V2$ fermée.
   - La partition initiale est $\hat{R} = \{(FFL,L),Q \times L - (FFL,L),Q \times M,Q \times H\}$.
   - Automate $T_{aux} = (\hat{R},\Sigma, \rightarrow' ,r_0 = L)$.
2. Comportement $c2$: Région à stabiliser : $Q \times M$.
   - Les commandes $w1 = 0,0.1,0.2,0.3$ ne permettent pas d'assurer la propriété voulue. En effet, pour chacune de ces commandes, aucune région de M ne peut être stabilisée.
   - En utilisant $w1 = 0.4$, l'exécution de l'algorithme de réduction expliqué dans la section 4.4.2 du chapitre 4, on a $r_{red} = \{0.09 \leq h2 \leq 0.11,0.3901 \leq h1 \leq 0.4099, -1.62 \cdot 10^{-4} \leq Ip \leq 4.62 \cdot 10^{-4}\}$.
     - Mettre à jour T_{aux} : $\{r_{red}\},w1 \rightarrow \{r_{red}\}$.
     - Mettre à jour le contexte. Le nouveau contexte est : $C = \{c1\}$ avec $c1 = (r_{src1},r_{red},r_{I1})$ avec $r_{src1} = \{I_P = 0,h1 <= 0.3,h2 <= 0.09\}$, et r_{red} généré par l'algorithme de réduction. Enfin, $r_{I1} = \mathbb{R}^3$. C'est l'exigence d'une évolution de r_{src1} vers la nouvelle région stabilisée r_{red}.

- L'automate généré lors de cette étape est donné par la Figure 5.7.

- Comportement $c1 = (r_{src1} = \{I_P = 0, h1 <= 0.3, h2 <= 0.09\}, r_{red}, r_{I1})$.
  - Les commandes discrètes ne font pas évoluer l'état discret du système.
  - En utilisant la commande $w1 = 0.6$, on est certain d'atteindre $r(red)$ dans un délai maximal T_{max} (voir la Figure 5.10) pour une partie de r_{src} (voir un aperçu réduit dans la Figure 5.11). Appelons cette partie r_1.
  - Mettre à jour le contexte. Le nouveau contexte est : $C = \{c1\}$ avec $c1 = (r_{src1} - r_1, r_{red}, r_{I1})$ avec $r_{src1} = \{I_P = 0, h1 <= 0.3, h2 <= 0.09\}$, et r_{red} généré par l'algorithme de réduction. Enfin, $r_{I1} = \mathbb{R}^3$.
  - L'automate généré lors de cette étape est donné par la Figure 5.8.

3. Comportement $c1 = (r_{src1} - r_1, r_{red}, r_{I1})$.
4. La commande $u = 0.4$ permet d'atteindre r_1. or r_1 permet d'atteindre r_{red}. Ainsi, on peut atteindre r_{red} à partir de $r_{src1} - r_1$. Le résultat est l'automate abstrait dans la Figure 5.9.
5. succès de l'algorithme.

La procédure d'abstraction fournit un superviseur capable d'assurer les objectifs du système, même lorsque l'algorithme de génération n'est pas achevé. Ce résultat justifie l'arrêt de la procédure de génération et l'utilisation du superviseur choisi.

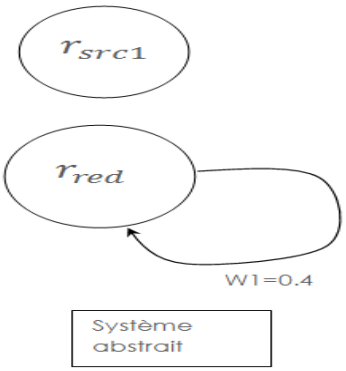

FIG. 5.7 – *Contexte et automate abstrait à l'issue de la première étape*

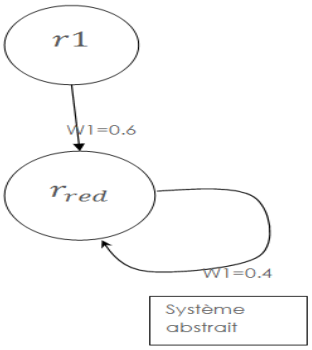

FIG. 5.8 – *Contexte et automate abstrait à l'issue de la deuxième étape*

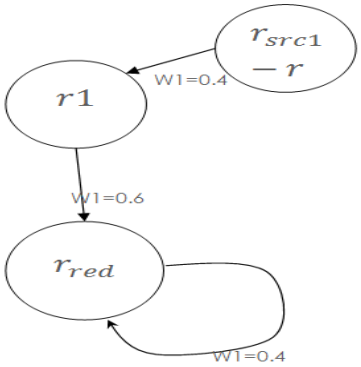

FIG. 5.9 – *Contexte et automate abstrait à l'issue de la troisième étape*

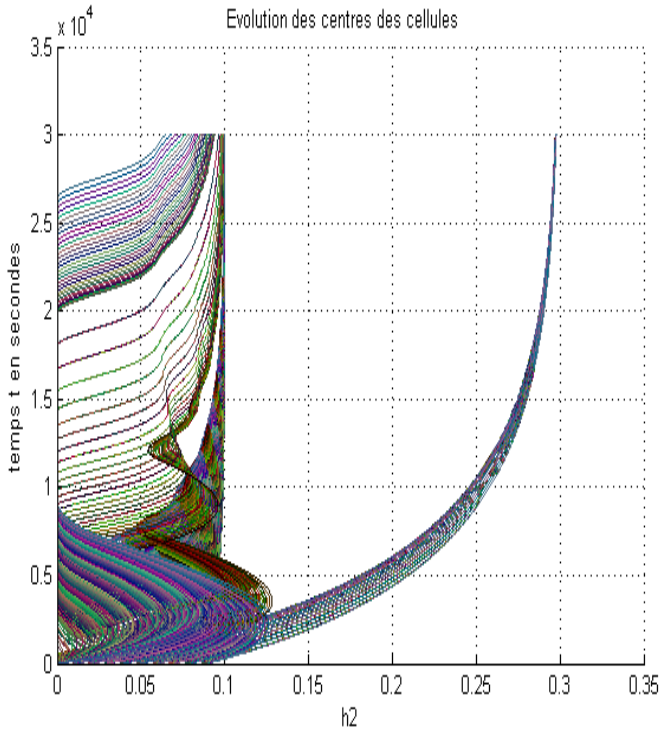

FIG. 5.10 – *Simulation de l'évolution à partir de la région inférieure du réservoir sous une commande* $w = 0.6$

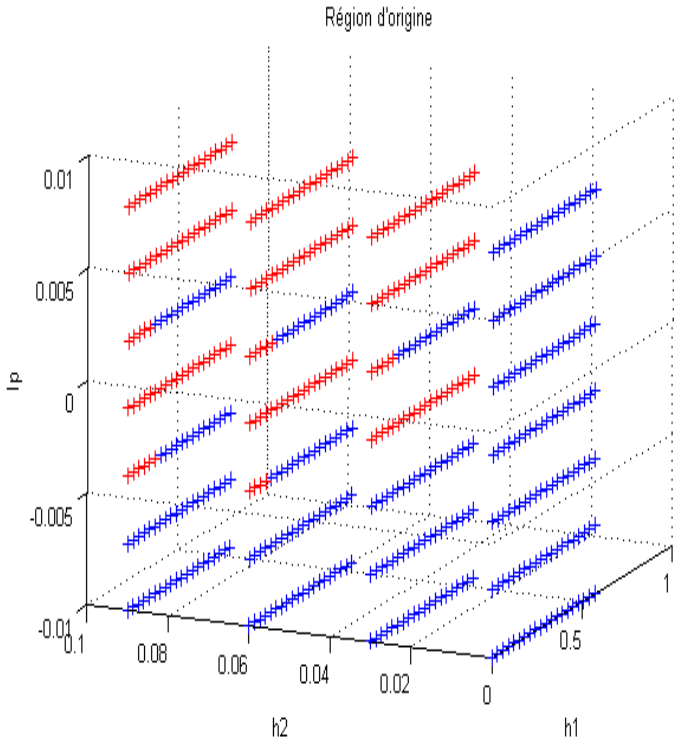

FIG. 5.11 – *Partie de* $r(c_1)$ *(en couleur bleue) permettant d'atteindre la région objective sous* $w = 0.6$

5.5 Conclusion

Notre méthodologie d'abstraction a pour avantage de réaliser une transformation d'un modèle initial selon un contexte vers un modèle réduit discret exploitable. Ceci permet la réalisation d'une transformation beaucoup plus rapide que celle observée lors de l'implémentation de la procédure de génération de l'espace quotient. L'algorithme de génération selon un contexte peut être couplé avec une procédure de haut niveau pour la vérification ou la commande. Il peut aussi être utilisé pour la mise à jour des modèles abstraits en cas de défaillance.

Chapitre 6

Approche Proposée pour la Reconfiguration en Temps Réel des Systèmes Complexes

6.1 Introduction

La reconfiguration des systèmes complexes joue un rôle capital dans le cas des systèmes critiques qui ne peuvent être supervisés facilement par des opérateurs humains. Cette difficulté peut découler de la complexité du système (tel que les centrales de production d'énergie, les systèmes de commande de vol...) ou de l'impossibilité d'assurer une communication fiable avec le processus (c'est les cas des réseaux de capteurs par exemple). Pour assurer l'autonomie de tels systèmes, les architectures de commande doivent être conçues en prenant en compte le risque de défaillances.

En général, on peut classifier les méthodologies de reconfiguration en passives/actives, incluant ou pas l'intervention humaine ou selon le paradigme de modélisation mis en œuvre (automates, équations algébro-différentielles...).

Un bref aperçu est donné dans le chapitre 1 destiné à l'état de l'art.

En pratique, l'approche la plus prisée dans l'industrie est l'utilisation de la redondance surtout matérielle. Elle est souvent couplée avec une analyse hors ligne permettant de prévoir les défaillances probables et les procédures de reconfiguration possibles. Cependant, cette méthode présente deux inconvénients majeurs : d'abord elle nécessite une analyse hors ligne très exhaustive. En plus, même les analyses et tests les plus poussés peuvent être incapables de prévoir certaines défaillances.

Dans ce chapitre, nous proposons une méthodologie pour la construction d'architectures de commande tolérantes aux fautes en temps réel dans le cas d'une modélisation en automates à états finis. À la suite d'une défaillance, le modèle du système est mis à jour selon les informations dont on dispose grâce aux modules d'identification et de diagnostic (voir chapitre 2). Ces modifications sont exploitées afin de mettre à jour le modèle discret abstrait du système complexe (voir chapitre 5). Les algorithmes proposés dans ce chapitre permettent de réduire le modèle abstrait dans des schémas prédéfinis hors ligne. Cette méthode sera illustrée dans le cas d'un système simple à deux réservoirs

6.2 Préliminaires

Dans la section 1.3.3 du chapitre 1, nous avons formulé la problématique de supervision et de reconfiguration dans le cas discret. Cette formulation prend en compte la possibilité de l'existence d'événements non observables et/ou non commandables. Cependant, étant donné que les transitions générées par les algorithmes d'abstraction sont toutes observables, et que les modules d'observation/identification/filtrage ne font pas partie de notre tra-

vail, nous nous limitons à la problématique *réduite* suivante:

Problématique 5 *La problématique de reconfiguration est de trouver à un instant $t' > t$ un langage commandable par rapport à M' et B_{deg} tel que l'état du système à t' soit reconnu et accepté par le superviseur.*

Les symboles B,B',M,M' sont définis de la même façon que dans la section 1.3.3 du chapitre 1.

6.2.1 Architecture globale

Dans la section 1.5 du chapitre 1, nous avons déjà donné un aperçu global de l'architecture proposée. L'architecture développée évolue selon le modèle abstrait (automate à états finis) dont on dispose du système complexe. La figure 1.7 du chapitre 1 résume cette architecture.

Notre approche de conception de l'architecture de commande haut niveau inclut deux phases importantes:

- Une phase hors ligne : elle inclut la construction d'un contrôleur *standard* pour commander le système complexe dans son état normal. Elle inclut aussi la construction de contrôleurs pour des objectifs dégradés pour différents schémas de fonctionnement prévus M_i.
- Une phase en ligne : elle contient un changement de la structure de commande selon l'état détecté du système, en s'appuyant éventuellement sur les schémas préconçus à l'étape hors ligne. Cette phase dépend fortement de la phase hors ligne, puisqu'elle est réalisée en réduisant le modèle défaillant M' vers un des schémas préconçus M_i. Si cette réduction n'est pas possible, une solution plus coûteuse en temps de calcul serait de refaire une synthèse de contrôleur dans le cas défaillant.

L'ensemble de ces étapes est décrit dans Fig. 6.1.

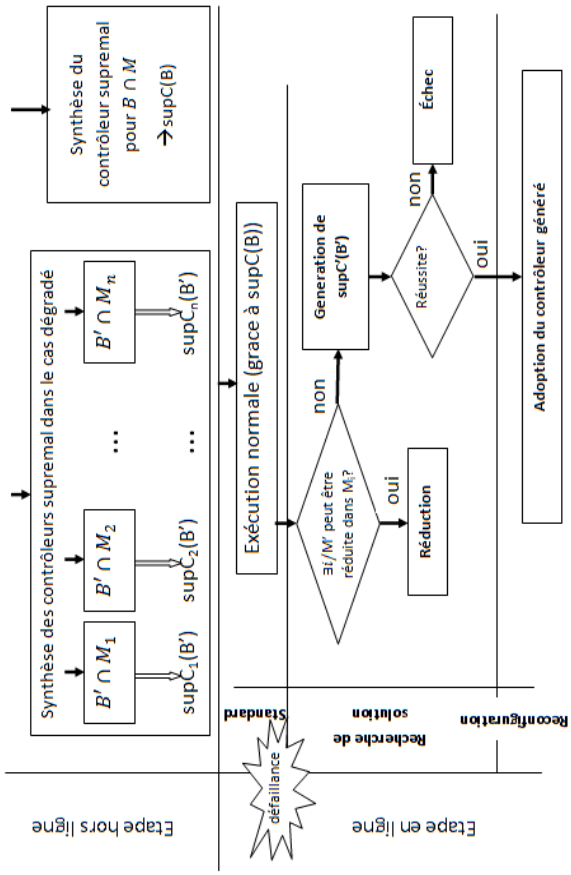

FIG. 6.1 – *Les différentes étapes de reconfiguration*

6.3 Reconfiguration par réduction

6.3.1 Défaillance et mise à jour du modèle abstrait

Dans la section 1.3.3.3 du chapitre 1, nous avons donné une formulation de la problématique de reconfiguration dans le cas discret. Cette formulation peut être adoptée et utilisée dans le cas du système abstrait généré ou mis à jour par les techniques du chapitre 5.

Une défaillance se traduit en pratique par le changement de comportement d'une partie ou de la totalité de l'espace de phase. Dans le niveau abstrait ceci peut se traduire par une incertitude à propos du comportement dans ces régions. Supposons que le modèle abstrait contienne $\{r\},\sigma \rightarrow \{r'\}$, $r,r' \in \hat{R}$ ($\hat{R}$ étant la partition générée par les algorithmes d'abstraction, voir le chapitre 5 à propos des notations). Si le comportement du système hybride dans r change (changement ou apparition d'incertitudes dans les paramètres continus ou discrets de défaillance, voir la section 2.3.3 du chapitre 2), la transition $\{r\},\sigma \rightarrow \{r'\}$ peut ne plus être valide. Par conséquent, la détection d'une défaillance se traduit par l'ajout de transitions partant de $\{r\}$ vers toutes les régions du voisinage de $\{r\}$ sous la commande σ.

Soit q un élément de $\hat{R}$. Nous appelons $V(q)$ l'ensemble d'éléments de Q qui sont voisins de q dans l'espace de phase concret.

Par la suite, les modules d'abstraction couplés à ceux du diagnostic réalisent une réduction des comportements possibles du système abstrait. Ceci veut dire que la mise à jour progressive est essentiellement une élimination de comportements superflus.

6.3.2 La phase hors ligne

Dans cette phase, plusieurs superviseurs sont générés:

- Le contrôleur standard : il s'agit de l'automate correspondant au sous langage maximal $supC(B)$ dans le cas des objectifs de fonctionnement B et du modèle M. Appelons A cet automate.
- Pour chaque schéma prédéfini M_i, on réalise une synthèse de l'automate correspondant au sous langage maximal dans le cas des objectifs dégradés de fonctionnement B' et du modèle M_i. Soit A_i l'automate du superviseur maximal correspondant au comportement M_i et les objectifs dégradés B'.

Dans le chapitre 1, nous avons introduit la modélisation discrète et les méthodologies de construction de superviseurs, notamment par la formule (6.1). Nous avons aussi montré qu'il était possible de construire l'automate A selon un algorithme avec un nombre fini d'étapes grâce à l'algorithme de construction de superviseur supremal (Algorithme 1), les détails étant disponibles dans la section 1.3.3.2 du chapitre1.

$$supC(B) = M \cap B - D_{uc}(M - B)\Sigma^* \tag{6.1}$$

D_{uc} étant l'opérateur qui associe à chaque mot $s \in \Sigma^*$ le plus grand préfixe de s dont le dernier symbole est commandable (section 1.3.3.2 du chapitre 1).

Soient $A(B)$ et $A(M)$ les automates correspondants aux langages B et M, avec $A(B)$ un automate complet. L'automate correspondant à $B \cap M$ est construit en tant que composition de B et M. Chaque état de $A(B \cap M)$ est une composition d'un état dans $A(B)$ et d'un autre dans $A(M)$. Ceci est valable aussi pour $A(M - B)$ car $M - B = M \cap \bar{B}$.

$A(D_{uc}(M-B))$ est construit en éliminant dans $A(M-B)$ toutes les séquences non commandables aboutissant à des états qui ne sont pas acceptés par B. En plus, $A(D_{uc}(M-B)\Sigma^*)$ est construit à partir de $A(D_{uc}(M-B))$ en ajoutant des transitions aux états finaux de $A(D_{uc}(M-B))$ vers un état puit.

Enfin, $A = A(supC(B)) = A(B \cap M - D_{uc}(M-B)\Sigma^*) = A(B \cap M \cap \overline{D_{uc}(M-B)\Sigma^*})$ est construit en rendant non acceptées dans $A(B \cap M)$ les états qui sont finales dans $A(D_{uc}(M-B)\Sigma^*)$ (Ceci est possible car $A(D_{uc}(M-B)\Sigma^*)$ a la même structure que $A(B \cap M)$).

6.3.2.1 Construction du superviseur standard pour le système à deux réservoirs

Au système hybride décrit dans la section 2.2 du chapitre 2, nous associons le comportement décrit par Fig. 6.2 lorsqu'on associe une entrée de commande continue constante $w1 = 0.4$, le système étant initialement vide ($h1 = h2 = 0$), et inoccupé ($Ip = 0$). La consigne initiale de commande est $w1 = 0$.

Chaque état (à part $q0$) est nommé par l'assemblage de trois lettres $l_1 l_2 l_3$ avec:

- l_1 état de la vanne $V1$: O pour vanne ouverte, F pour vanne fermée
- l_2 état de la vanne $V2$: O pour vanne ouverte, F pour vanne fermée
- l_3 état du réservoir $T2$: H désigne un état ou le niveau de fluide est supérieur à 0.11, L désigne un état ou le niveau de fluide est inférieur à 0.09, M est l'état ou le niveau de fluide est entre 0.09 et 0.11.

$Vi+$ et $Vi-$ sont les événements de commande pour l'ouverture et fermeture de la vanne Vi. Les niveaux de fluide sont détectés grâce aux événements:

- $LM-$: passage d'un niveau bas $h2 \in$ [0,0.09] vers un niveau moyen

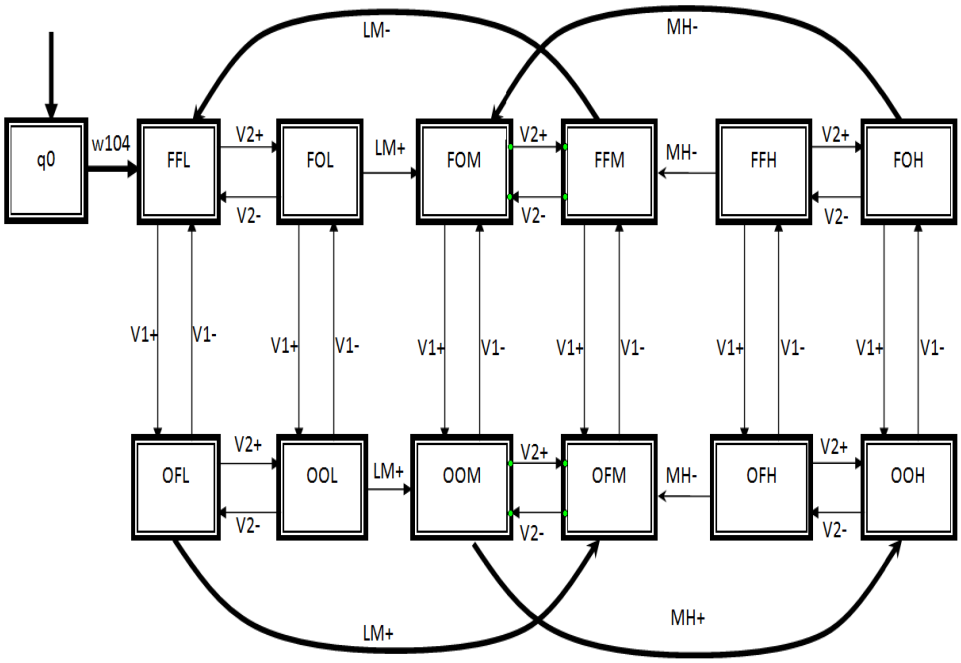

FIG. 6.2 – *Fonctionnement standard initial : $A(M)$*

$h2 \in [0.09,0.11]$.

- $LM+$: passage d'un niveau moyen $h2 \in [0.09,0.11]$ vers un niveau bas $h2 \in [0,0.09]$.
- $MH-$: passage d'un niveau moyen $h2 \in [0.09,0.11]$ vers un niveau haut $h2 \in [0.11,0.6]$.
- $MH+$: passage d'un niveau haut $h2 \in [0.11,0.6]$ vers un niveau moyen $h2 \in [0.09,0.11]$.

En plus de ces événements de commande discrets, nous utiliserons des événements de commande correspondant à des niveaux de commande continue pour le régulateur PI de la pompe.

- $w100$: associée à la commande $w1 = 0$.
- $w101$: associée à la commande $w1 = 0.1$.

- $w102$: associée à la commande $w1 = 0.2$.
- $w103$: associée à la commande $w1 = 0.3$.
- $w104$: associée à la commande $w1 = 0.4$.
- $w105$: associée à la commande $w1 = 0.5$.
- $w106$: associée à la commande $w1 = 0.6$.

Les objectifs de fonctionnement sont de faire évoluer le niveau de fluide dans $T2$ jusqu'à l'intervalle [0.09,0.11] et de le stabiliser dedans. Par conséquent, $A(B)$ peut être décrite par un automate selon Fig. 6.3.

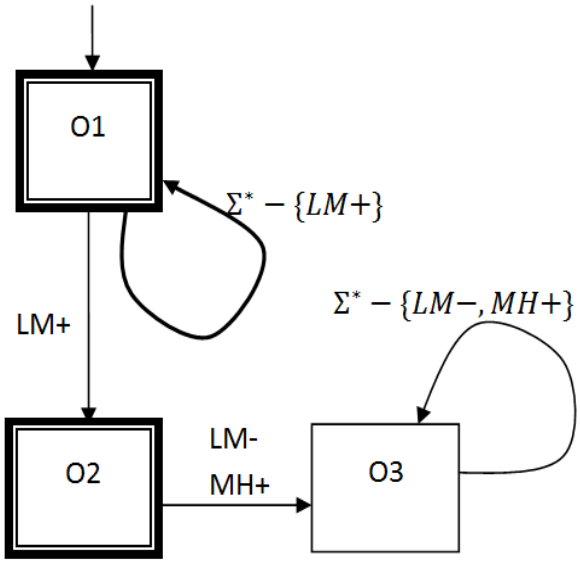

FIG. 6.3 – *Automate d'objectifs : $A(B)$*

Essayons maintenant d'exécuter l'algorithme 1:

- Réalisation de $A(B \cap M)$. Voir la Figure 6.4.
- Réalisation de $A(B - M)$. Voir la Figure 6.5.
- Réalisation de $A(D_{uc}(M - B)\Sigma^*)$. Voir la Figure 6.6.
- Réalisation de $A(M \cap B - D_{uc}(M - B)\Sigma^*)$. Voir la Figure 6.7.

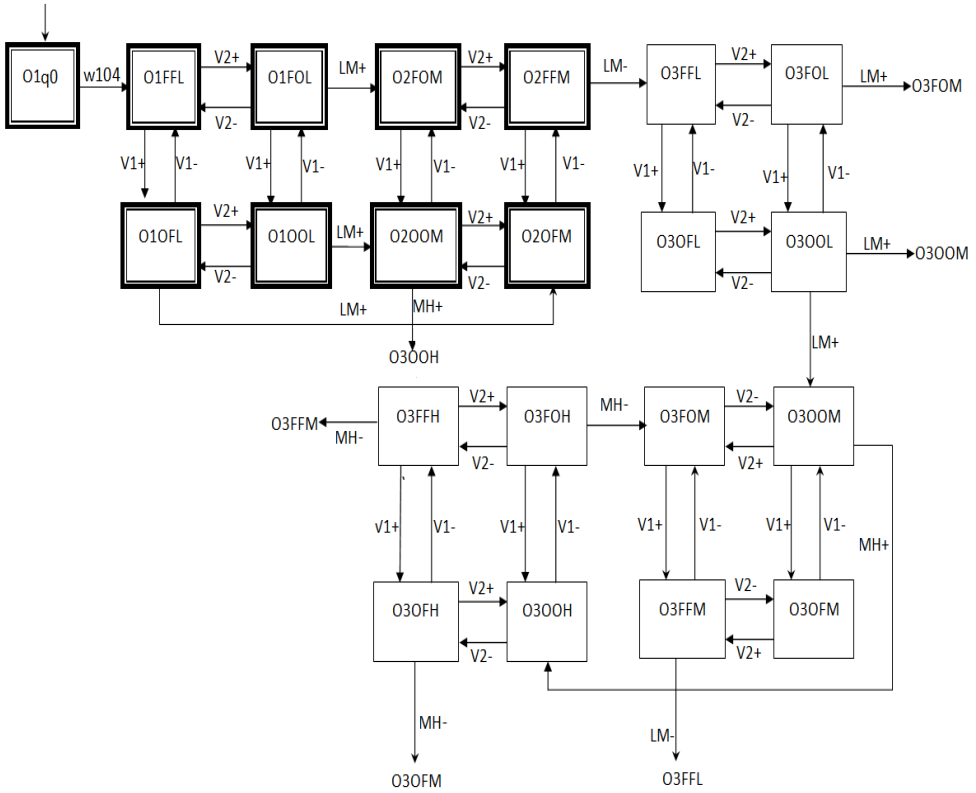

FIG. 6.4 – *Réalisation de* $A(B \cap M)$

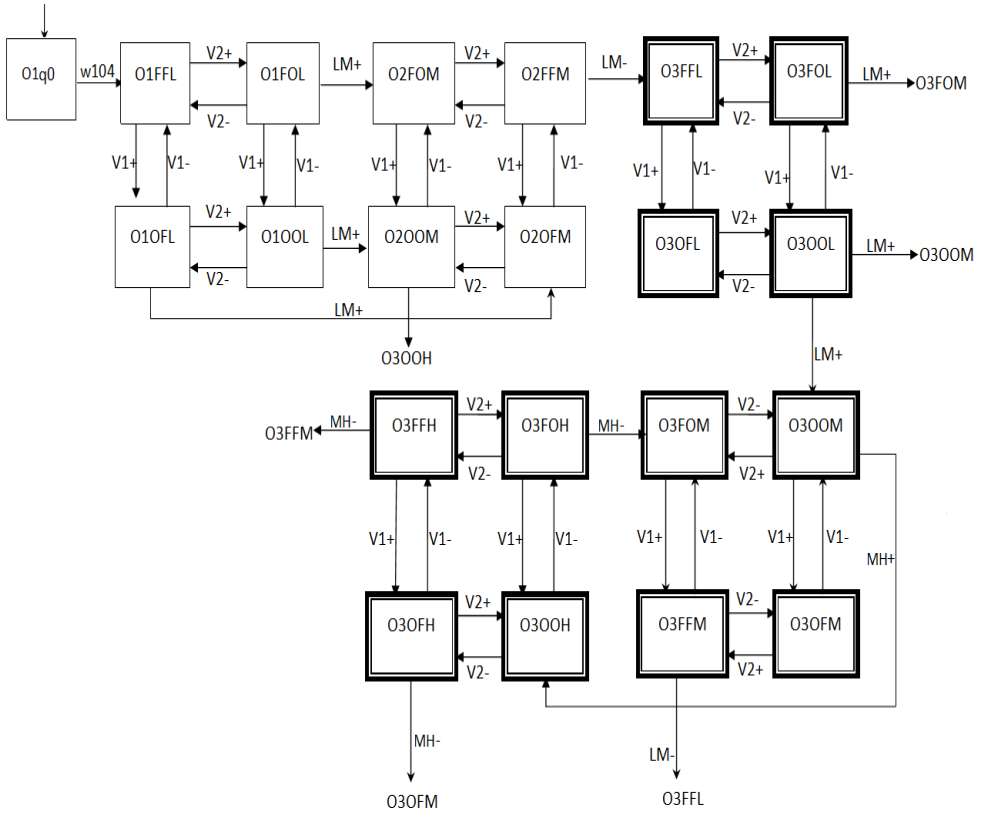

FIG. 6.5 – *Réalisation de* $A(B-M)$

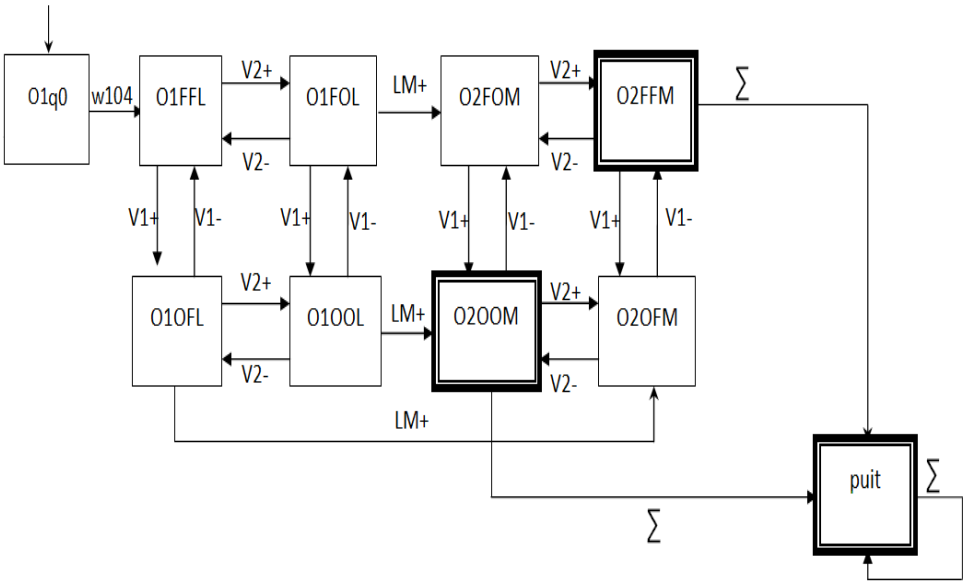

FIG. 6.6 – *Réalisation de* $A(D_{uc}(M-B)\Sigma^*)$

Le superviseur décrit par la Figure 6.7 constitue la solution de supervision dans le cas standard lorsqu'on dispose d'une commande constante $w1 = 0.4$.

Durant la phase hors ligne, nous disposons de deux schémas prédéfinis de modèles défaillants:

- Comportement normal (Figure 6.2).
- Apparition d'une fuite dans le réservoir $T1$ (Figure 6.8).

L'automate de supervision associé au schéma 6.8 est donné par la Figure 6.9.

6.3.3 La phase en ligne

6.3.3.1 Défaillance et détection:

Quand une défaillance apparaît, le comportement discret est mis à jour de façon progressive grâce aux algorithmes développés dans le chapitre 5.

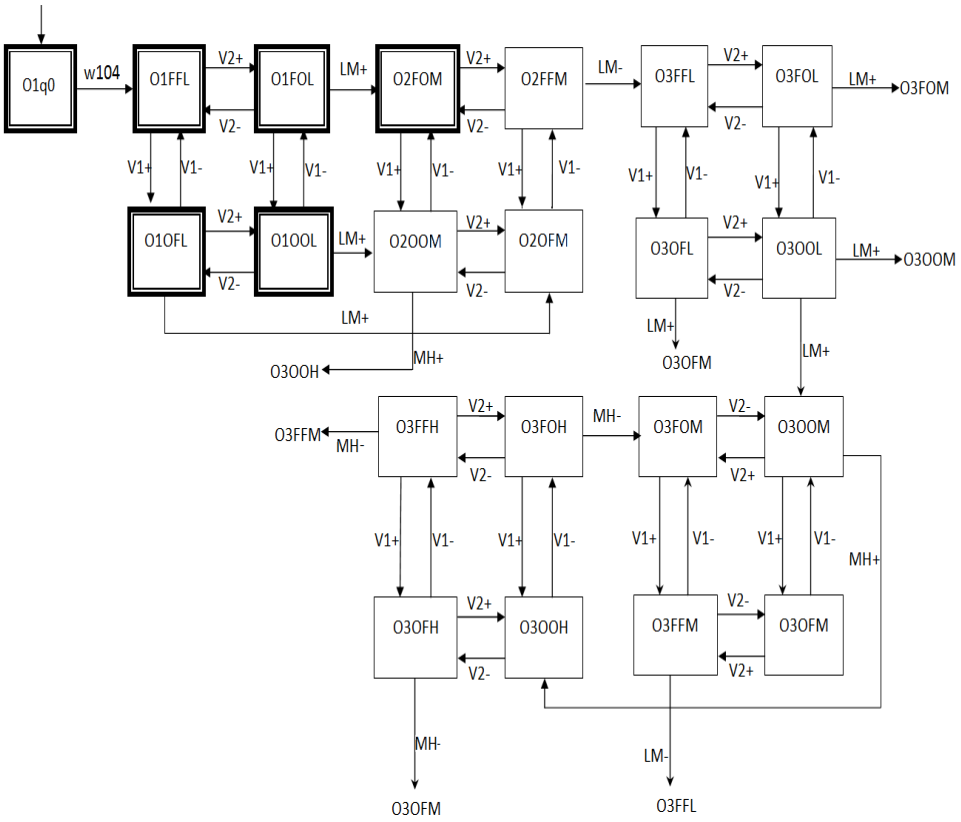

FIG. 6.7 – *Le superviseur standard*

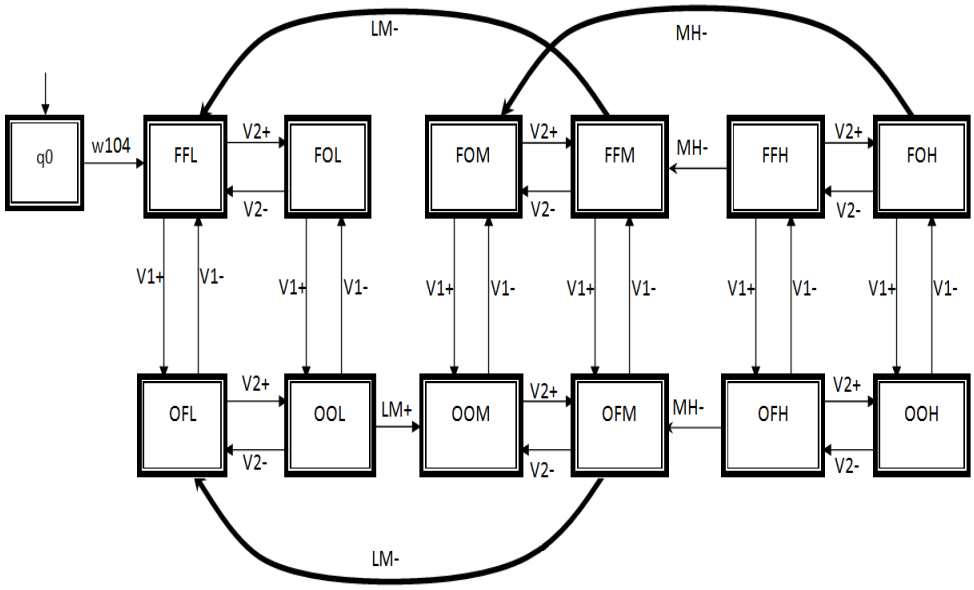

FIG. 6.8 – *Schéma correspondant à une fuite*

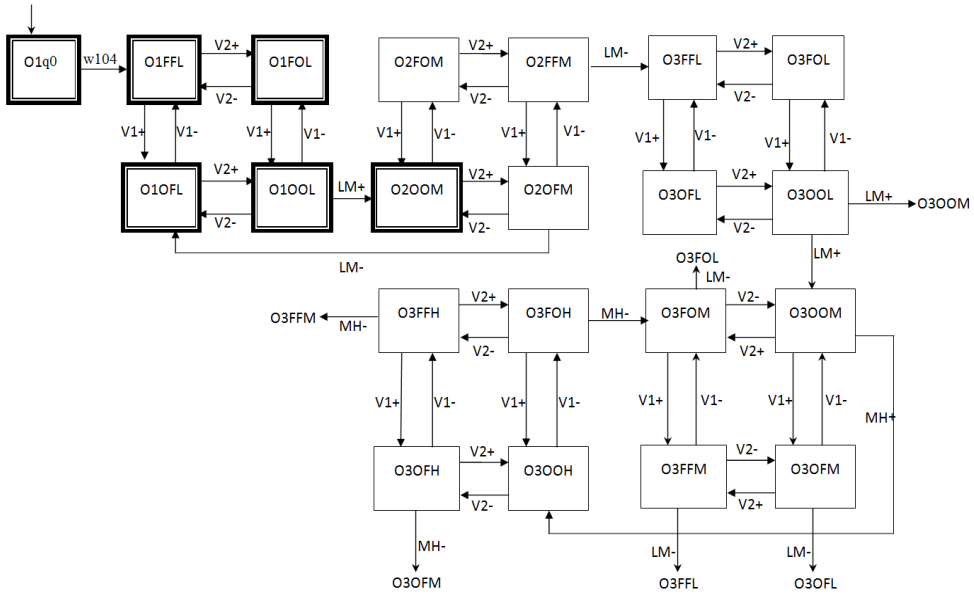

FIG. 6.9 – *Superviseur associé au système décrit dans la Figure 6.8*

Par conséquent, la reconfiguration peut démarrer dès la détection et la mise à jour initiale du modèle global. La détection s'accompagne immédiatement de l'ajout de toutes les transitions possibles de la région concernée par des nouvelles dynamiques vers leurs voisins : les transitions ajoutées sont des commutations causées par des commandes continues ou discrètes vers des états voisins. Par la suite, le diagnostic couplé à l'abstraction permet la réduction progressive de nouveaux comportements, c'est à dire des transitions superflues.

Considérons par exemple une fuite dans le réservoir $T1$ couplée avec un blocage en état ouverture de la vanne $V1$. Dans ce cas la détection d'une défaillance s'accompagne par une incertitude par rapport à l'évolution du niveau de fluide dans le réservoir T2. En effet, lorsqu'une des vannes est ouverte, on ne sait plus le flux délivré par le réservoir T1 vers le réservoir T2. Par conséquent, on ne sait pas si le niveau de fluide dans T2 augmente ou diminue.

Ainsi, à l'instant de la détection d'une défaillance, des transitions incertaines sont ajoutées à l'automate pour les états discrets associés à V1 ou V2 ouverte.

L'algorithme d'abstraction selon un contexte exposé dans le chapitre 5 est alors réexécuté pour le comportement du contexte $c = (r_{src}, r_{dest}, r_I)$ lorsque le comportement hybride dans la région r_{src} est modifié à cause de la défaillance. Par exemple, lorsqu'il y'a une fuite dans le réservoir T2 du système à deux réservoirs, le comportement dans la région moyenne n'est plus le même d'ou la nécessité de réexécution de l'algorithme d'abstraction pour le comportement du contexte spécifiant la stbilisation dans la région moyenne.

La mise à jour du modèle abstrait est réalisée de la façon progressive suivante, $\rightarrow$ étant la relation de transition sur M. Nous supposons aussi que

le système est dans l'état OFM pour une durée suffisante pour l'étape de diagnostic:

1. Désactivation de l'événement $V1-$ à cause du blocage de la vanne.
2. Détection de l'existence d'une fuite:
   - Apparition de transitions à la suite de l'incertitude:
     - FOM,$MH+ \rightarrow FOH$.
     - FOM,$LM- \rightarrow FOL$.
     - OFM,$MH+ \rightarrow OFH$.
     - OFM,$LM- \rightarrow OFL$.
     - OOM,$LM- \rightarrow OOL$.
     - OOH,$MH- \rightarrow OOM$.
   - Disparition de OOM,$LM- \rightarrow OOL$.
   - Disparition de OOM,$MH+ \rightarrow OOH$.

D'ou le nouveau modèle de fonctionnement au début (Figure 6.10) et à la fin (Figure 6.11) du processus d'abstraction.

6.3.3.2 La phase de reconfiguration:

L'algorithme de reconfiguration peut être décrit comme étant une tentative de réduction du nouveau comportement vers un des schémas prédéfinis. Ceci est réalisé au fur et à mesure de la procédure de diagnostic.

Algorithme 12

1. *Étape hors ligne:*
   - *Génération de l'automate correspondant au superviseur maximal standard (qu'on nomme A).*

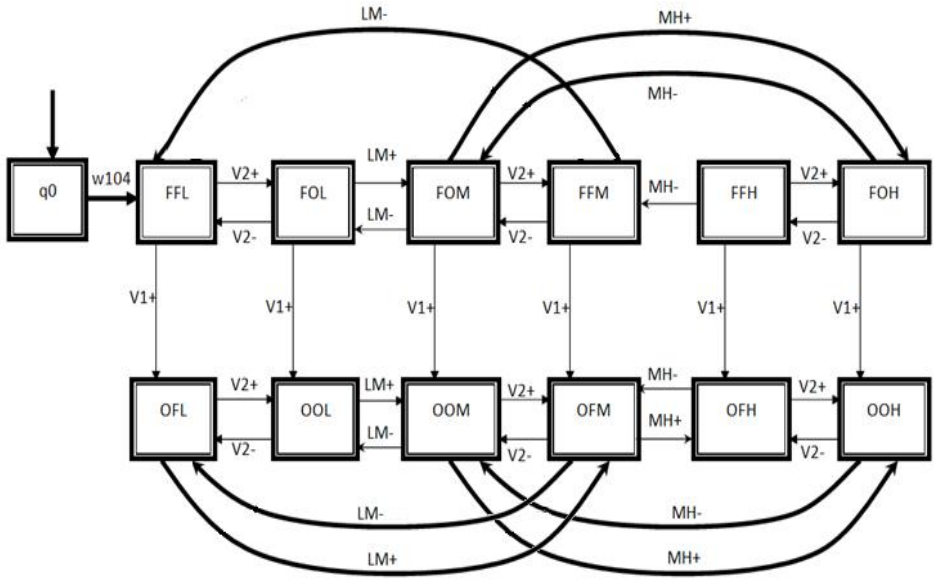

FIG. 6.10 – *Modèle disponible du système défaillant au moment de la détection*

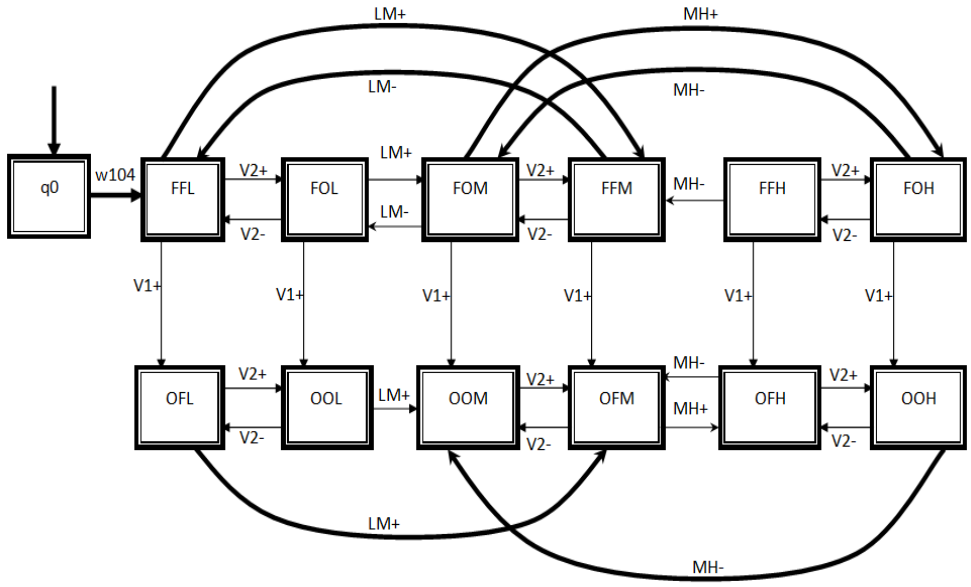

FIG. 6.11 – *Modèle du système défaillant après diagnostic*

- *Génération des automates des superviseurs maximaux correspondant aux schémas $\{M_i\}_i$ pour les objectifs dégradés B'.*

2. *Étape en ligne :*
   - *Fonctionnement normal tant qu'il n'y a pas de défaillance.*
   - *Défaillance : Ajout de certaines transitions des comportements incertains standards à la suite de la détection du changement de comportement dans certains états du système hybride original.*
   - *Pour chaque automate de supervision A'_i,*
     - *Pour chaque nouvelle transition non commandable apparue qui n'est pas incluse dans le schéma M_i, s'il existe une séquence dans A'_i contenant cette transition et aboutissant à un état qui n'est pas dans les objectifs de fonctionnement, on désactive dans A'_i et M'_i les événements de commande pouvant causer ce déclenchement.*
     - *Pour chaque nouvelle transition commandable apparue qui n'est pas incluse dans le schéma M_i, s'il existe une séquence dans A'_i déclenchée par cette transition et aboutissant à un état qui n'est pas dans les objectifs de fonctionnement, on désactive les transitions dans A'_i et M_i qui lui correspond.*
     - *Si M' est incluse dans un des schémas prédéfinis M_i et que l'état courant peut être amené à un état final dans M_i : adopter le superviseur associé A_i. Fin de l'algorithme avec succès.*
   - *À chaque fois que le modèle est mis à jour par les algorithmes d'abstraction et qu'une nouvelle transition $q,\sigma \rightarrow q'$ disparaît.*
     - *Pour chaque schéma M_i,*
       - *Si aucune séquence de transitions non commandables conte-*

nant cette nouvelle transition n'aboutit à un état non objectif.

- *σ est commandable : activer cette transition.*
- *σ n'est pas commandable : activer les transitions commandables pouvant déclencher cette transition sans risque de violation des objectifs.*
- *Si M' est incluse dans un des schémas prédéfinis M_i et que l'état courant puisse être amené à un état final dans M_i : adopter le superviseur associé A_i. Fin de l'algorithme avec succès.*

FIN.

De façon informelle, l'algorithme de reconfiguration essaie à chaque instant de trouver une réduction telle que l'un des superviseurs puisse être compatible avec les objectifs de fonctionnement énoncés; ceci au fur et à mesure que le comportement du système est précisé et durant le déroulement des algorithmes d'abstraction. Pour illustrer cet algorithme, nous l'exécutons sur l'exemple des deux réservoirs.

6.3.3.3 Application de la reconfiguration par réduction sur l'exemple des deux réservoirs

Étant donné les superviseurs synthétisés dans les Figure 6.9 et 6.7 pour les schémas prédéfinis dans les Figure 6.2 et 6.8, nous exécutons l'algorithme 12.

- Défaillance : Le modèle incertain disponible est donné dans la Figure 6.10.
  - Pour le schéma standard (Figure 6.2), on ajoute des transitions

correspondant aux nouvelles transitions incertaines au superviseur de la Figure 6.7.

- Le superviseur ainsi mis à jour est donné dans la Figure 6.12.
- On désactive toutes les transitions commandables risquées en accord avec l'algorithme 12. Celles-ci sont les transitions commandables entre un état accepté du superviseur qui peuvent déclencher une séquence de commandes non commandables vers un état qui n'est pas compatible avec les objectifs de fonctionnement (Un état du superviseur qui n'est pas compatible avec les objectifs de fonctionnement est un état composé d'un état dans $A(M)$ et d'un état non accepté dans $A(B)$). Le résultat est donné dans la Figure 6.13.
- L'état courant du superviseur ($O2OFM$) n'est pas autorisé. On ne peut accepter ce superviseur à l'état actuel du diagnostic.

– Pour le deuxième schéma (Figure 6.8), on ajoute des transitions correspondant aux nouvelles transitions incertaines au superviseur de la Figure 6.9.

- Le superviseur ainsi mis à jour est donné dans la Figure 6.14.
- On désactive toutes les transitions commandables risquées en accord avec l'algorithme 12. Le résultat est donné dans la Figure 6.13.
- L'état courant du superviseur ($O2OFM$) n'est pas autorisé. On ne peut accepter ce superviseur à l'état actuel du diagnostic.

– Abstraction : Disparition des transitions OOM,$LM- \rightarrow OOL$ et de

OOM,$MH+ \rightarrow OOH$.

– Le superviseur pour le schéma standard est mis à jour dans la Figure 6.16. Une séquence d'événements commandables permet d'amener l'état courant du système (en $O2OFL$) vers un état final ($O2OOM$). Ce superviseur peut ainsi être adopté. D'ou l'arrêt de l'algorithme avec succès.

À la suite de cette étape nous disposons d'un superviseur pour le cas dégradé à partir du schéma standard. Notons qu'en poursuivant l'algorithme, nous aurions trouvé un autre superviseur à partir du schéma de fuite.

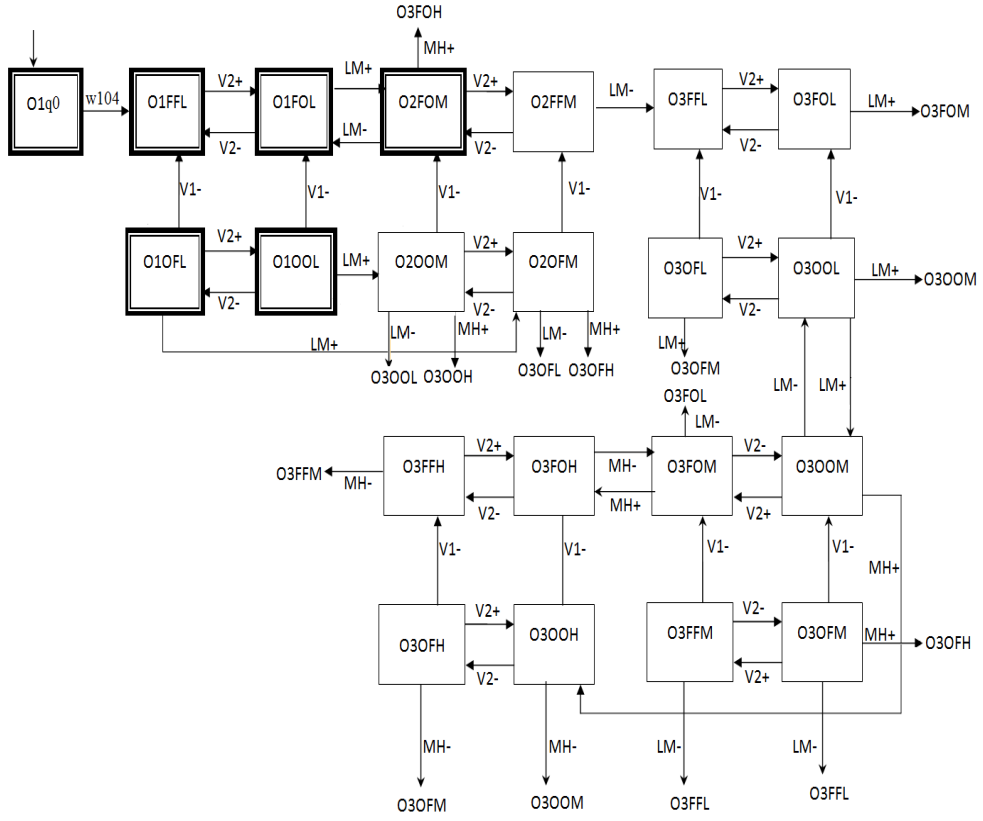

FIG. 6.12 – *Le superviseur standard après détection*

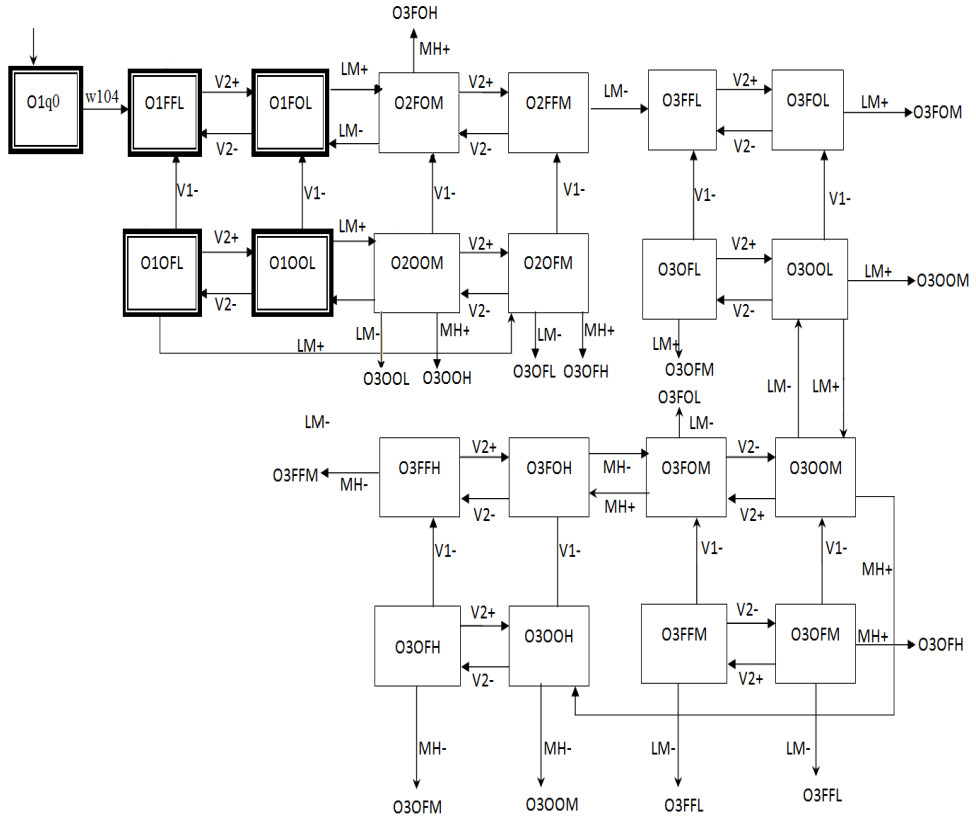

FIG. 6.13 – *Le superviseur standard après désactivation des transitions risquées*

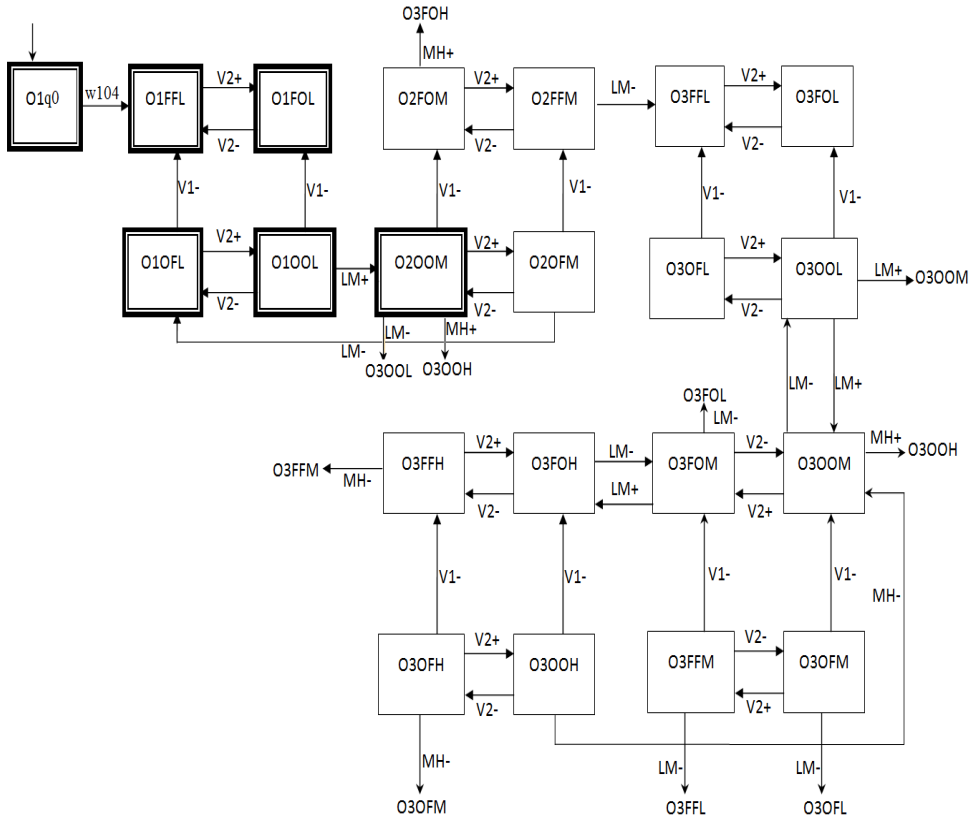

FIG. 6.14 – *Le superviseur correspondant au schéma de fuite mis à jour après détection de la défaillance*

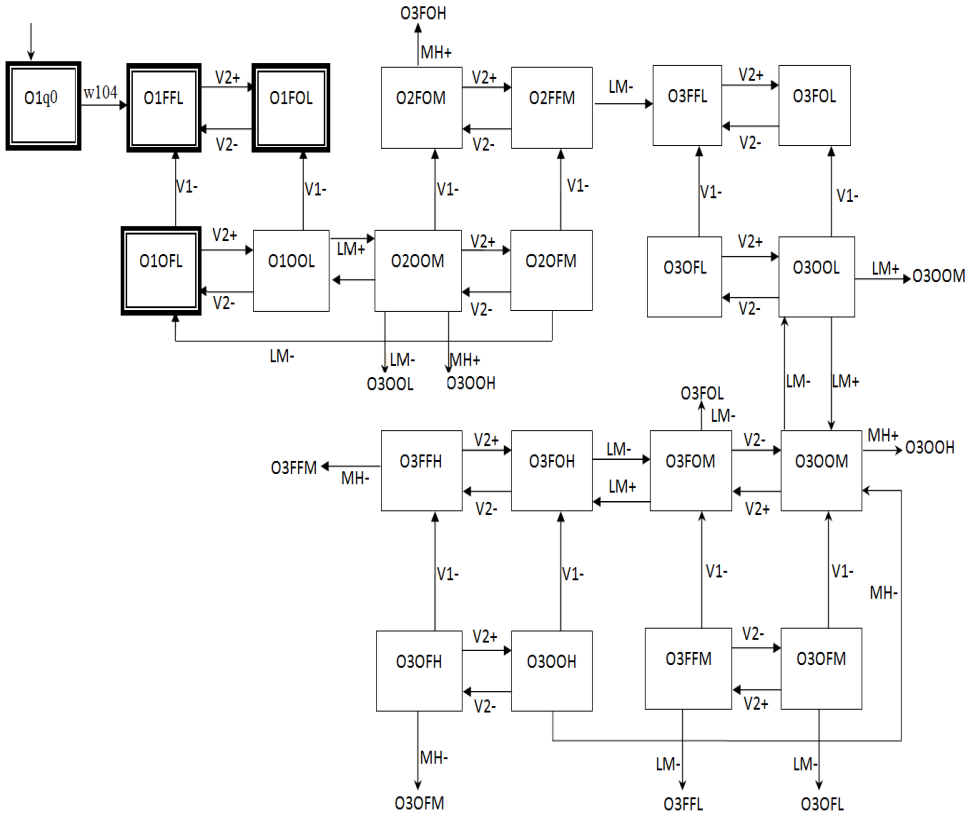

FIG. 6.15 – *Le superviseur correspondant au schéma de fuite après désactivation des transitions risquées*

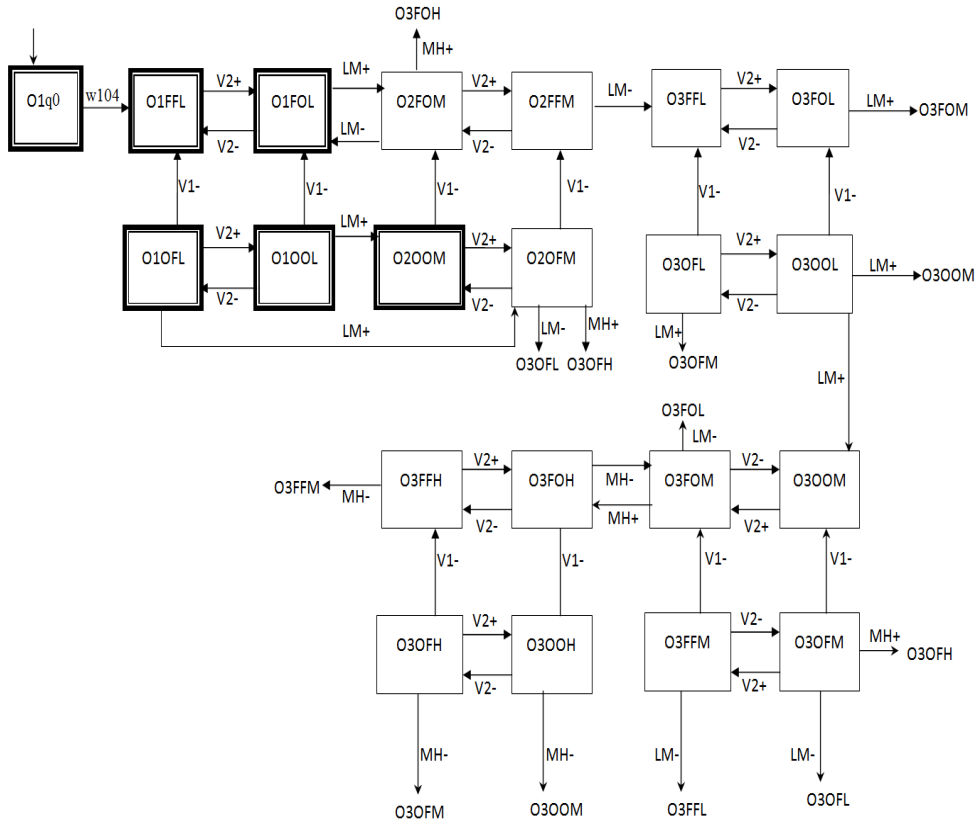

FIG. 6.16 – *Le superviseur correspondant au schéma standard après l'abstraction à la suite de la fuite dans* $T1$ *et du blocage en état ouverture de* $V1$

6.4 Conclusion

Dans ce chapitre, nous avons exposé une méthodologie de reconfiguration qui peut être greffée sur l'architecture globale exposée dans la section 1.5. Cette méthodologie est indépendante dans les faits par rapport aux algorithmes de bas niveau (de vérification d'atteignabilité, de stabilité et d'abstraction). Elle exploite cependant le résultat de ces méthodes : seuls les résultats de l'abstraction sont visibles par le module de reconfiguration. Ces informations sont exploitées progressivement au fur et à mesure du fonctionnement du diagnostic/atteignabilité/abstraction.

Conclusion

Dans ce travail, nous avons développé une architecture de commande assurant la tolérance aux fautes de façon active tout en prenant en compte les possibilités partielles de bas niveau. Pour réaliser un tel objectif, il a été nécessaire de réfléchir à une architecture permettant d'allier une représentation exhaustive de bas niveau (chapitre 2), une représentation abstraite de haut niveau, des modules réalisant des calculs pour la vérification d'atteignabilité et de stabilité (chapitres 3 et 4), et un module de supervision qui réalise l'abstraction, la mise à jour du modèle abstrait (chapitre 5) et la reconfiguration en temps réel (chapitre 6). Dans le cadre de ce travail, nous n'avons pas traité la problématique de diagnostic et d'identification. Cependant, les résultats de l'identification sont prises en compte pour la modification et affinement du système hybride décrivant le fonctionnement détaillé du système. Ceci a pour avantage de laisser plus de liberté pour l'implémentation de modules pour le diagnostic et l'identification qui soient adaptés à l'application.

Cette architecture présente l'avantage de ne nécessiter l'intervention humaine que dans la phase initiale (Construction du contrôleur standard, choix des constantes pour les approximations d'atteignabilité et de stabilisation). Notre approche de modélisation exhaustive a l'avantage de pouvoir englober une très grande classe de systèmes critiques (grâce au caractère hybride) et de défaillances (totales, partielles, progressives). En plus, cette approche permet

de supporter des descriptions incertaines en permettant aux variables continues θ de résider dans des intervalles. Lors d'une défaillance, les modules de diagnostic et d'identification (qui n'entrent pas dans le cadre de nos travaux) permettent de retourner un modèle hybride incertain qui est affiné au fur et à mesure du temps. Afin de pouvoir exploiter cette description exhaustive (de bas niveau) pour la tâche de reconfiguration (de haut niveau), nous avons développé des opérateurs pour le calcul d'atteignabilité (chapitre 3), ainsi que des techniques pour la vérification de stabilité et la génération de commandes stabilisatrices (chapitre 4).

À la suite d'une défaillance, des algorithmes d'abstraction sont utilisés en profitant des opérateurs développés pour le calcul d'atteignabilité et la stabilisation. L'abstraction du système hybride décrivant le fonctionnement du système est réalisée par rapport à un *contexte* précis; celui-ci est généré à partir des objectifs de fonctionnement énoncés dans la phase hors ligne. L'algorithme d'abstraction peut être réalisé dans le cas de systèmes à états infinis et a plus de chance de converger dans un délai fini par rapport aux approches sans contexte. Enfin, une approche active pour la reconfiguration des systèmes complexes grâce à l'utilisation de schémas prédéfinis a été réalisée. À chaque fois que le modèle abstrait est modifié par les algorithmes d'abstraction, l'algorithme de reconfiguration essaie de trouver une réduction du modèle abstrait disponible dans un des schémas prédéfinis. Une réduction réussie dans un schéma permet d'utiliser le superviseur qui lui est associé.

Cependant, cette architecture tolérante aux fautes totalement automatisée peut être améliorée pour représenter une option envisageable dans la pratique. En effet, il est nécessaire d'optimiser la complexité des différents algorithmes utilisés (Par exemple, l'algorithme utilisé pour l'abstraction ne présente pas de garantie de convergence en un nombre fini d'étapes). Ceci per-

mettrait de disposer d'une architecture permettant de trouver des solutions de reconfiguration dans un délai acceptable dans le cas de grands systèmes. Pour cela, on peut envisager une architecture modulaire et éventuellement hiérarchisée. Ceci permettra de maîtriser la complexité du système en réalisant des opérations d'abstraction locales et progressives, ce qui limite les calculs qui doivent être réalisés.

L'algorithme itératif de calcul d'atteignabilité donné dans le chapitre 3 permet une exécution dans le cas de petits systèmes continus ou hybrides. Instaurer la modularité et la hiérarchisation au sein de l'architecture proposée permet de réduire les dynamiques dans le niveau d'un module/composant. La principale difficulté réside dans le couplage existant entre les dynamiques des composants communicants. Ceci est aussi valable pour les algorithmes de stabilisation.

Les algorithmes d'abstraction ont été développés dans le cas général de systèmes à transitions. Ainsi, ils sont bien adaptés pour une hiérarchisation puisqu'ils peuvent être exécutés sur un système décrit de façon exhaustive (grâce aux opérateurs d'atteignabilité et algorithmes de stabilisation) ou sur un système abstrait (décrit en tant que système à états finis). L'algorithme d'abstraction couplé à la réduction (Algo. 11) permet la construction d'un système abstrait convenable, mais sans garantie de convergence en temps fini. Une piste de recherche est d'améliorer cet algorithme pour retourner un résultat plus rapidement (Intégration d'autres méthodologies automatiques. Par exemple: vérification de commandabilité...)

Enfin, l'algorithme de reconfiguration est réalisé par rapport à des schémas prédéfinis. Afin de pouvoir réaliser un tel algorithme, il est nécessaire de disposer d'un schéma abstrait qui ne soit pas très compliqué. La réussite de la reconfiguration est ainsi conditionnée par la réussite de l'étape d'abstraction.

Elle est d'autant plus efficace que le système abstrait est réduit. Une amélioration possible serait de considérer l'ajout d'événements non observables. Ceci permet de considérer le cas plus général dont certaines variables ne sont pas observables.

Si on parvient à améliorer l'architecture proposée de la sorte, il serait envisageable de construire des systèmes tolérants aux fautes grâce à des modules embarqués réalisant les différents algorithmes développés. Un tel système serait capable de s'adapter à son environnement en exploitant au maximum les possibilités partielles du système complexe.

Bibliographie

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis et S. Yovine. « The Algorithmic Analysis of Hybrid Systems ». *IEEE Transactions on Automatic Control*, 43:540–554, Avril 1998.

[2] R. Alur, T. Dang et F. Ivancic. « Counter-example guided predicate abstraction of hybrid systems ». 2003.

[3] R Alur, R Grosu, Y Hur, V Kumar et I Lee. « Modular Specification of Hybrid Systems in Charon ». *Hybrid Systems, Computations and Control*, pages 6–19, 2000.

[4] Rajeev Alur, Thao Dang et Franjo Ivançic. « Reachability Analysis of Hybrid Systems via Predicate Abstraction ». *Hybrid Systems : Computation and Control*, 2289/2002:758–819, 2002.

[5] Rajeev Alur et David L. Dill. « A Theory of Timed Automata ». *Theoretical Computer Science*, 126(2):183–235, 1994.

[6] Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere et George J. Pappas. « Discrete Abstractions of Hybrid Systems ». *Proceedings of the IEEE*, 88(7):971–984, 2000.

[7] Ken Arnold. *Embedded Controller Hardware Design*. Newnes, 2001.

[8] J. Askari-Marnani, B. Heiming et J. Lunze. « *Control Reconfiguration : The COSY benchmark problem and its solution by means of a*

qualitative model », Chapitre 21. European Science Foundation, 1999.

[9] Karl Aström, Pedro Albertos, Morgens Blanke, Alberto Isidori, Walter Schaulelberger et Ricardo Sanz. *Control of Complex systems*. Springer - Verlag, 2001.

[10] Pierre Bacher. « Réacteurs Nucléaires. Généralités ». *Techniques de l'Ingénieur*, (BN 3020):1–16, 2005.

[11] M. Blanke, M. Kinnaert, J. Lunze et M. Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer, 2nd edition, 2006.

[12] Morgens Blanke, Marcel Staroswiecki et N. Eva Wu. « Concepts and Methods in Fault-Tolerant Control ». *Proceedings of the 2001 American Control Conference*, 4:2606–2620, 2001.

[13] P. Borne, G. Dauphin-Tanguy, J. P. Richard, F. Rotella et I. Zambettakis. *Commande et Optimisation des Processus*. Éditions Technip, 2001.

[14] R. D. Brandt, V. Garg, R. Kumar, F. Lin, S. I. Marcus et W. M. Wonham. « Formulas for Calculating Supremal Controllable and Normal Sublanguages ». *System and Control Letters*, 15(2):111–117, 1990.

[15] Michael S. Branicky, Vivek S. Borkar et Sanjoy K. Mitter. « A Unified Framework for Hybrid Control: Model and Optimal Control Theory ». *IEEE Transactions on Autoamtic Control*, 43(1):31 – 45, 1998.

[16] Michael Stephen branicky. « *Studies in Hybrid Systems : Modeling, Analysis, and Control* ». PhD thesis, Massaschussetts Institute of Technology, 1995.

[17] M.S. Branicky. « Multiple Lyapunov functions and other analysis tools for switched and hybrid systems ». *Automatic Control, IEEE Transactions on*, 43(4):475–482, Apr 1998.

[18] P.E. Caines et Yuan-Jun Wei. « Hierarchical hybrid control systems: a lattice theoretic formulation ». *Automatic Control, IEEE Transactions on*, 43(4):501–508, Apr 1998.

[19] Alongkrit Chutinan. « *Hybrid System Verification Using Discrete Model Approximations* ». PhD thesis, Department of Electrical and Computer Engineering Carnegie Mellon University, 1999.

[20] Alongkrit Chutinan et Bruce H. Krogh. « Computing Polyhedral Approximations to Flow Pipes for Dynamics Systems ». *Proceedings of the 37th IEEE Conference on Decision and Control 1998*, 2:2089–2094, December 1998.

[21] Alongkrit Chutinan et Bruce H. Krogh. « Verification of Infinite-StateDynamic Systems Using Approximate Quotient Transition Systems ». *IEEE Transactions on Automatic Control*, 46(9):1401, 1410, 2001.

[22] Etienne Craye. « Contribution au Contrôle/Commande de Systèmes Flexibles de Production Manufacturière ». *Habilitation à diriger les recherches, Université des Sciences et Technologies de Lille, Lille*, 1994.

[23] Thao Dang. « Approximate Reachability Computation for Polynomial Systems ». *Hybrid Systems : Computation and Control, Lecture Notes in Computer Science*, 3927/2006:138–152, 2006.

[24] René David et Hassane Alla. « On Hybrid Petri Nets ». *Discrete Event Dynamic Systems*, 11:9–40, 2001.

[25] Jack Ganssle. *The Art of Designing Embedded Systems*. Newnes, 2008.

[26] Sylviane Gentil. *Supervision des Procédés Complexes*. Hermès, Lavoisier, 2007.

[27] Rafal Goebel, Joao Hespanha, Andrew R. Teel, Chaohong Cai et Ricardo Sanfelice. « Hybrid Systems : Generalized Solutions and Ro-

bust Stability ». *IFAC Symposium on Nonlinear Control Systems, Stuttgart, Germany*, 2004.

[28] J.K. Hedrick, M. Tomizuka et P. Varaiya. « Control issues in automated highway systems ». *Control Systems Magazine, IEEE*, 14(6):21–32, Dec 1994.

[29] M.R. Henzinger, T.A. Henzinger et P.W. Kopke. « Computing simulations on finite and infinite graphs ». *Foundations of Computer Science, Annual IEEE Symposium on*, 0:453, 1995.

[30] T.A. Henzinger, Pei-Hsin Ho et H Wong-Toi. « The Algorithmic Analysis of nonlinear Hybrid Systems ». *IEEE Transactions on Automatic Control*, 43:540–554, Avril 1998.

[31] Thomas A. Henzinger. « Hybrid Automata with Finite Bisimulations ». *proceedings of the 22nd International Colloquium on Automata Languages and Programming (ICALP), Lecture Notes in Computer Science*, 944:324–335, 1995.

[32] Thomas A. Henzinger. « The Theory of Hybrid Automata ». *Symposium on Logic in Computer Science, 1996. LICS '96. Proceedings., Eleventh Annual IEEE*, pages 278–292, 1996.

[33] Thomas A. Henzinger et Jean-François Raskin. « Robust Undecidability of Timed and Hybrid Systems ». *Proceedings of the International Workshop on Hybrid and Real-Time Systems*, pages 331–345, 1999.

[34] J.P. Hespanha et A.S. Morse. « Stability of switched systems with average dwell-time ». volume 3, pages 2655–2660 vol.3, 1999.

[35] J. E. Hopcroft et J; D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Reading, MA: Addison-Wesley, 1979.

[36] R. Horowitz et P. Varaiya. « Control design of an automated highway system ». *Proceedings of the IEEE*, pages 913–925, 2000.

[37] Alberto Isidori. *Nonlinear Control Systems.* Springer, 1995.

[38] Y. Kesten et A. Pnueli. « Timed and Hybrid Statecharts and Their Textual Representation ». Dans J. Vytopil, éditeur, *Formal Techniques in Real-Time and Fault-Tolerant Systems 2nd International Symposium*, volume 571, pages 591–, Nijmegen, The Netherlands, 1992. Springer-Verlag.

[39] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy et H. Van Brussel. « Reconfigurable Manufacturing System ». *A Keynote paper, CIRP annals*, 48(2):6–12, Novembre 1999.

[40] X.D. Koutsoukos, P. J. Antsaklis, J.A. Stiver et M.D. Lemmon. « Supervisory Control of Hybrid Systems ». *Proceedings of the IEEE*, 88:1026–1049, July 2000.

[41] Xenofon Koutsoukos, Feng Zhao, Horst Haussecker, Jim Reich et Patrick Cheung. « Fault modeling for monitoring and diagnosis of sensor-rich hybrid systems ». Dans *In Proc. of the 40th IEEE Conference on Decision and Control*, pages 793–801, 2001.

[42] Alexander B. Kurzhanski. « Ellipsoidal Techniques for Reachability Analysis ». *Hybrid Systems : Computation and Control, Lecture Notes in Computer Science*, 1790/2000:202–214, 2000.

[43] G. Lafferriere, G. J. Pappas et S. Sastry. « Hybrid Systems with Finite Bisimulations ». Rapport Technique, Hybrid Systems V, Lecture Notes in Computer Science, April 1998.

[44] G. Lafferriere, G. J. Pappas et S. Yovine. « Decidable Hybrid Systems ». Rapport Technique, University of California at berkeley, June 1998.

[45] Gerardo Lafferriere, George J. Pappas et Sergio Yovine. « Symbolic Reachability Computation for Families of Linear Vector Fields ».

Journal of Symbolic Computation, 32(3):231–253, 2001.

[46] Eun Joo Lee. « *Reconfiguration dynamique de la commande d'un système manufacturier : approche par la synthèse de la commande* ». PhD thesis, Ecole Centrale de Lille et Université des Sciences et Technologies de Lille, 2006.

[47] F. Lin. « *On Controllability and Observability of Discrete Event Systems* ». PhD thesis, Department of Electrical Engineering, University of Toronto, November 1987.

[48] F. Lin et W. M. Ramadge. « On Observability of Discrete-Event Systems ». *Information Sciences*, 44:173–198, 1988.

[49] Lionel Lorimier. « *La caractérisation dynamique des défaillances, Une nouvelle approche pour la gestion active des défaillances au sein des systèmes physiques industriels complexes* ». PhD thesis, Ecole Centrale de Lille et Université des Sciences et Technologies de Lille, 2005.

[50] J. Lygeros, K.H. Johansson, S. Sastry et M. Egerstedt. « On the existence of executions of hybrid automata ». *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, 3:2249–2254 vol.3, 1999.

[51] John Lygeros, Datta N. Godbole et Mireille broucke. « A fault tolerant control architecture for automated highway systems ». *IEEE Transactions on Control Systems Technology*, 8:205–219, 2000.

[52] John Lygeros, Karl Henrik Johansson, Slobodan N. Simić, Jun Zhang et Shankar Sastry. « Dynamical Properties of Hybrid Automata ». *IEEE Transactions on Automatic Control*, 48(1):2–17, 2003.

[53] Thomas Moor et Jörg Raisch. « *Abstraction Based Supervisory Controller for High Order Monotone Continuous Systems* », volume

279/2002, Chapitre 14, pages 246–266. Springer Berlin/Heidelberg, 2002.

[54] Raymond MURRAY. *Nuclear Energy: An Introduction to the Concepts, Systems and Applications of Nuclear Processes*. Butterworth-Heinemann Ltd, 2000.

[55] Nicolas NAVET et Françoise SIMONOT-LION. *Embedded Controller Hardware Design*. Newnes, 2008.

[56] Andrea PAOLI, Matteo SARTINI et Stéphane LAFORTUNE. « A Fault Tolerant Architecture for Supervisory Control of Discrete Event Systems ». *Proceedings of the 17th IFAC World Conference*, pages 6542– 6547, 2008.

[57] J. L. PETERSON. *Petri Net Theory and the Modeling of Systems*. Reading, MA: Addison-Wesley, 1979.

[58] Gérald POINT. « *AltaRica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement* ». PhD thesis, Université Bordeaux 1, École Doctorale de Mathématiques et Informatique, 2000.

[59] Anuj PURI, Pravin VARAIYA et Vivek BORKAR. « ϵ-Approximation of Differential Inclusions ». *Proceedings of Hybrid Systems : Computation and Control, Lecture Notes in Computer Science*, 1066:362–376, 1996.

[60] P. J. G. RAMADGE et W. M. WONHAM. « The control of discrete event systems ». *Proceedings of the IEEE*, 77:81–98, 1 1989.

[61] J. P. RICHARD. *Algèbre et Analyse pour l'Automatique*. Hermes, 2001.

[62] Renaud SIRDEY. « *Modèles et algorithmes pour la reconfiguration des systèmes répartis utilisés en téléphonie cellulaire* ». PhD thesis, Université de Technologie de Compiègne, UMR CNRS Heudiasyc, 2007.

[63] James A. STIVER, Panos J. ANTSAKLIS et Michael D. LEMMON. « Interface and Controller Design for Hybrid Control Systems ». Dans

Hybrid Systems II, pages 462–492, London, UK, 1995. Springer-Verlag.

[64] Carlton Stoiber, Alec Baer, Norbert Pelzer et Wolfram Tonhauser. *Handbook on Nuclear Law*. International Atomic Energy Agency, Vienna, 2003.

[65] Ashish Tiwari et Gaurav Khanna. « *Nonlinear Systems : Approximating reach Sets* », volume 2993/2004, pages 600–614. 2004.

[66] P. Varaiya. « Smart Cars on Smarts Roads : Problems of Control ». *IEEE Transactions on Automatic Control*, 38(2), 1993.

[67] W. M. Wonham et P. J. Ramadge. « On the supremal controllable sublanguage of a given language ». *Proceedings of the 23rd IEEE Conference on Decision and Control*, 23:1073–1080, decembre 1984.

[68] W. M. Wonham et P. J. Ramadge. « Modular supervisory control of discrete-event systems ». *Mathematics of Control, Signals, and Systems (MCSS)*, 1(1):13–30, 1988.

[69] S. Hashtrudi Zad, R. H. Kwong et W. M. Wonham. « Fault diagnosis in finite-state automata and timed discrete-event systems ». Dans *In 38th IEEE Conference on Decision and Control*, 1999.

[70] J. Zaytoon. *Systèmes dynamiques hybrides*. hermes-sciences, 2001.

[71] H. Zhong. « *Hierarchical Control of Discrete Event Systems* ». PhD thesis, University of Toronto, 1992.

[72] Richard Zurawski. *Embedded Systems Handbook*. CRC, 1 édition, 2005.

[73] Gilles Zwingelstein. *Diagnostic des défaillances, théorie et pratique pour les systémes industriels*. éditions Hermes, 1995.

Annexe A

Applications de la tolérance aux fautes des systèmes complexes

L'importance de la tolérance aux fautes dépend de la nature de l'application et des exigences utilisateurs. En plus, elle dépend de la technologie et méthodologie de conception utilisée et des contraintes d'environnement. Les effets d'une défaillance lors du fonctionnement peuvent être qualifiés de bénignes, tolérables ou catastrophiques selon les diverses conditions déjà citées. Dans cette section, nous donnons un aperçu de quelques applications notables de la tolérance aux fautes.

A.0.1 Les télécommunications

On peut définir la télécommunication comme étant la transmission à distance de signaux porteurs d'informations grâce à des moyens informatiques et électroniques. Les télécommunications englobent plusieurs disciplines telles que les réseaux locaux industriels, le téléphone, la radio, la télémesure (technique permettant de lire à distance les données d'un appareil de mesure)...

Parmi toutes ces applications, la tolérance aux fautes est très importante

pour les réseaux locaux industriels ou des machines au vol. Elle l'est aussi pour l'avionique (qui sera présentée dans le paragraphe A.0.3). Par la suite, nous exposerons deux applications typiques de la tolérance aux fautes dans le domaine des télécommunications.

A.0.1.1 Reconfiguration en téléphonie cellulaire

Dans ce paragraphe, nous donnons un bref aperçu de la problématique de tolérance aux fautes dans le cas d'un réseau GSM (cas particulier des systèmes de téléphonie cellulaire). Voir les travaux de Renaud Sirdey [62] pour plus de détails.

Un réseau GSM est construit autour des composants suivants :

- Les sous systèmes radio (Base Station Subsystem : BSS) : c'est la partie radio du système chargée de la commutation entre le système mobile (MS : Mobile System) et la partie commutation. Il est composé de BTS (Base Transceiver Station) déployés dans la zone à desservir, de contrôleurs de stations radio (BSC : Base Station Controller) dont chacun permet de gérer plusieurs BTS et permet une première concentration du traffic. Enfin, un BSS contient des transcoders pour la transmission des données dans les médias physiques.
- Les sous systèmes d'acheminement (Network and Switching System : NSS) : C'est un composant du système GSM permettant la commutation (grâce à des MSC (Mobile Switching Center) associés aux bases de données VLR (Visitor location register)). Elle contient aussi la base de données HLR (Home Location Register) stockant des détails à propos de chaque abonné.
- Les sous systèmes d'exploitation et de maintenance ou OSS (Operation SubSystem) qui permettent la maintenance et la configuration des

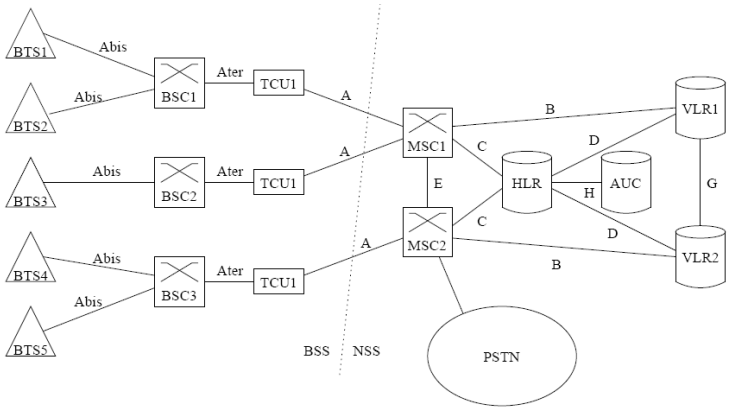

FIG. A.1 – *Principe général de la commande*

composants de réseau, l'approvisionnement des services et la gestion des défauts. Ce sous système est composé de deux OMU (Operation and Maintenance Unit) en redondance passive qui sont responsables de la gestion des équipements du sous système de contrôle, du contrôle du sous système d'interface...

La A.1 résume l'architecture globale du système. Plus de détails sont disponibles dans [62].

Un tel réseau démarre avec de bonnes propriétés telles que l'équirépartition de la charge. Il est sujet à des défaillances telles que la mise hors tension d'une unité de traitement d'appel.

L'objectif de la tolérance aux fautes dans le cas d'un réseau cellulaire de type GSM est d'assurer un fonctionnement avec une bonne qualité de service du réseau selon l'état des équipements du système. Une approche de redondance passive logicielle a été prévue pour assurer la protection de la charge de traitement d'appels. Elle est réalisée grâce à l'utilisation d'un processus actif et d'un processus passif tournant sur des unités de management du trafic gérant les traitements d'appels (TMU : Traffic Management Unit) différents. Le processus passif peut prendre la main à chaque instant si le processus

passif ne fonctionne pas de façon concluante. L'utilisation de la redondance passive logicielle est aussi étendue aux sous systèmes d'exploitation et de maintenance.

La mise hors tension d'une unité de traitement d'appel peut nécessiter d'entreprendre des modifications telle que la limitation du nombre d'opérations réalisées sur la charge de traitement d'appels et le redémarrage des processus actifs perdus ou manquants. Par la suite, il est possible que le fonctionnement du système n'ait pas la même qualité de service qu'avant la défaillance : déséquilibre de charge, manque de certains processus passifs... Ainsi, une procédure supplémentaire peut être nécessaire afin de restaurer un état équivalent au démarrage. Une telle modification peut être nécessaire aussi lors de l'ajout ou retrait de stations dans le réseau.

Les aspects de la procédure de reconfiguration et les considérations pratiques par rapport à la conception et mise en œuvre d'un tel système sont discutés dans [62].

A.0.2 Les centrales nucléaires

Une centrale nucléaire est composée de plusieurs tranches. Chaque tranche est constituée de réacteurs nucléaires, les générateurs de vapeur, des circuits d'eau principaux et secondaires, une turbine à vapeur, un alternateur et un condenseur. L'architecture globale peut être résumée par la Figure A.2.

Dans une tranche nucléaire, le réacteur nucléaire réalise de façon continue des réactions en chaîne contrôlées (en maintenant une masse critique de combustible nucléaire dans le réacteur) qui produisent de l'énergie calorifique. Celle ci est transformée en énergie mécanique à l'aide d'une turbine à vapeur. Enfin, l'énergie mécanique est convertie en électricité grâce à l'alternateur.

La construction d'une centrale nucléaire obéit à diverses contraintes im-

posées par des régulations, normes et traités internationaux. Ces règles sont spécifiées par divers organismes à travers les pays maitrisant la technologie nucléaire tel que l'Autorité de sûreté nucléaire aux États Unis (*Nuclear Regulatory Commission*), l'Autorité de sûreté nucléaire en France, la commission de l'énergie atomique au Japon... Un organisme important au niveau international est l'Agence Internationale de l'Énergie Atomique (AIEA). Les principales contraintes devant être respectées sont :

- la sûreté de fonctionnement : le contrôle de la réaction en chaîne, l'évacuation à tout moment de l'énergie produite et le confinement de la radioactivité [10]. Ces contraintes doivent être vérifiables et assurées à un niveau *raisonnable* [54].
- Tenir compte des quantités de déchets nucléaires produits et des conséquences pour l'environnement [10]. Dans ce contexte, l'AIEA a identifié plusieurs conditions nécessaires pour avoir une licence de production et d'utilisation des déchets radioactifs : donner des garanties de sécurité, existence de plans d'urgence dans le cas d'accidents... Voir [64].
- Assurer la non-prolifération nucléaire, ce qui limite les possibilités technologiques acceptables pour les industriels (tel que l'enrichissement du plutonium en qualité militaire [10, 64]).

Durant une expérience de quelques dizaines d'années, des centrales nucléaires ont été sujettes à des accidents plus ou moins dramatiques. Afin de garantir une sûreté de fonctionnement à un niveau raisonnable selon les normes fixées par les organismes internationaux, la conception d'une centrale doit prendre en comptes les risques suivants :

- Utilisation des matériaux radioactifs : Pour cela, il est nécessaire de prévoir des outils pour une utilisation inoffensive des matériaux et dé-

chets radioactifs tout au long du fonctionnement. En plus, les matériaux sont stockés dans des conteneurs particulièrement étanches. Des tests doivent être accomplis pour vérifier ces propriétés. Des plans sont prévus pour gérer les risques possibles.

– Vulnérabilité face à une attaque terroriste ou militaire : En plus de la garde assignée aux installations et de la robustesse des clôtures contrôlées électroniquement. La procédure d'arrêt doit être particulièrement rapide (elle est estimée de l'ordre de quelques secondes dans plusieurs centrales). En plus, des procédures de gestion d'incendies ou explosions sont toujours prévues.

Les centrales nucléaires sont des systèmes artificiels d'une grande complexité. Ceci est dû à une utilisation intensive de la redondance matérielle pour être en conformité avec toutes les contraintes citées. Une stratégie de défense en profondeur doit être implémentée afin de réduire le risque d'accidents dûs à une combinaison d'erreurs humaines et matérielles. En plus, une stratégie corrective est prévue pour chaque mode de défaillance possible.

A.0.3 Les systèmes embarqués

On désigne par systèmes embarqués les systèmes informatiques qui sont intégrés dans un système artificiel plus grand, adaptés à un contexte temps réel et sont dédiés à des tâches spécifiques. Le caractère dédié de tels systèmes permet une optimisation de la structure, taille et coût de tels systèmes. Souvent, des contraintes fortes par rapport à la fiabilité, sécurité et disponibilité sont exigées de la part des systèmes embarqués puisqu'elles sont incorporées dans des processus critiques et qu'elles implémentent en général des fonctionnalités de commande importantes. Citons par exemple la commande des

systèmes de vol, la commande des centrales de production d'énergie, les équipements médicaux, les systèmes de transport... Dans le cas de tels systèmes, le degré de sûreté est précisé selon des apprôches quantitatives/qualitatives analytiques/expérimentales.

Les systèmes embarqués peuvent comporter des microcontrôleurs, des microprocesseurs, des réseaux locaux, des interfaces d'affichage et de surveillance, des périphériques pour la commande (convertisseurs, bus, compteurs...), des mémoires et bases de données auxiliaires... Il est évident qu'assurer un niveau de tolérance aux fautes requiert alors un soin particulièrement méticuleux dans la conception de chaque composant, de l'intégration des programmes et structures de données implémentées. Notons que des modes de fonctionnements considérés comme sains pour des composants individuels peuvent ne pas l'être dans le cas du système en globalité, ce qui limite fortement les possibilités de commande tolérante aux fautes dans le cas d'un système complexe large. Par exemple, on ne peut accepter la remise à zéro d'un calculateur de commande de vol lorsqu'une faute logicielle est détectée en plein vol.

Pour cela, la tolérance aux fautes est implémentée à chaque niveau d'abstraction du système embarqué : Par exemple, une redondance matérielle permet de réduire le risque d'échec à cause de défaillances de capteurs ou actionneurs. La redondance au niveau logiciel permet de réduire le risque de perte ou corruption des données dans les réseaux ou supports de stockage. Les protocoles implémentés de transfert de données doivent être particulièrement fiables (avec checksum, confirmation...).Une stratégie de défense en profondeur est peut être utilisée afin d'augmenter la sûreté. D'autres mécanismes peuvent encore être envisagés selon le cas. Par exemple : l'utilisation d'un compteur chien de garde pour vérifier le non blocage du système à une étape

donnée, le développement de code selon des pratiques prenant en compte les possibilités d'erreurs de fonctionnement des composants (Immunity Aware Programming), la conception selon le pire des cas d'évolution temporelle... Voir les références [7, 25, 62, 72] pour plus de détails.

Malgré les difficultés induites par la tâche de quantification des risques d'échec de fonctionnement dans le cas de systèmes complexes embarqués, des processus de certification ont été établis dans divers domaines permettant de formuler des contraintes assez vérifiables de la fiabilité, disponibilité et sûreté de fonctionnement. Citons par exemple la certification RTCA/DO-178B pour les logiciels en avionique. Le niveau de risque de sécurité et d'intégrité a été fixé par la norme IEC 61508 selon les niveaux SIL 1 ($10^{-6} < P < 10^{-5}$), SIL 2 ($10^{-7} < P < 10^{-6}$), SIL 3 ($10^{-8} < P < 10^{-7}$), SIL 4($P < 10^{-8}$). P étant la probabilité d'échec de fonctionnement dangereux par heure.

L'exemple de la sûreté de fonctionnement dans le cas des véhicules autonomes dans [55] expose plusieurs considérations pour la conception des systèmes embarqués.

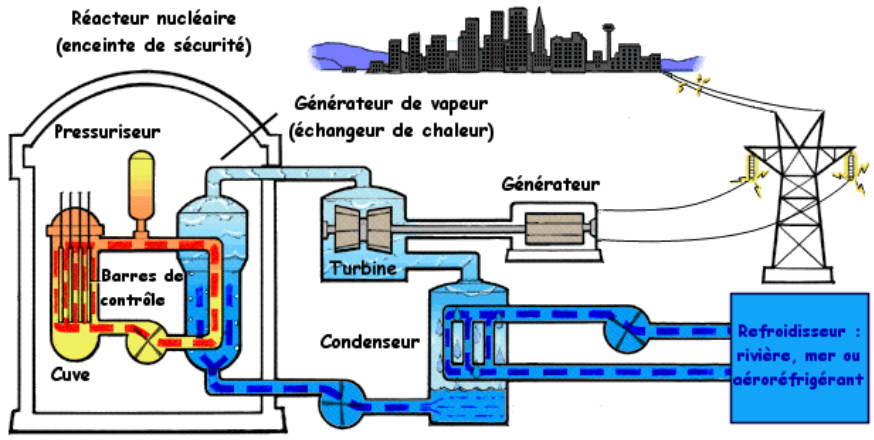

FIG. A.2 – *Architecture générale d'une centrale nucléaire. Source : wikipedia, traduite du NRC (Nuclear Regulatory Commission)*

Annexe B

Application sur l'exemple des deux réservoirs

Lors de l'approximation réalisée dans l'exemple de la section 3.3.2, nous nous sommes aperçu que l'approximation réalisée de façon itérative fait gagner en efficacité, mais que le gain en efficacité n'est pas très grand. Ceci est dû à la nature monotone du système étudié (système linéaire). En effet, la divergence de l'erreur était assez proche de la divergence réelle des points atteints.

Dans cet exemple, nous allons montrer que le gain en efficacité peut être très important dans des cas non monotones.

Considérons le système du pendule inversé du type :

$$\ddot{\theta} + \frac{\delta}{m}\dot{\theta} + \frac{g}{l}sin(\theta) = 0 \tag{B.1}$$

Avec :

- δ : frottement
- l : longueur

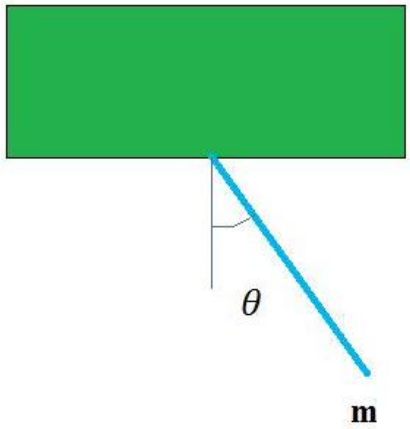

FIG. B.1 – *Schéma global du pendule inversé*

- m : masse
- θ : Angle
- g : constante de gravité

Dans ce qui suit nous adoptons la norme infinie :

$$\|x\| = max\{|x_i|\}$$

On peut reformuler l'équation B.1 sous la forme canonique selon:

$$\begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix}' = f(\theta,\dot{\theta}) = \begin{pmatrix} \dot{\theta} \\ -\frac{\delta}{m}\dot{\theta} - \frac{g}{l}sin(\theta) \end{pmatrix} \tag{B.2}$$

Un point $(\theta,\dot{\theta})$ définit un point exact de l'espace de phase du système. Par la suite, on va chercher une constante k pour la propriété Lipchitzienne. Après calculs (voir annexe C), on obtient:

$k = max(1,2 * max(0.4/0.5,9.8/0.3)) = 65.4$

En développant, on obtient :

$$\left\| f(\theta_1,\dot{\theta}_1) - f(\theta_2,\dot{\theta}_2) \right\| \leq k \left\| \begin{pmatrix} \theta_1 - \theta_2 \\ \dot{\theta}_1 - \dot{\theta}_2 \end{pmatrix} \right\| \tag{B.3}$$

Soit $\hat{epsilon} = 0.01$, Même avec un rayon maximal aussi réduit, on a l'erreur qui diverge à $t = 10$ vers un nombre très grand (MATLAB retourne alors le symbole Inf de l'infini). Essayons cependant d'obtenir une approximation avec la méthode itérative. Sur une durée de 0.01 seconde, et avec un découpage à rayon maximal $\epsilon = 0.01$, on aura 1000 étapes à 0.01 secondes, avec une erreur à chaque fois de $\alpha = \epsilon \times e^{65.4*0.01} = 0.0192$; dans ce cas, la simulation est un peu plus lente, puisqu'on doit refaire un découpage à chaque 0.01 seconde, mais le résultat est de loin plus précis.

D'ou l'approximation obtenue par les figures suivantes:

Chaque trajectoire à partir d'une cellule à partir de la région origine est distinguée par une couleur différente. On voit bien que la région atteignable est nécessairement contractante.

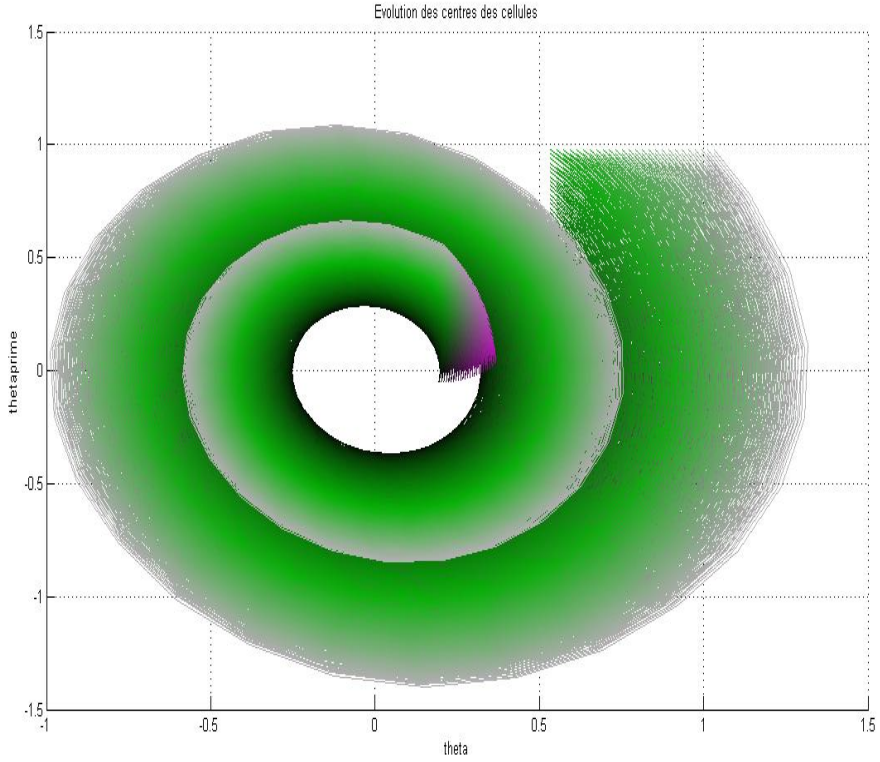

FIG. B.2 – *vue en deux dimension de l'évolution de l'approximation*

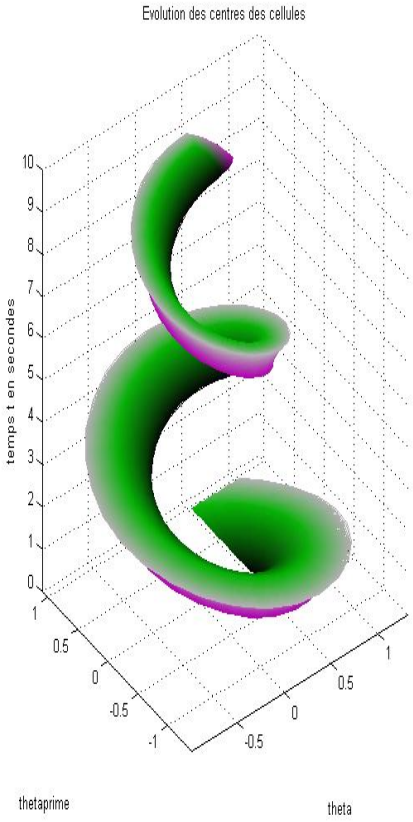

FIG. B.3 – *Vue en trois dimensions avec l'axe temporel*

Annexe C

Calcul de constante lipchitzienne pour l'exemple 2 du chapitre 3

Soient les deux points $(\theta_1,\dot{\theta}_1)$, $(\theta_2,\dot{\theta}_2)$ de l'espace de phase. On a :

$$
\begin{aligned}
&\left\| f(\theta_1,\dot{\theta}_1) - f(\theta_2,\dot{\theta}_2) \right\| \\
&= max\left\{ \left\| -\frac{\delta}{m}\dot{\theta}_1 - \frac{g}{l}sin(\theta_1) + \frac{\delta}{m}\dot{\theta}_2 + \frac{g}{l}sin(\theta_2) \right\|, |\dot{\theta}_1 - \dot{\theta}_2| \right\} \\
&= max\left\{ \left\| \frac{\delta}{m}(\dot{\theta}_2 - \dot{\theta}_1) + \frac{g}{l}(sin(\theta_2) - sin(\theta_1)) \right\|, |\dot{\theta}_1 - \dot{\theta}_2| \right\} \\
&\leq max\left\{ \frac{\delta}{m}\left\|(\dot{\theta}_2 - \dot{\theta}_1)\right\| + \frac{g}{l}\left\|(sin(\theta_2) - sin(\theta_1))\right\|, |\dot{\theta}_1 - \dot{\theta}_2| \right\} \\
&\text{(Inéglité triangulaire)} \\
&\leq max\left\{ max\{\frac{\delta}{m},\frac{g}{l}\}(\left\|(\dot{\theta}_2 - \dot{\theta}_1)\right\| + \|(sin(\theta_2) - sin(\theta_1))\|), |\dot{\theta}_1 - \dot{\theta}_2| \right\}
\end{aligned}
$$

Puisque $sin(\theta)$ est une fonction monovariable, la norme infinie n'est autre que la valeur absolue. On peut alors appliquer l'inégalité des accroissements finis, c'est à dire :

$$\begin{aligned} |sin(\theta_2) - sin(\theta_1)| &\leq |max(\frac{d}{dt}sin(\theta))||\theta_2 - \theta_1| \\ &\leq |max(cos(\theta))||\theta_2 - \theta_1| \\ &\leq |\theta_2 - \theta_1| \end{aligned}$$

Soit

$$k = max\{1,2max\{\frac{\delta}{m},\frac{g}{l}\}\}$$

Dans le cas des constantes utilisées dans cet exemple, on a :

$k = max(1,2 * max(0.4/0.5,9.8/0.3)) = 65.4$

Annexe D

Calcul de constante lipchitzienne pour l'exemple du chapitre 4

Afin de pouvoir réaliser l'algorithme, nous avons besoin d'une constante k telle que f soit k-Lipchitzienne. La norme adoptée dans ce cas est la norme infinie définie par:

$$v = \begin{pmatrix} v_1 \\ . \\ . \\ . \\ v_{|V|} \end{pmatrix}, |v\| = max(|v_i|)$$

D'abord:

$$K_I(min(w_1) - max(h_1)) \leq \dot{I}_P(t) \leq K_I(max(w_1) - min(h_1))$$
$$\Rightarrow K_I(0 - 0.6) \leq \dot{I}_P(t) \leq K_I(0.6 - 0)$$
$$\Rightarrow -0.6K_I \leq \dot{I}_P(t) \leq 0.6K_I$$
$$\Rightarrow \left\|\dot{I}_P(t)\right\| \leq 0.6K_I$$

Soit:

$$x = \begin{pmatrix} I_P \\ h_1 \\ h_2 \end{pmatrix}, x' = \begin{pmatrix} I'_P \\ h'_1 \\ h'_2 \end{pmatrix}$$

En plus:

$$\begin{aligned} & \left|\dot{I}_P - \dot{I}'_P\right| \\ \leq\ & 2K_I max\{|w_1 - w'_1|, |h_1 - h'_1|\} \\ \leq\ & 2K_I \|x - x'\| \end{aligned}$$

$$\left\|\dot{I}_P - \dot{I}'_P\right\| \leq 2K_I \|x - x'\| \tag{D.1}$$

Par contre:

$$\begin{aligned} \left\|\dot{h}_1 - \dot{h}'_1\right\| &\leq \tfrac{K_P}{A} \|w_1 - w'_1\| + \tfrac{K_P}{A} \|h_1 - h'_1\| + \tfrac{\left\|I_P - I'_P\right\|}{A} \\ &\quad + \tfrac{S\sqrt{2g}(a_Z + a_{max})}{A} \left\|\sqrt{h_1} - \sqrt{h'_1}\right\| \\ &\leq \tfrac{2K_P}{A} \|x - x'\| + 2K_I/A \|x - x'\| + \tfrac{S\sqrt{2g}(a_Z + a_{max})}{A} \left\|\tfrac{h_1 - h'_1}{(\sqrt{h_1} + \sqrt{h'_1})}\right\| \end{aligned}$$

Dans ce cas, il n'est possible d'obtenir une constante k nous limitons h_1 et h_2 au cas ou $h_1 \geq 0.1$,$h_2 \geq 0.1$. Ainsi, la constante Lipchitzienne ne serait valide que dans ce cas précis.

Donc:

$$\begin{aligned}
&\left\|\dot{h}_1 - \dot{h}'_1\right\| \\
&\leq \frac{2K_P + 2K_I}{A} \left\|x - x'\right\| + \frac{S\sqrt{2g}(a_Z + a_{max})}{A} \left\| \frac{h_1 - h'_1}{(\sqrt{h_1} + \sqrt{h'_1})} \right\| \\
&\leq \frac{2K_P + 2K_I}{A} \left\|x - x'\right\| + \frac{S\sqrt{2g}(a_Z + a_{max})}{A} \left\| \frac{h_1 - h'_1}{0.2} \right\| \\
&\leq (\frac{2K_P + 2K_I}{A} + \frac{S\sqrt{2g}(a_Z + a_{max})}{0.2A}) \left\|x - x'\right\| \\
&\leq (\frac{2K_P + 2K_I + 5S\sqrt{2g}(a_Z + a_{max})}{A}) \left\|x - x'\right\|
\end{aligned}$$

C'est à dire:

$$\left\|\dot{h}_1 - \dot{h}'_1\right\| \leq (\frac{2K_P + 2K_I + 5S\sqrt{2g}(a_Z + a_{max})}{A}) \left\|x - x'\right\| \tag{D.2}$$

Enfin:

$$\begin{aligned}
&\left\|\dot{h}_2 - \dot{h}'_2\right\| \\
&\leq \frac{a_Z S\sqrt{2g}}{A} \left\| \sqrt{h_1} - \sqrt{h'_1} \right\| + \frac{(a_Z + a_{max})S\sqrt{2g}}{A} \left\| \sqrt{h_2} - \sqrt{h'_2} \right\| \\
&\leq \frac{a_Z S\sqrt{2g}}{A} \left\| \frac{(h_1 - h'_1)}{(\sqrt{h_1} + \sqrt{h'_1})} \right\| + \frac{(a_Z + a_{max})S\sqrt{2g}}{A} \left\| \frac{(h_2 - h'_2)}{(\sqrt{h_2} + \sqrt{h'_2})} \right\| \\
&\leq \frac{a_Z S\sqrt{2g}}{A} \left\| \frac{(h_1 - h'_1)}{0.2} \right\| + \frac{(a_Z + a_{max})S\sqrt{2g}}{A} \left\| \frac{(h_2 - h'_2)}{0.2} \right\| \\
&\leq \frac{a_Z S\sqrt{2g} + (a_Z + a_{max})S\sqrt{2g}}{0.2A} max\{\left\|h_1 - h'_1\right\|, \left\|h_2 - h'_2\right\|\} \\
&\leq \frac{a_Z S\sqrt{2g} + (a_Z + a_{max})S\sqrt{2g}}{0.2A} \left\|x - x'\right\|
\end{aligned}$$

Soit:

$$\left\|\dot{h}_2 - \dot{h}'_2\right\| \leq \frac{a_Z S\sqrt{2g} + (a_Z + a_{max})S\sqrt{2g}}{0.2A} \left\|x - x'\right\| \tag{D.3}$$

Soient

$$k_1 = 2K_I$$
$$k_2 = (\frac{2K_P + 2K_I + 5S\sqrt{2g}(a_Z + a_{max})}{A})$$
$$k_3 = \frac{a_Z S\sqrt{2g} + (a_Z + a_{max})S\sqrt{2g}}{0.2A}$$

D'après Eq. (D.1), Eq. (D.2), Eq. (D.3), on a:

$$\|\dot{x} - \dot{x}'\| \leq max\{k_1,k_2,k_3\} \|x - x'\| \tag{D.4}$$

Afin de pouvoir appliquer l'algorithme, nous avons besoin d'une bornes supérieure et inférieure de Ip. Elle est donnée par:

$$\begin{aligned}
&\|I_P\| \\
&= \left\| K_I \int_0^t (w_1(t) - h_1(t)) d\tau \right\| \\
&\leq K_I \int_0^t \|(w_1(t) - h_1(t))\| \, d\tau \\
&\leq 0.6 * T * K_I
\end{aligned}$$

D'ou $\|\nabla h\| \leq N$, $\|f(x,u)\| \leq M$, $\|\nabla h(x) - \nabla h(y)\| \leq l \|x - y\|$ avec $N = 1$,$l = 1$,$M = 1$.

Annexe E

Modélisation hybride orientée composants

La spécification des systèmes hybrides modulaires a été proposée dans plusieurs travaux telle que ([38], [49], [3]) et outils (telle que le langage MODELICA et les outils DYMOLA, CHARON). Afin d'assurer la cohérence du modèle global spécifié, ces travaux ont défini et investigué la communication entre composants. Un problème qu'on rencontre de façon recurrente lors de la spécification du modèle global est l'existence de cycles (par exmeple, Un cycle apparait lorsque le modèle global des composants interconnectés fait apparaitre des variables définies de plusieurs façons contradictoires). Plusieurs approches ont été proposés dans la littérature pour vérifier l'absence de cycles du modèle à composants [70]: une approche naïve consiste en la réduction du système en un seul composant et de vérifier s'il y'a une double définition. D'autres approches plus efficaces ont été proposé dans SIMULINK (résolution en temps réel des boucles algébriques avec l'algorithme de Newton) et MODELICA/DYMOLA (recherche des composants fortement connexes avec l'algorithme de Tarjan).

Afin de réaliser une modélisation orientée composants, nous donnons une définition du composant en tant qu'automate hybride qui communique avec son environnement. Afin de préserver de façon maximale les possibilités de description des défaillances, nous utilisons des paramètres et un automate de modes de défaillances pour chaque composant. Ainsi, une défaillance peut désormais être décrite en tant qu'altération locale de fonctionnement d'un ensemble de composants.

E.0.4 Modèle du composant

Definition 27 *Un composant est un uplet* $c.A = (c.Q,c.X,\Sigma,c.A,Inv,c.Y,c.f,c.U,c.E, c.g,c.q_0,c.x_0,c.A_\Phi,c.\Theta)$ *avec:*

- $c.Q,c.X,c.\Sigma,c.q_0,c.x_0,c.A_\Phi,c.\Theta$ *définis comme dans Def. 15.*
- $c.Y$*: est l'espace des variables continues. Soit* $c.y$ *le vecteur associé.* $c.Y$ *est composé de* $c.Y_r$ *(avec le vecteur correspondant* y_r*) et* $c.Y_w$ *(avec le vecteur correspondant* y_w*):* $c.Y = c.Y_r \times c.Y_w$*.*
- $c.inv$*: associé à chaque état* $q \in c.Q$ *une région* $q.inv \subset X \times Y$
- $A = \{q.A\}_{q \in Q}$*: est la collection de partitions de* $\{q.inv, q \in Q\}$*.*
- $c.f = \{c.f_q\}_{q \in Q}$*: associe à chaque état* q *l'ensemble de dérivées possibles* $c.f_q$ *est l'ensemble de fonctions de* $(c.x,c.y,c.\theta) \in c.X \times c.Y \times c.\Theta$*.*
- $c.U$*: espace d'entrées continues pour* $c.A$
- $c.E$*: est l'ensemble de transitions discrètes.* $e = (q,q',\sigma,g,reinit) \in c.E$ *est une transtion caractérisée par une* source q*, une* destination q'*, une garde* g *et un événement déclencheur* $\sigma \in \Sigma$*.* $reinit()$ *est une fonction de réinitialisation de* $c.x$ *dont les arguments sont* $c.x$,$c.y$ *et* $c.\theta$*.*
- $q.g,q \in c.Q$*: est une fonction de partage qui spécifie la façon selon laquelle une variable de partage d'un composant voisin perçoit le fonction-*

nement du composant c. Les arguments de cette fonction sont c.x,c.y et c.θ.

Def. 27 décrit les composants en tant qu'*entités* (ou *agents* dans [3]) qui intéragissent en échangeant les valeurs des variables continues. Un tel paradigme de modélisation permet de modéliser la plupart des systèmes physiques de façon modulaire et exhaustive. En plus, il met en avant le caractère local des défaillances. Ceci permet d'envisager des stratégies de conception, commande et supervision décentralisées.

La construction du modèle global requiert la définition des communications et de l'initialisation. Cette dernière doit être compatible avec les initialisation acceptées par les définitions des composants. Soit Def. 28

Definition 28 *L'architecture d'un système hybride est donnée par* $S = (C, \Leftarrow, \Pi, INTERFACE, \gamma, INIT)$ *avec:*

- $C = \{c_1,...,c_{|C|}\}$ *est l'ensemble de composants du système.*
- $\Leftarrow$ *est la relation d'*association *entre* $c.y_r, c \in C$ *et* $c.y_w, c \in C$.
- Π *est l'automate de commande selon Def. 16.*
- $INTERFACE$ *contient les modules de l'interface comme défini dans Def. 16.*
- γ *est le vecteur d'entrée de l'environnement.*
- $INIT \subset Q \times X$ *est un vecteur contenant les initialisations.* Q *et* X *sont définis par*

$$Q = \prod_{c \in C} c.Q, X = \prod_{c \in C} c.X$$

Def. 28 décrit l'architecture complète du système hybride commandé orienté composants. Cet assemblage est réalisé grâce à la relation $\rightarrow$ qui définit la façon selon laquelle les composants intéragissent pour former la

globalité du système. En plus du système proprement dit, le niveau de commande et les entrées de perturbation sont aussi décrites. Cette définition est aussi valide pour la description de l'architecture globale dans le cas du système hybride de Def. 15.

Def. 29 décrit la façon avec laquelle les variables partagées sont échangées.

Definition 29 *Soient* c_1,$c_2 \in C$, y_1 *une variable partagée en lecture du composant* c_1, y_2 *une variable partagée en écriture de* c_2. *Si* $c_1.y_1 \Rightarrow c_2.y_2$ *alors pour tout instant du temps hybride* (t,n), *on a: La valeur de* $c_1.y_1(t,n)$ *est affectée à* $c_2.y_2(t,n)$; *Autrement dit:* $c_2.y_2(t,n) := c_1.y_1(t,n)$.

E.0.5 Cohérence

La cohérence du modèle développé est une exigence fondamentale lors de la construction du modèle global. Ici, nous présentons certaines règles dont le respect est nécessaire:

1. Les variables réinitialisés lors du franchissement d'une transition doivent résider dans l'invariant de l'état destination. Soit:

$$\forall c \in C, \forall e = (q,q',\sigma,g,reset), reset(g \geq 0 \cap q.inv) \subseteq q'.inv$$

2. Pour $c \in C$,$q \in c.Q$, Les variables dynamiques continues doivent résider dans $q.inv$ jusqu'à franchissement d'une transition autonome (transition avec $\sigma = \epsilon$) ou controllée (lorsque $\sigma \neq \epsilon$).
3. Il n'existe pas de boucles algébriques.
4. Il n'existe pas de boucles de transitions autonomes. Ceci peut être vérifié en parcourant les séquences de transitions autonomes à la recherche de cycles.

Notons que les conditions 3 et 4 peuvent être vérifiées au début de la construction du système hors ligne. Par contre, les conditions 1 et 2 peuvent ne plus être respectées lorsque les modules d'identification/diagnostic modifient le modèle du système après une défaillance. Par la suite, nous supposerons que ces conditions sont toujours vérifiées.

E.0.6 Execution

Quand les conditions de cohérence sont respectées, le modèle orienté composants décrit dans 28 peut être réduit à un modèle hybride simple selon Def. 15). Ainsi, les définitions qui sont données dans 2.3.2.2 restent valides, avec q, x définis comme suit:

$$q = \prod_{c \in C} c.q$$
$$x = \prod_{c \in C} c.x$$

L'exécution du système $S = (C, \Leftarrow, \Pi, INTERFACE, \gamma, INIT)$ est traitée selon Def. 30.

Definition 30 *Une exécution d'un système* $S = (C, \Leftarrow, \Pi, INTERFACE, \gamma, INIT)$ *est une collection* $\chi = (\tau, q, x)$ *avec* τ *une trajectoire dans le temps hybride et* q,x *définis sur cette trajectoire.* $q :< \tau > \rightarrow Q$ *est l'état discret du système durant chaque étape.* $x = \{x_i : i \in < \tau >\}$ *est une collection de fonctions différentiables* $x^i : I_i \rightarrow X$ *avec:*

- $(q(0,0), x(0,0)) = (q[0], x^0(0)) \in INIT$;
- *Pour tout instant hybride* (t,n), $\forall c \in C$, $c.y_w := q(t,n).g(c.x, c.y, c.\theta)$.

- $\forall t \in [\tau_i,\tau_i'[,\ c.\dot{x}^i(t) = f(q(i),c.x^i(t),c.\theta)$ *et* $(c.x^i(t)^T,c.y^i(t)^T)^T \in c.q[i].inv;$
- $\forall i \in < \tau > -\{N\}$, *il y'a un composant* $c \in C$ *et une transition* $e = (q,q',\sigma,a,reset) \in c.E$, *avec* $c.q[i] = q,c.q[i+1] = q'$, $x(\tau_i',i) \in a$ *et* $x(\tau_{i+1}',i+1) = reinit(x(\tau_i',i));$

E.0.7 Exemple

Retournons au système à deux réservoirs de la section 2.2. Nous pouvons considérer le système en tant qu'un ensemble de composants communicants. par exemple, nous posons $S = (C, \Leftarrow, \Pi, INTERFACE, \gamma, INIT)$ avec:

- $C = \{P,T1,V1,V2,T2\}$. Voir par exemple le $T2$ donné dans Fig. E.1.
- $\Leftarrow$ est la relation d'association. Par exmeple, $V1.Q_{rout} \Rightarrow T2.Q_{1in}$ et $T2.h_{2out} \Rightarrow V1.lh2$.
- Π et $INTERFACE$ sont respectivement le contrôleur et interface associés au système.
- $$\gamma = \begin{pmatrix} a_{min} \\ a_{max} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
- $INIT = ((P1low,V1F,V2F,T1low,T2low),(0,0,0))$, avec $x = (Q_P,h_1,h_2)$.

Par la suite, nous présentons les modèles pour $V1$ et $T2$.

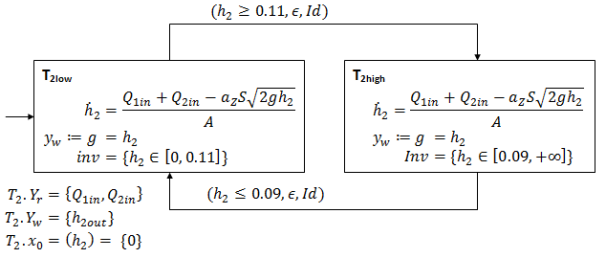

FIG. E.1 – *Le composant* $T2$

Une défaillance partielle possible dans ce cas est de considérer a_{min} comme étant une valeur incertaine dans un intervalle (Par exemple $a_{min} \in [0.1,0.3]$). Étant donnée que les dérivées de h_2 et h_1 dépendent directement de la constante a_{min}, l'évolution dynamique du système est incertaine.

Résumé : Nous proposons une méthodologie de conception pour les systèmes de commande tolérants aux fautes en partant d'un modèle de base exhaustif pour le système complexe à superviser. En pratique, la modélisation exhaustive est réalisée grâce à un automate hybride enrichi par des paramètres quantifiant les défaillances possibles. Ceci permet de modéliser les défaillances partielles. Dans la phase hors ligne, ce système complexe est transformé en un système discret abstrait et exploitable selon des techniques dédiées. Un superviseur est alors construit selon les objectifs de fonctionnement. Lors du fonctionnement du système, l'occurrence d'une défaillance se traduit par l'invalidation de plusieurs comportements dans le modèle abstrait et l'introduction d'incertitudes. Par la suite, les modules de diagnostic et d'identification (qui ne rentrent pas dans l'objet de notre thèse) réduisent de façon progressive le modèle hybride au cours du temps. Afin de pouvoir mettre à jour le modèle discret abstrait, on a développé des algorithmes de calcul d'atteignabilité, de vérification et de génération de régions stabilisées. Pour pouvoir superviser un tel système, l'utilisation de méthodologies d'abstraction est nécessaire afin de transformer le modèle bas niveau exhaustif en un modèle discret approprié. Nous réalisons cette abstraction en proposant des algorithmes qui tiennent compte du contexte d'utilisation (objectifs, contraintes...). Lorsqu'une défaillance est détectée, la reconfiguration est déclenchée en essayant, au fur et à mesure de l'enrichissement du modèle abstrait, de réduire le fonctionnement du système défaillant dans un des schémas prédéfinis.