Modélisation et Intégration du Diagnostic Actif dans une Architecture Embarquée

Elodie Chanthery — Yannick Pencolé

CNRS ; LAAS ; 7 avenue du Colonel Roche, F-31077 Toulouse, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France
[elodie.chanthery ; yannick.pencole]@laas.fr

RÉSUMÉ. *Cet article présente la formalisation du principe de diagnostic actif, et son intégration comme un module fonctionnel dans une architecture embarquée. L'objectif du diagnostic actif est de trouver une séquence d'actions permettant de raffiner le diagnostic sans changer radicalement le plan de mission. On définit le problème comme un problème de planification. Un critère pondéré est explicité. Le but est de trouver le meilleur plan conditionnel permettant au diagnostiqueur d'atteindre une région non ambiguë. Les conflits possibles entre le plan de la mission et le plan pour le diagnostic sont gérés par un contrôleur d'exécution au sein d'une architecture embarquée. Le contrôle d'exécution est spécifié à l'aide de réseaux de Petri.*

ABSTRACT. *This article presents some principles for self-maintenance in an on-board architecture that embeds on-line active diagnosis. The objective of active diagnosis is to find an action plan that refines the diagnosis without radically changing the mission plan. This leads to the definition of a planning problem that relies on an* active diagnoser. *According to a decision criterion, the aim is then to find the best conditional plan to reach a diagnosable region of the active diagnoser. The interactions problems between the result of the planning for diagnosis and the mission planning are then detailed and a solution that uses the execution controller of the on-board architecture is proposed. The execution control is specified with Petri nets.*

MOTS-CLÉS : *diagnostic actif, systèmes à événements discrets, architecture embarquée, contrôle d'exécution*

KEYWORDS: *active diagnosis, discrete-event systems, on-board architecture, execution control*

1. Introduction

L'autonomie décisionnelle d'un système est sa capacité à prendre des décisions et à agir de manière autonome dans un environnement dynamique. Il a pour cela besoin d'estimer en temps-réel son état interne. L'autonomie décisionnelle requiert par conséquent des fonctions de diagnostic en ligne (détection et isolation de fautes) et de planification/replanification en ligne (Chanthery *et al.*, 2005). Ces fonctions doivent être intégrées dans une architecture embarquée.

Le diagnostic en ligne consiste à recevoir un flot d'observations à partir des capteurs et à en déduire une estimation de l'état du système. Le but est de suivre l'évolution temporelle de l'état du système. Cependant, cette tâche est souvent limitée par le nombre réduit d'observations fournies par les capteurs.
Le diagnostic hors ligne se concentre sur la localisation de la faute. Le but est de déterminer l'information à ajouter pour raffiner le diagnostic à moindre coût. Ce problème est à rapprocher du séquencement de tests. Décrit dans (Pattipati *et al.*, 1992) pour des tests binaires, puis étendu dans le cadre de la méthode AGENDA (Olive *et al.*, 2003) pour des tests multi-valués, le problème de séquencement de tests est une procédure hors ligne. Le but est de trouver la meilleure séquence de tests permettant d'isoler la faute, tout en minimisant le coût des tests.

L'objectif de cet article est de présenter une méthode de diagnostic en ligne pour un système autonome en prenant en compte son autonomie opérationnelle, c'est-à-dire la capacité de ce système à agir par lui-même. Une manière d'améliorer la performance du processus de diagnostic en ligne est d'utiliser l'autonomie opérationnelle du système pour effectuer des actions permettant de raffiner le diagnostic en cas d'ambiguïté : c'est ce que l'on appelle le *diagnostic actif*.

Cet article est organisé de la manière suivante. La section 2 présente quelques travaux relatifs au diagnostic actif pour les systèmes dynamiques. La section 3 présente le diagnostic de fautes dans les Systèmes à Événements Discrets (SED). La section 4 propose une caractérisation formelle du diagnostiqueur actif. La section 5 explique comment le diagnostiqueur actif fournit l'entrée à un algorithme de planification. Un algorithme est proposé, et les critères de la planification de mission et de la planification pour le diagnostic sont comparés. Enfin la section 6 présente les points durs à traiter lors de l'intégration du diagnostic actif dans une architecture embarquée et propose une solution à l'aide d'un contrôleur spécifié en réseaux de Petri.

2. Travaux connexes au diagnostic actif

Comme on l'a soulevé lors de l'introduction, le choix de l'ensemble des actions à effectuer en ligne peut être comparé au choix effectué lors d'un problème de séquencement de tests. Cependant, il reste des différences notables. Pour les systèmes dynamiques autonomes ayant pour objectif de réaliser une mission, un des défis de la fonction de diagnostic actif est de proposer une séquence d'actions (ou *plan*) permettant de raffiner le diagnostic sans changer radicalement le plan de la mission. En

d'autres termes, le diagnostic actif doit être intégré dans une architecture embarquée dont le but premier est l'achèvement d'une mission. L'architecture embarquée inclut un planificateur de mission optimisant les actions pour réaliser la mission. L'ensemble des actions proposées par le diagnostic actif peut donc parfois être en conflit avec la réalisation de la mission. Les conflits entre le plan pour le diagnostic et le plan de la mission doivent être gérés. Par ailleurs, les méthodes de séquencement de tests utilisent des tests dont le résultat est binaire (oui/non) ou multi-valué. Dans le cas du diagnostic actif, le résultat d'une action est un nouvel état du diagnostiqueur.

La contribution majeure sur le diagnostic actif des systèmes à événements discrets est le travail de (Sampath *et al.*, 1998). Le diagnostic actif est formulé comme un problème de supervision (Ramadge *et al.*, 1989) où le langage légal est un sous-langage régulier "approprié" du langage régulier du système. Une procédure itérative permet d'obtenir le sous-langage diagnosticable, observable, contrôlable minimal et d'obtenir le superviseur qui synthétise ce langage. La solution proposée est de construire un contrôleur de manière à ce qu'il satisfasse à la fois les objectifs de la commande et que le système contrôlé résultant soit diagnosticable. En d'autres termes, le domaine des actions est restreint de façon à ce que le système résultant soit diagnosticable. Cette approche semble restrictive pour les systèmes autonomes, qui ont besoin de toute leur capacité d'action pour réaliser leur mission. On préférera perdre de manière ponctuelle la propriété de diagnosticabilité plutôt que de ne pas pouvoir achever la mission. Dans cet article, l'idée est donc de combiner la supervision, les capacités de diagnostic et le contrôleur d'exécution, sans pour autant réduire les capacités d'actions du système.

Dans le domaine des systèmes hybrides, (Bayoudh *et al.*, 2008) propose une approche pour réaliser un diagnostic actif dans le cadre des systèmes hybrides. Une méthode d'analyse de diagnosticabilité est utilisée pour déterminer, à partir d'une région non diagnosticable, la séquence d'actions contrôlables à appliquer sur le système pour l'amener vers une région diagnosticable. Étant donné un état incertain du diagnostiqueur actif, l'objectif du diagnostic actif est de trouver un chemin contrôlable menant à un état certain. Les auteurs proposent de résoudre ce problème comme un problème de planification conditionnelle en utilisant un graphe ET-OU. Cependant, l'algorithme n'est pas explicité, notamment le type d'exploration et le critère d'exploration de l'arbre ne sont pas donnés. Nous proposons de donner une définition formelle du diagnostiqueur actif et de discuter le problème de planification plus en détail.

L'intégration de la tâche de diagnostic de défaillance d'actions dans la planification de production a été traitée par (Kuhn *et al.*, 2008) et expérimentée sur des outils d'impression. La méthode s'appuie sur un diagnostiqueur qui met à jour l'état courant du système et stimule le planificateur afin qu'il génère des plans d'actions plus informatifs pour déterminer quelles actions ont été défaillantes au cours de la production tout en maintenant autant que possible l'objectif de production. L'avantage de l'imbrication directe du diagnostic dans la planification réside dans le fait que le diagnostic est confirmé ou infirmé rapidement. L'objectif du diagnostic est ici de déterminer quelles ont été les actions défaillantes sans chercher à expliquer les raisons (physiques ou logicielles) de ces défaillances. Une fois le diagnostic confirmé, une in-

tervention de réparation extérieure rapide est nécessaire afin de maintenir le système de production opérationnel. Contrairement à (Kuhn *et al.*, 2008), notre étude se focalise sur des systèmes dont la finalité est l'autonomie (robots, satellites) face à des pannes physiques et/ou logicielles alors qu'aucune intervention extérieure n'est possible. En conséquence les besoins de diagnostic explicatif sont plus importants car il est crucial de déterminer l'état de santé du système et le mode opérationnel qui en découle afin de savoir si la mission peut effectivement être réalisée en présence de la panne ou si une planification en mode dégradé est nécessaire. Notre approche consiste dans un premier temps à laisser les tâches de diagnostic et de planification séparées, car cela correspond à une implémentation communément utilisée dans les architectures embarquées, dans lesquelles la décision et le suivi de situations sont deux modules fonctionnels distincts.

3. Le diagnostic de fautes dans les Systèmes à Événements Discrets

Définition 1 (Modèle de SED) *Un modèle de SED est un tuple* $G = (X, \Sigma, T, x_0)$ *où :* X *est un ensemble fini d'états ;* Σ *est un ensemble fini d'événements ;* $T \subseteq X \times \Sigma \times X$ *est un ensemble fini de transitions ;* x_0 *est l'état initial du système.*

Un modèle de SED produit un langage régulier clos par préfixe $L(G) \subseteq \Sigma^\star$ où $\Sigma^\star$ est la fermeture de Kleene de Σ. Chaque mot w de $L(G)$ représente une séquence finie d'événements survenant sur le système à partir de l'état initial x_0. L'ensemble $\Sigma_f \subseteq \Sigma$ est l'ensemble des fautes du système. Pour proposer un diagnostic, il faut observer le système à l'aide de capteurs. Les capteurs mettent en œuvre une fonction $Obs : \Sigma \rightarrow (\Sigma_o \cup \{\varepsilon\})$, appelée le *masque des observations* (Jiang *et al.*, 2004) où Σ_o est un ensemble d'événements observables. [1]. Obs associe à n'importe quel événement du système, soit un événement observable de Σ_o, soit l'événement vide ε. Soit $w \in L(G)$, $Obs(w)$ est la trace observable de w et est récursivement définie comme suit :

$$\begin{aligned} Obs(w) &= \varepsilon \text{ si } w = \varepsilon \\ &= Obs(u).Obs(v) \text{ si } w = uv, u \in \Sigma, v \in \Sigma^\star \end{aligned}$$

Définition 2 (Système observé) *Un* système observé *est représenté par une paire* (G, Obs) *où* G *est un modèle de SED et* Obs *est le masque des observations.*

Pour proposer un diagnostic, il faut mettre en œuvre un *moniteur* capable de traiter toutes les observations produites par le système observé, c'est-à-dire n'importe

1. En fonction des capacités des capteurs, l'ensemble des observables Σ_o peut faire partie des événements Σ du système ou non.

quelle séquence observable $Obs(w), \forall w \in L(G)$. Le langage observable de G, noté $Obs(L(G))$, est un sous-ensemble de $\Sigma_o^\star$.
Le problème classique de diagnostic de fautes dans les SED consiste à observer le système et fournir l'ensemble des fautes possibles dont l'occurrence est cohérente avec les observations : étant donné la séquence des observations relevée σ, l'occurrence d'une faute F est cohérente avec σ s'il existe au moins une séquence d'événements w dans le modèle G ($w \in L(G)$) qui contient F et qui vérifie $Obs(w) = \sigma$. Dans ce cas, la séquence observable $Obs(w)$ est une *trace observable* de F.

Définition 3 (Trace observable) *L'ensemble $Traces(F)$ des traces de la faute F est le langage régulier défini comme l'ensemble des séquences observables finies :* $Traces(F) = \{\sigma \in \Sigma_o^\star, \exists w \in L(G), (F \in w) \wedge (Obs(w) = \sigma)\}$.

Étant données les observations σ, une faute F est une solution possible du problème de diagnostic si $\sigma \in Traces(F)$. Il peut y avoir plusieurs solutions car une séquence observable peut être une trace observable de plusieurs fautes. Par extension, on définit aussi les traces de *l'absence de* F, notées $Traces(\neg F)$, qui rassemble les traces observables possibles du système quand F n'est pas arrivé : $Traces(\neg F) = \{\sigma \in \Sigma_o^\star, \exists w \in L(G), (F \notin w) \wedge (Obs(w) = \sigma)\}$.

De ces deux ensembles, on déduit que : si $\sigma \in Traces(F) \setminus Traces(\neg F)$, l'occurrence de F est sure ; si $\sigma \in Traces(\neg F) \setminus Traces(F)$, l'occurrence de F est impossible ; si $\sigma \in Traces(F) \cap Traces(\neg F)$, l'occurrence de F est possible.

L'ensemble des traces d'une faute F est défini comme un langage régulier donc il peut être représenté par une machine à états finie déterministe et minimale (Hopcroft *et al.*, 2001) $M(F) = (S, \Sigma_o, \delta, s_0, etiq)$ où : S est un ensemble fini d'états ; Σ_o l'alphabet de la machine ; $\delta : S \times \Sigma_o \rightarrow S$ est la fonction de transition ; $s_0 \in S$ est l'état initial ; $etiq : S \rightarrow \{F\text{-}possible, F\text{-}impossible\}$. Si l'on note $\delta^\star : S \times \Sigma_o^\star \rightarrow S$ l'extension de δ aux séquences d'événements de Σ_o, toute trace σ de F est telle que $etiq(\delta^\star(s_0, \sigma)) = F\text{-}possible$.

Par analogie, l'ensemble des traces $Traces(\neg F)$ peut aussi être représenté par une machine $M(\neg F)$ (voir (Chanthery *et al.*, 2009) pour le détail). Les deux machines $M(F)$ et $M(\neg F)$ peuvent être utilisées comme un moniteur du système et peuvent donc déterminer à tout moment si la séquence d'observations courante est une trace de F ou non. Le diagnostiqueur classique (Sampath *et al.*, 1996) peut être trivialement caractérisé par le produit synchrone de l'ensemble des machines $M(F), F \in \Sigma_f$ (Pencolé *et al.*, 2006).

4. Diagnostic actif

Cette section étend le cadre classique des SED dans le but de réaliser du *diagnostic actif*. Le diagnostiqueur actif proposé prend en compte le fait qu'à un instant donné, il peut être possible d'agir sur le système et d'obtenir un meilleur diagnostic si une

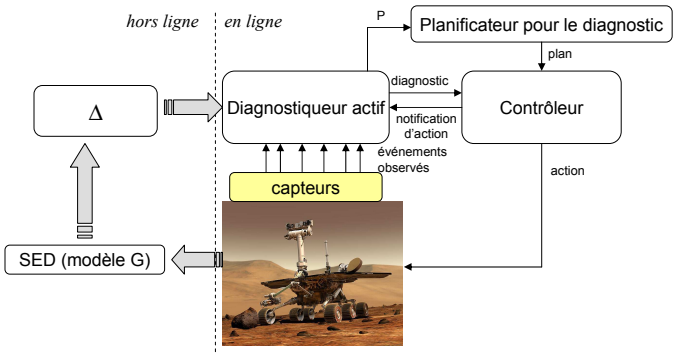

Figure 1. *La partie de l'architecture embarquée relative au diagnostic actif.*

ou plusieurs actions sont exécutées. Comme le montre la figure 1, le but du diagnostiqueur actif modélisé par la machine Δ est premièrement de fournir un diagnostic pour n'importe quelle situation observable (comme les autres diagnostiqueurs) et deuxièmement de fournir des informations sur l'utilité du déclenchement d'une session de diagnostic actif. En outre, il fournit un problème de planification P à un planificateur qui peut planifier des actions pour affiner le diagnostic (voir la section 5).

4.1. *Extension du modèle pour le diagnostic actif*

Du point de vue du modèle du système, une *action* exécutée par le contrôleur est représentée comme un événement contrôlable (Ramadge *et al.*, 1989).

Définition 4 (Action) *Une action est un événement du système qui survient si seulement si le contrôleur du système exécute l'action.*

Dans la suite, on suppose que le contrôleur notifie le diagnostiqueur de l'action exécutée. Cela signifie que n'importe quelle action sera observée par le diagnostiqueur (Figure 1). Soit $\Sigma_a \subseteq \Sigma$ l'ensemble des actions disponibles pour le système observé modélisé par (G, Obs). On peut écrire : $\forall a \in \Sigma_a, (a \in \Sigma) \wedge (Obs(a) = a)$ en d'autres termes $\Sigma_a \subseteq \Sigma_o$.

Finalement, pour s'assurer que le contrôleur est toujours capable d'exécuter une action sur le système, l'étude du problème de diagnostic actif est limitée à la sous-classe de systèmes à événements discrets pour lesquels l'hypothèse suivante tient.

Hypothèse 1 *De n'importe quel état $x \in X$, il est toujours possible d'exécuter une action $a \in \Sigma_a$ après l'occurrence d'une séquence finie d'événements réactifs $e \in \Sigma \setminus \Sigma_a$.*

Si cette hypothèse ne tient pas, le système peut atteindre un état où le contrôleur ne peut plus exécuter d'action. Dans ce cas, le problème de diagnostic actif n'a évidemment aucune solution.

Pour la clarté de l'exposé, la définition du diagnostiqueur actif Δ est décomposée en deux étapes. D'abord, on présente le diagnostiqueur actif $\Delta(F)$ spécialisé pour une faute F donnée, puis on présente le diagnostiqueur actif comme l'union des diagnostiqueurs spécialisés.

4.2. *Diagnostiqueur actif spécialisé*

Le défi du diagnostic actif est d'affiner le diagnostic en élaguant des ambiguïtés sur la présence ou l'absence de fautes à un moment donné en effectuant des actions sur le système. La définition du diagnostiqueur $\Delta(F)$ tient sur les faits suivants : 1- s'il existe un plan d'actions pour diagnostiquer la faute F avec certitude alors ce plan va produire une trace observable qui appartient nécessairement à $Traces(F) \setminus Traces(\neg F)$; 2- s'il existe un plan d'actions pour diagnostiquer l'absence de la faute F avec certitude alors ce plan va produire une trace observable qui appartient nécessairement à $Traces(\neg F) \setminus Traces(F)$.

Soit $M(F) = (S_1, \Sigma_o, \delta_1, s_{01}, etiq_1)$ la machine à états représentant $Traces(F)$ et $M(\neg F) = (S_2, \Sigma_o, \delta_2, s_{02}, etiq_2)$ celle de $Traces(\neg F)$. Le diagnostiqueur actif peut alors être défini comme le résultat de la synchronisation des machines $M(F)$ et $M(\neg F)$ sur les événements observables.

Définition 5 *Le diagnostiqueur actif de F est la machine à état déterministe $\Delta(F) = (S, \Sigma_o, \delta, s_0, etiq)$ où : $S = S_1 \times S_2$ est l'ensemble des états ; Σ_o est l'ensemble des événements observables ; $\delta : S \times \Sigma_o \rightarrow S$ est la fonction de transition : $\forall s_1 \in S_1, s_2 \in S_2, o \in \Sigma_o, \delta((s_1, s_2), o) = (\delta_1(s_1, o), \delta_2(s_2, o))$; $s_0 = (s_{01}, s_{02})$ est l'état final ; la fonction $etiq : S \rightarrow \{F\text{-}sain,\ F\text{-}s\hat{u}r,\ F\text{-}discriminable,\ F\text{-}nondiscriminable,\ nonAdmissible\}$ est définie séquentiellement par les étapes suivantes. $\forall s = (s_1, s_2) \in S$:*

1) si $etiq_1(s_1) = F\text{-}possible$ et $etiq_2(s_2) = \neg F\text{-}impossible$ alors $etiq(s) = F\text{-}s\hat{u}r$;

2) si $etiq_1(s_1) = F\text{-}impossible$ et $etiq_2(s_2) = \neg F\text{-}impossible$ alors $etiq(s) = nonAdmissible$;

3) si $etiq_1(s_1) = F\text{-}impossible$ et $etiq_2(s_2) = \neg F\text{-}possible$ alors $etiq(s) = F\text{-}sain$;

4) si $etiq_1(s_1) = F\text{-}possible$ et $etiq_2(s_2) = \neg F\text{-}possible$, on étudie deux cas : s'il existe une séquence de transitions de s vers s' dans $\Delta(F)$ telle que $etiq(s') = F\text{-}s\hat{u}r$ ou $etiq(s') = F\text{-}sain$, alors $etiq(s) = F\text{-}discriminable$, sinon $etiq(s) = F\text{-}nondiscriminable$.

De cette définition, on peut tirer les résultats suivants. Pour tout état s de $\Delta(F)$:

Théorème 1 *1)* $etiq(s) = F\text{-}s\hat{u}r \equiv$ *pour toute séquence observable* σ *telle que* $\delta^\star(s_0, \sigma) = s$*, pour toute séquence d'événements* w *du système* G *telle que* $Obs(w) = \sigma$*,* $F \in w$ *;*

2) $etiq(s) = F\text{-}sain \equiv$ *pour toute séquence observable* σ *telle que* $\delta^\star(s_0, \sigma) = s$*, pour toute séquence d'événements* w *du système* G *telle que* $Obs(w) = \sigma$*,* $F \notin w$ *;*

3) $etiq(s) = F\text{-}discriminable \equiv$ *pour toute séquence observable* σ *telle que* $\delta^\star(s_0, \sigma) = s$*, pour toute séquence d'événements* w *du système* G *telle que* $Obs(w) = \sigma$*, il existe au moins une séquence finie d'événements observables* $\sigma_{s \to s'}$ *telle que* $\delta^\star(s, \sigma_{s \to s'}) = s'$ *et* $etiq(s') \in \{F\text{-}s\hat{u}r, F\text{-}sain\}$ *;*

4) $etiq(s) = F\text{-}nondiscriminable \equiv$ *pour toute séquence observable* σ *telle que* $\delta^\star(s_0, \sigma) = s$*, pour toute séquence d'événements* w *du système* G *telle que* $Obs(w) = \sigma$*, il n'existe pas de séquence finie d'événements observables* $\sigma_{s \to s'}$ *telle que* $\delta^\star(s, \sigma_{s \to s'}) = s'$ *et* $etiq(s') \in \{F\text{-}s\hat{u}r, F\text{-}sain\}$.

4.3. *Diagnostiqueur actif*

Soit $F_1, \ldots, F_n$ les fautes anticipées, soit $\Delta(F_i) = (S_i, \Sigma_o, \delta_i, s_{0i}, etiq_i)$ le diagnostiqueur actif spécialisé de la faute F_i. Le diagnostiqueur actif Δ de G est alors défini comme l'union des diagnostiqueurs spécialisés définie comme suit. Pour toute séquence d'observations σ, le diagnostiqueur spécialisé $\Delta(F_i)$ atteint l'état s_i avec une étiquette $etiq(s_i)$. Le résultat du diagnostiqueur actif Δ après l'observation de σ est donc : $s = s_1, \ldots, s_n$ et $etiq(s) = \{etiq(s_1), \ldots, etiq(s_n)\}$.

Le diagnostiqueur actif fournit donc les informations suivantes : *le diagnostic courant :* pour n'importe quelle faute F, il fournit le statut courant de la présence de F. Par exemple, si $\forall i \in \{1, \ldots, n\}$, le statut de F_i est sain, cela signifie qu'aucune faute n'est survenue ; *le statut de la session de diagnostic actif :* il indique l'utilité du déclenchement d'une session de diagnostic actif dont l'objectif est de désambiguïser la présence et l'absence de chaque faute F pour lesquelles l'étiquette de s indique que la faute est discriminable.

5. Planifier pour diagnostiquer

5.1. *Quelques notions sur la planification de mission*

Le problème de planification de mission considéré a été décrit dans (Chanthery *et al.*, 2005) et (Meuleau *et al.*, 2009). Un système doit agir de manière autonome pour réaliser un ensemble d'objectifs dans un environnement incertain. Cette réalisation, plus ou moins complète et optimale, est appelée la *mission*. Chaque objectif a une récompense associée qui dépend de l'intérêt que lui porte l'opérateur. Ce problème est connu comme un problème de planification sur-prescrit (Smith, 2004). Dans ce problème, il est impossible de réaliser tous les objectifs. Le planificateur de mission doit choisir un sous-ensemble d'objectifs pouvant être réalisés en respectant des créneaux

temporels (appelés les *contraintes temporelles*) et les limites des ressources tout en maximisant la récompense attendue. Les limitations en ressources (carburant, espace mémoire, etc) sont appelées par la suite *contraintes de ressources*.

5.2. *Formulation du problème de planification pour le diagnostic*

Dans la suite, on notera σ une séquence finie d'événements observables telle que $\delta^\star(s,\sigma) = s'$ et π_σ la séquence d'actions résultant de la projection de σ sur Σ_a. Pour la clarté de l'exposé, la formulation du problème de planification a été décomposée de la même manière que la construction du diagnostiqueur actif.

5.2.1. *Problème de planification pour une faute simple*

Soit s^I un état du diagnostiqueur pour lequel il existe **seulement une** faute F étiquetée F-$discriminable$; les autres fautes F_j sont étiquetées F_j-$s\hat{u}r$ ou F_j-$sain$. L'ensemble des séquences d'actions admissibles pour diagnostiquer F avec certitude est contenu dans $AP_F = \{\pi_\sigma | \exists \sigma, \delta(s^I,\sigma) = s', s'$ est étiqueté F-$s\hat{u}r$ ou F-$sain\}$

Proposition 1 *S'il existe une séquence d'actions pouvant être réalisée par le système et permettant de raffiner le diagnostic, alors cette séquence est dans AP_F.*

Preuve immédiate par le théorème 1.

L'objectif est de formuler un problème de planification classique sous la forme d'un tuple $P = (s_i, S, A, T, But, C)$ où les informations suivantes sont extraites du diagnostiqueur actif : l'état initial s_i est s^I ; $A = \Sigma_a$; S est l'ensemble fini des états de Δ ; $T : S \times A \times S \to \{0,1\}$ est la fonction de transition : $T(s,a,s') = 1$ si s' peut être atteint lorsque a est réalisé dans l'état s, i-e $\pi_{s \to s'} = a$; $T(s,a,s') = 0$ sinon [2] ; l'ensemble des états but $But = \{s' \in S$ tels que s' est étiqueté F-$s\hat{u}r$ ou F-$sain\}$; C un critère à minimiser.

Le problème de planification peut être formulé comme suit : trouver un plan conditionnel dont au moins une des réalisations mène le diagnostiqueur de s_i à un état $s_p \in But$, en minimisant un critère C et respectant des contraintes de ressources. On peut voir le plan conditionnel comme un arbre, dont la racine est s_i, dans lequel les nœuds sont des états (ou des séquences d'états) et les arcs sont des actions (ou des séquences d'actions).

Définition 6 *Un plan est admissible de s_i ssi les contraintes de ressources sont respectées et s'il exécute au moins une séquence d'actions $\{a_{i+1}, a_{i+2}, \ldots, a_p\}$ telle qu'il existe $s_p \in But$ pour lequel $\forall k \in \{i+1, \ldots, p\}, T(s_{k-1}, a_k, s_k) = 1$.*

2. $T(s,a,s') = 0$ en particulier si a est une action qui ne peut pas être réalisée dans s pour des raisons de sécurité ou de contraintes physiques.

5.2.2. *Problème de planification*

Soit s^I un état du diagnostiqueur pour lequel **il existe un sous-ensemble** $D \subseteq \{1, \ldots, n\}$ tel que pour tout i dans D, s^I est étiqueté F_i-$discriminable$ et pour tout j en $\{1, \ldots, n\} \setminus D$, s^I est étiqueté F_j-$s\hat{u}r$ ou F_j-$sain$. Alors l'ensemble des séquences d'actions pour diagnostiquer tous les F_i pour lesquels i est dans D avec certitude est contenu dans $AP = \{\pi_\sigma | \exists \sigma, \delta(s^I, \sigma) = s', \forall i \in D, s'$ est étiqueté F_i-$s\hat{u}r$ ou F_i-$sain\}$. L'ensemble AP peut être l'ensemble vide. Ceci est dû à la propriété suivante.

Proposition 2 *L'ensemble des séquences d'actions admissibles pour diagnostiquer tout F_i pour lequel i est dans D est contenu dans l'intersection des ensembles AP_{F_i}.*

La reformulation en un problème de planification classique est un tuple $P = (s_i, S, A, T, But, C)$ où s_i, S, A, T et C sont les mêmes que pour le problème de planification pour une faute simple, et $But = \{s' \in S$ tels que $\forall i \in \{1, \ldots, n\}$, si s est étiqueté F_i-$discriminable$ alors s' est étiqueté F_i-$s\hat{u}r$ ou F_i-$sain\}$. Le problème de planification reste le même.

5.3. *Algorithme de planification*

La recherche de l'arbre optimal est formulée comme une recherche ordonnée en profondeur d'abord avec élagage dans un arbre ET-OU (Nilsson, 1998) où les nœuds OU correspondent aux états de système et les nœuds ET aux actions. Le choix d'une recherche en profondeur d'abord est justifié par le caractère *au plus tôt* de l'algorithme une fois qu'une première solution a été trouvée. Ainsi, même si toutes les branches de l'arbre ET-OU ne sont pas explorées pour des raisons de temps de calcul, on dispose d'un plan admissible. Le problème se résout par un algorithme AO* (Martelli *et al.*, 1973). Cet algorithme doit être adapté aux particularités du problème, comme la prise en compte de coûts variés, de récompenses, et de contraintes temporelles et de ressources. La solution est un arbre que nous appellerons arbre solution.

5.4. *Critère*

Dans cet article, on formalise C sous la forme d'une somme pondérée. On souhaite prendre en compte (1) les similitudes entre l'arbre solution et le plan de mission : un plan exécutant une séquence d'actions menant le système dans un état compatible avec l'accomplissement de la mission doit être préféré, un plan dont la séquence d'actions mène le système à un état qui contrarie l'accomplissement de la mission doit être écarté ; (2) la criticité des fautes à découvrir ; (3) le coût des actions.

5.4.1. *Coûts et récompenses*

On distingue 3 types d'actions dans l'ensemble A. Les *actions de réparation* : les fautes peuvent être réparables ou non (Cordier *et al.*, 2007). On définit la variable booléenne Φ_F qui vaut 1 si F est réparable, 0 sinon. Chaque faute peut être réparée avec un coût $C_{répare_F} > 0$. Pour une faute F non réparable, $C_{répare_F} = \infty$. Les *actions de replanification* représentent du temps de calcul. Le coût de la replanification à partir d'un état but s' dépend de la similitude entre cet état et les états du plan de mission défini dans la sous-section 5.1. La détermination de l'état s' étant très difficile, on suppose dans ce travail que le coût d'une replanification est constant, égal à $C_{replanifie} > 0$. Les *autres actions*, comme par exemple se déplacer, observer, transmettre des informations, ... sont utiles à l'accomplissement de la mission. Cet ensemble d'actions est noté A_m. Chaque action $a \in A_m$ a un coût $C_a > 0$. Enfin, on note O l'ensemble des objectifs de la mission. Chaque objectif $o \in O$ a une récompense associée $R_o > 0$.

5.4.2. *Critère et contraintes*

L'idée est d'utiliser un critère similaire à celui pour la planification de mission. Soit π_σ^m la séquence d'actions exécutée par le système pour réaliser sa mission et O^m le sous-ensemble d'objectifs qui sont réalisés lors de l'exécution de π_σ^m. Le critère de la planification de mission est :$C^m = \min(\sum_{a \in \pi_\sigma^m} C_a - \sum_{o \in O^m} R_o)$ sous les contraintes temporelles et de ressources.

De la même façon, on définit le critère pour la planification pour le diagnostic, d'un état s^I à un état s' : $C = \min(Coûts - Future_R)$ sous les contraintes temporelles et de ressources, où $Coûts$ représente les coûts d'actions et $Future_R$ représente une évaluation de la récompense future. π_σ étant la séquence d'actions réalisée par le système pour raffiner le diagnostic entre s^I et s', si $s' \notin But$ alors : $Coûts = \sum_{a \in \pi_\sigma} C_a$. Si $s' \in But$, on note D_s le sous-ensemble de fautes F tel que s est étiqueté F-$sûr$ et alors $Coûts = \sum_{a \in A_m^{pi}} C_a + \sum_{F \in D_s} C_{répare_F} + C_{replanifie}$.

L'évaluation de la récompense future étant trop coûteuse en temps de calcul, une première heuristique donnant la récompense maximale possible est utilisée. Soit O^I l'ensemble des objectifs réalisés avant d'arriver en s^I, et O_F l'ensemble des objectifs pouvant être réalisés sachant que la faute F a eu lieu. Alors : $Future_R = \sum_{F \in D_s} (\phi_F \sum_{o \in O \setminus O^I} R_o + (1 - \phi_F) \sum_{o \in O_F \setminus O^I} R_o)$. L'utilisation de la récompense future permet de prendre en compte la criticité de la faute.

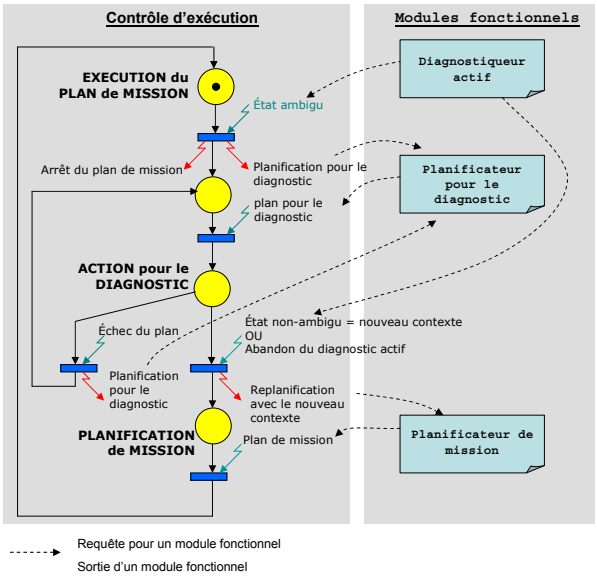

Figure 2. *Gestion des conflits entre la planification pour le diagnostic et la planification de mission*

6. Intégration du diagnostic actif dans une architecture embarquée

Pour résumer, trois modules doivent coopérer au niveau décisionnel dans l'architecture embarquée. Le *diagnostiqueur actif* contrôle le système et détecte des situations où il est utile de déclencher une session de diagnostic actif ; le *planificateur pour le diagnostic* prend comme entrée un problème de planification P donné par le diagnostiqueur actif et calcule un arbre solution pour raffiner le diagnostic. Enfin, le *planificateur de mission* prend comme entrée l'état courant du système et calcule un plan de mission qui réalise un sous-ensemble faisable d'objectifs en satisfaisant les contraintes temporelles et les limites des ressources, tout en maximisant les récompenses attendues. Ces trois tâches informatiques doivent être intégrées dans une architecture embarquée. Pour gérer les conflits entre les différents modules fonctionnels et maintenir une description logique de leurs états, une couche de contrôle d'exécution doit être introduite.

6.1. *L'outil ProCoSA*

Plusieurs outils (Fleury *et al.*, 1997), (Barbier *et al.*, 2006) existent pour concevoir le contrôle d'exécution dans les architectures embarquées de systèmes complexes comme des robots mobiles autonomes ou des satellites. Dans cet article, le contrôle d'exécution sera conçu avec l'outil ProCoSA car il permet de spécifier facilement les

liaisons entre des modules et de gérer leurs interactions grâce à des réseaux de Petri. Dans une architecture conçue avec ProCoSA, les composants de la partie délibérative du système, comme l'algorithme de planification ou le diagnostic actif, sont mis encapsulés dans des modules embarqués. Ces modules sont dynamiquement lancés, interrompus ou (re)paramétrés sur des *requêtes* asynchrones. Quand un traitement est fini, le module répond de manière asynchrone : cette réponse est un *événement*. Hors ligne, ProCoSA permet de modéliser le contrôle d'exécution sous la forme d'un réseau de Petri grâce à une interface graphique. En ligne, il permet de suivre l'évolution du marquage des réseaux de Petri. Dans une architecture conçue avec ProCoSA, un automate, appelé le Joueur de Petri, gère en ligne la mise à jour du marquage des réseaux. Le niveau temps réel de ce langage dépend de l'application. Dans ce travail, l'architecture se comporte comme un système temps-réel souple. En effet, il n'y a aucune garantie sur le temps de réponse, principalement parce que ProCoSA utilise en interne un protocole de communication par sockets et que le traitement des événements dépend de l'état du joueur de Petri. Dans les réseaux de Petri de ProCoSA, on assigne les événements et les requêtes aux transitions : une transition est tirée si elle est validée et qu'un des événements qui lui est assigné arrive. Le tirage de cette transition produit les requêtes assignées. Ces requêtes sont des événements vers d'autres transitions ou des demandes à un module.

6.2. *Gestion des conflits entre la planification pour le diagnostic et la planification de mission*

La solution proposée pour gérer les conflits entre la planification pour le diagnostic et la planification de mission est d'arrêter l'exécution du plan de mission si le diagnostiqueur actif détecte un état ambigu qui doit être désambiguïsé. Suite à cette détection, le contrôleur d'exécution doit ouvrir une session de diagnostic actif, appliquer le plan pour le diagnostic, la réparation si nécessaire, fermer la session de diagnostic actif et lancer une replanification de mission. La replanification est nécessaire car le système et son environnement sont dynamiques : appliquer une action implique une évolution de l'état du système et de son environnement. La solution est illustrée sur la Figure 2. À droite, on trouve les trois modules fonctionnels impliqués : le diagnostiqueur actif, le planificateur pour le diagnostic et le planificateur de mission. L'outil ProCoSA est utilisé pour spécifier les interactions entre ces modules grâce au réseau de Petri à gauche de la figure. Au début, le système exécute sa mission. Quand le diagnostiqueur actif détecte une situation où une session de diagnostic active est nécessaire, il envoie un message "état ambigu". Le contrôleur d'exécution arrête l'exécution de la mission et envoie une requête de planification pour le diagnostic, associé au problème P fourni par le diagnostiqueur actif. Le planificateur pour le diagnostic cherche l'arbre solution et l'envoie au contrôleur. La place "ACTION pour le DIAGNOSTIC" représente l'application de l'arbre solution. Trois cas sont alors possibles : (Cas 1) L'arbre solution réussit, c'est-à-dire que le nœud courant de Δ est dans But : le système est dans un état où toutes les fautes sont étiquetées $F\text{-}sûr$ ou $F\text{-}sain$. Le contrôleur envoie alors une requête au planificateur

de mission pour replanifier avec le nouveau contexte de mission. Le nouveau plan peut inclure des actions de réparation. (Cas 2) Après une action, le diagnostiqueur actif se retrouve dans un état s où une faute est étiquetée $F\text{-}nondiscriminable$. L'arbre solution échoue car il n'y a aucun moyen pour affiner le diagnostic : la session de diagnostic actif doit être fermée. Le contrôleur d'exécution envoie une requête au planificateur de mission pour replanifier avec un contexte de mission dégradé et incertain. (Cas 3) L'arbre solution est réduit à un nombre limité d'actions et après la dernière action, le diagnostiqueur actif est dans un état s où certaines fautes sont encore étiquetées $F\text{-}discriminable$. L'arbre solution échoue mais le diagnostiqueur actif est dans un état où certaines fautes sont toujours étiquetées $F\text{-}discriminable$, le contrôleur d'exécution envoie alors une nouvelle requête de planification pour le diagnostic, associé à un nouveau problème P fourni par diagnostiqueur actif. Quand une requête arrive sur le planificateur de mission, il replanifie la mission et envoie le meilleur plan de mission au contrôleur qui suit son exécution.

7. Conclusion et perspectives

Cet article modélise le diagnostic actif et montre le processus qui mène à sa génération dans le cadre des automates à états finis. Le diagnostic actif ainsi défini permet au système de raffiner son diagnostic sans pour autant réduire son autonomie opérationnelle, ce qui est crucial dans pour des systèmes autonomes réalisant une mission. On spécifie en outre les interactions de ce nouveau processus dans une architecture embarquée à l'aide d'un contrôleur d'exécution.

De nombreux problèmes restent néanmoins en suspens. L'entrée du problème de diagnostic actif est le diagnostiqueur actif. Il est évident que pour des problèmes réels, ce diagnostiqueur doit prendre une forme compacte pour pouvoir être embarqué. Un travail, basé notamment sur les réseaux de Petri colorés, est en cours. L'algorithme de planification devra s'adapter à cette modification et on pourra ainsi réduire les problèmes de complexité de calcul. Au niveau de l'architecture, on souhaite définir le critère pour suspendre l'exécution de plan de mission, un autre pour l'abandon de la session de diagnostic actif si la Situation 3 se répète plus d'une fois : en effet, si le plan pour le diagnostic échoue encore et encore, un critère doit aider le contrôle d'exécution à choisir entre replanifier pour le diagnostic ou abandonner la session de diagnostic actif. Si la session de diagnostic actif échoue, il est toujours possible de planifier la mission sous incertitudes. Par ailleurs, l'algorithme AO* est en cours de développement et soulève des questions, notamment sur la ou les fonctions d'élagage, et sur l'heuristique à employer.

8. Bibliographie

Barbier M., Gabard J.-F., Vizcaino D., Bonnet-Torrès O., « ProCoSA : a software package for autonomous system supervision », *CAR'06*, 2006.

Bayoudh M., Travé-Massuyès L., Olive X., « Towards Active Diagnosis of Hybrid Systems », *DX'08*, 2008.

Chanthery E., Barbier M., Farges J.-L., *Planning, Scheduling and Constraint Satisfaction : from Theory to Practice*, IOS Press, chapter Integration of Mission Planning and Flight Scheduling for Unmanned Aerial Vehicles, 2005.

Chanthery E., Pencolé Y., « Monitoring and Active Diagnosis for Discrete-Event Systems », *7th IFAC Symp. on Fault Detection, Supervision and Safety of Technical Processes*, 2009.

Cordier M.-O., Pencolé Y., Travé-Massuyès L., Vidal T., « Self-healability = Diagnosability + Repairability », *DX'07*, 2007.

Fleury S., Herrb M., Chatila R., « Genom : A Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture », *Intern. Conf. on Intelligent Robots and Systems*, IEEE, p. 842-848, 1997.

Hopcroft J., Motwani R., Ullman J., *Introduction to automata theory, languages, and computation (2nd ed)*, Addison-Wesley, 2001.

Jiang S., Kumar R., « Failure diagnosis of discrete-event systems with linear-time temporal logic specifications », *IEEE Trans. on Automatic Control*, vol. 49, n° 6, p. 934- 945, 2004.

Kuhn L., Price B., de Kleer J., Do M. B., Zhou R., « Pervasive Diagnosis : The Integration of Diagnostic Goals into Production Plans », *AAAI 2008*, p. 1306-1312, 2008.

Martelli A., Montanari U., « Additive AND/OR Graphs », *Proc. of the 3rd Intern. Joint Conf. on Artificial Intelligence*, p. 1-11, 1973.

Meuleau N., Benazera E., Brafman R. I., Hansen E. A., Mausam, « A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains », *Journal of Artificial Intelligence Research*, vol. 34, p. 27-59, 2009.

Nilsson N., *Artificial intelligence, a new synthesis*, 1998.

Olive X., Trave-Massuyes L., Poulard H., « AO* Variant Methods for Automatic Generation of Near-optimal Diagnosis Trees », *DX'03*, 2003.

Pattipati K. R., Dontamsetty M., « On a generalized test sequencing problem », *IEEE trans. on systems, man, and cybernetics*, vol. 22, n° 2, p. 392-396, 1992.

Pencolé Y., Kamenetsky D., Schumann A., « Towards low-cost diagnosis of component-based systems », *6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Process*, 2006.

Ramadge P., Wonham W., « The control of discrete event processes », *IEEE Proc. : Special issue on Discrete Event Systems*, 1989.

Sampath M., Lafortune S., Teneketzis D., « Active diagnosis of discrete-event systems », *IEEE Trans. on Automatic Control*, 1998.

Sampath M., Sengupta R., Lafortune S., Sinnamohideen K., Teneketzis D., « Failure Diagnosis Using Discrete Event Models », *IEEE Trans. on Control Systems Technology*, vol. 4, n° 2, p. 105-124, March, 1996.

Smith D. E., « Choosing Objectives in Over-Subscription Planning », *14th International Conference on Automated Planning and Scheduling*, 2004.