Université Paris 13
École Doctorale de l'Institut Galilée

Diagnostic à base de modèles intégrant les modèles de fautes

Application : Diagnostic du système de freinage dans un véhicule

MÉMOIRE

Soutenu le 1^{er} octobre 2004

Pour l'obtention du

DEA Intelligence Artificielle et Optimisation Combinatoire
Université Paris 13 – Villetaneuse
Spécialité Informatique

Par

Sujeevan ASEERVATHAM

Encadrants :

Philippe DAGUE
Aomar OSMANI

Laboratoire d'Informatique de Paris Nord — UMR 7030

Remerciements

Je souhaite remercier tout particulièrement mes encadrants : Philippe DAGUE et Aomar OSMANI :

- pour m'avoir proposé le stage,
- pour m'avoir accordé de leurs temps lors des différentes étapes du stage,
- pour leurs soutiens et leurs enthousiasmes vis à vis de mon travail.

Un remerciement particulier est adressé à Thierry GAUDRÉ de Renault pour sa gentillesse.

Je tiens, aussi, à exprimer ma gratitude et ma reconnaissance envers mes professeurs et plus particulièrement envers mes professeurs du Laboratoire d'Informatique de Paris Nord et, ainsi, qu'envers Pierre FIGUIÈRE, professeur de chimie à l'université de Cergy-Pontoise.

Résumé

Résumé

La tâche de diagnostic consiste à détecter les anomalies intervenant sur un système puis à expliquer ces erreurs en indiquant les composants pouvant être fautifs. Pour cela, le diagnostic à base de modèles utilise une description du fonctionnement du système. L'erreur est détectée lorsque les données du modèle et les données du systèmes sont incohérentes. Suite à la détection de l'erreur, un ensemble d'explications est généré. Le diagnostic peut, alors, se poursuivre en tentant d'identifier les diagnostics pertinents. L'identification peut alors utiliser les modèles de fautes afin de vérifier la pertinence des diagnostics. Ces modèles de fautes décrivent le comportement du système lorsqu'un ou plusieurs éléments sont fautifs.
Le diagnostic sera appliqué au domaine automobile dans le cadre de ce stage. Ainsi, ce document présente le projet RESEDA, cadre applicatif du diagnostic, l'état de l'art du diagnostic à base de modèles, les apports au diagnostic, notamment dans l'utilisation des modèles de fautes pour l'identification et des fenêtres temporelles pour améliorer la détection, et finalement le document présentera l'implémentation de ces apports au sein d'un environnement de diagnostic.

Abstract

The diagnostic task consists in detecting system's anomalies then in attempting to explain those errors by indicating the components that can be faulty. For this, the model-based diagnostic uses a description of the system's behavior. The error is detected when the model data and the system data are inconsistent. Further to the error's detection, a set of explanations is generated. The diagnosis can, then, continue by identifying the relevant diagnoses. The identification can, then, use fault models in order to ensure the correctness of the diagnoses. Those fault models describe the system's behavior when one or more parts of the system are faulty.
The diagnosis will be applied to the field of automotive within this training. Thus, this document presents the RESEDA project, the diagnostic's application field, the state of the art of the model-based diagnosis, the contribution to the diagnostic's process, especially in the use of fault models for the identification's step and of time windows to improve the detection's step, and finally the document will present the implementation of these contributions within the diagnostic framework.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Chapitre 1

Introduction

Face à la complexité des systèmes et à la miniaturisation des composants, les tâches de détection de panne et la réparation sont devenues des processus du service après-vente de plus en plus coûteux.
Dans ce cadre, le diagnostic est un processus intéressant de plus en plus les industriels. En effet, un processus de diagnostic peut permettre d'aider le réparateur en lui indiquant le symptôme (le conflit) et en lui fournissant une liste de composants pouvant expliquer le symptôme. Ainsi, cela résulte en un gain de temps qui se répercute par une diminution du coût de la réparation.

La tâche de diagnostic est constituée de plusieurs étapes dont la première consiste à surveiller le système et à détecter toute anomalie. Une fois, l'anomalie détectée, l'étape suivante est d'expliquer la défaillance en fournissant des diagnostics (des listes de composants pouvant être à l'origine de l'erreur). L'étape suivante est une identification plus précise de l'erreur qui aboutit à une réduction du nombre de diagnostics.

Dans le cadre de ce stage, nous nous intéresserons, essentiellement, au diagnostic à base de modèles. Le diagnostic à base de modèles repose sur l'utilisation d'un modèle du système physique. Ce modèle est décrit à l'aide d'un langage de formalisme adapté au besoin de l'application.
Ainsi, lorsque les données du modèle différent de celles du système physique, une anomalie est détectée.
L'erreur détectée, il s'agit ensuite de l'expliquer en fournissant un espace de diagnostic (un diagnostic étant un ensemble de composant fautif).
Enfin, la dernière partie, sur laquelle nous nous concentrerons, consiste à identifier les diagnostics les plus pertinents. Pour cela, nous utiliserons des modèles de fautes. A l'instar du modèle de bon fonctionnement, sur lequel se base la détection des erreurs, les modèles de fautes sont décrits en utilisant un langage de formalisme. Ces modèles matérialisent le système lors d'un comportement incorrect.

De plus, nous étudierons les techniques de diagnostic dans le cadre applicatif du projet RESEDA. Ce projet consiste en la mise au point d'un environnement de diagnostic évolutif permettant de diagnostiquer un véhicule Renault.
Cette application nous amènera à intégrer des informations supplémentaires spécifiques

au véhicule Renault.

Ainsi dans le cadre de ce rapport, nous présenterons le projet RESEDA suivi de l'état de l'art sur le diagnostic à base de modèles. Puis, nous continuerons par exposer les différentes techniques que nous avons mises au point concernant le diagnostic dans le cadre de ce projet et nous terminerons par la présentation de l'implémentation de ces techniques sous la forme d'un framework (environnement de diagnostic).

Chapitre 2

RESEDA

Le projet RESEDA est un projet du Réseau National des Technologies Logicielles et signifie "Réalisation d'un Environnement Système Évolutif pour le Diagnostic Automobile".

2.1 Problématique

Les véhicules modernes proposent, aujourd'hui, de plus en plus de services aussi bien au niveau de la sécurité que du confort. Ces services sont obtenus par des moyens combinant l'électronique, l'informatique et les systèmes mécaniques. Ainsi, le véhicule est devenu un véritable concentré de technologies de pointes.

En effet, les véhicules disposent de plusieurs calculateurs et d'organes électroniques chargés d'assister le conducteur faisant, ainsi, office d'interface entre l'homme et les organes mécaniques. Ces composants sont reliés entre eux par des bus multiplexés (CAN, VAN, ...), éliminant, ainsi, une multitude de fils électriques qui encombrée, jadis, le circuit électrique. Néanmoins, l'utilisation de ce bus fait apparaître un réseau complexe semblable à un réseau informatique où les ordinateurs seraient remplacés par des petits composants électroniques. Cette complexité devient plus visible lorsqu'une erreur apparaît. Corriger une erreur devient, alors, un parcours du combattant nécessitant la mise en place d'outils innovant d'identification et de localisation des erreurs.

De plus, l'arrivée de ces technologies, dans les véhicules, a bouleversé le secteur de l'après-vente dans le domaine de la réparation. En effet, la réparation au niveau du garagiste nécessite une connaissance, plus ou moins poussée, de l'informatique et de l'électronique. Il devient, ainsi, impérative au constructeur de proposer une panoplie d'outils informatiques d'aide au diagnostic et à la localisation de pannes. En outre, l'utilisation d'outils simples peut permettre « grossièrement » de localiser un ensemble de composants supposés défectueux. La réparation consistera alors à tout remplacer, dès lors que le garagiste n'a pas les compétences suffisantes pour affiner le diagnostic. La conséquence est inévitablement une augmentation du coût de la réparation. L'une des solutions consisterait alors à capitaliser les connaissances des pannes au fur et à mesure des interventions et de les intégrer aux outils de diagnostics. Ceci implique que les outils de diagnostics doivent être évolutifs.

Par ailleurs, l'un des problèmes majeurs rencontré par le service après-vente est bien souvent la difficulté qu'a le conducteur pour décrire les pannes constatées. En effet, les constatations, dans la majorité des cas, se résument à « tel prestation ne fonctionne pas ». Il est, alors, indispensable de passer le véhicule « au peigne fin » en effectuant une batterie de test. La solution serait d'embarquer un système de diagnostics qui suivrait l'évolution du véhicule en temps réel.

Pour résumer, le constructeur doit pouvoir :

- Garantir la sécurité, en détectant les mauvais comportements des composants.
- Minimiser les coûts de réparation.
- Localiser précisément les défaillances.
- Améliorer le système de diagnostics au fur et à mesure des retours d'expériences.

2.2 Objectifs

En tenant compte de ces différentes observations, le projet consistera à développer une infrastructure de diagnostic embarquée, qui permettra de suivre le comportement du véhicule, et une infrastructure débarquée, qui permettra d'améliorer en affinant le diagnostic généré par les outils embarqués. Les outils embarqués devront être évolutifs pour exploiter au mieux les connaissances acquise par retour d'expériences.

Ainsi, le projet RESEDA se propose trois objectifs majeurs :

1. Définir l'architecture d'un environnement permettant le déploiement et l'exécution d'outils de diagnostics (Diaglet) évolutifs au seins du véhicule. Le langage orienté objet java a été sélectionné pour le développement de l'environnement. Ce langage permettra la réalisation des diaglets en utilisant une architecture de code mobile et d'objets distribués.
2. Réaliser un démonstrateur qui permettra de valider l'infrastructure réalisée et les techniques de diagnostics. La validation se fera à partir de scénarios qui devront être prédéfinis.
3. Élaborer des méthodes formelles de diagnostics exploitant, non seulement, les connaissances acquises lors des retours d'expériences mais aussi les observations effectuées en temps réels par la partie embarquée. De plus, ces méthodes devront alimenter la base de connaissances en fonction des défaillances détectées en temps réel.

2.3 Partenaires

Comme tous projets RNTL, RESEDA relie plusieurs partenaires privés et des laboratoires de recherche publique. Ainsi, les différents acteurs de ce projet sont :

- Renault : Chargé de superviser le projet, de fournir les éléments matériels et de valider l'infrastructure finale.

- Trialog : Chargé d'effectuer les spécifications relatives à l'environnement matériels et logiciels pour le projet.
- Silicomp : Chargé de développer l'infrastructure logicielle nécessaire au projet.
- LIPN : Chargé de définir les méthodes formelles de diagnostics et de réaliser le développement des outils de diagnostics.

2.4 Démarches

La société Renault est chargée de fournir le support matériel, sur lequel le projet devra être validé. Ainsi, le support matériel sera un environnement physique et/ou logiciel capable de simuler le comportement d'un véhicule. Il s'agira dans un premier temps d'un modèle logiciel s'exécutant sur un PC standard et simulant toutes les opérations du circuit électronique du véhicule et notamment ses interactions avec l'environnement extérieur à savoir les composants mécaniques et les interfaces utilisateurs tel que le tableau de bord.

Les communications entres les différents composants s'effectueront, comme dans la réalité, à l'aide de messages circulant sur un bus multiplexé et plus précisément sur le bus CAN. Le bus CAN pourra être étendu, hors PC, pour connecter des cartes électroniques via le port série ou USB du PC.

Cette dernière fonctionnalité est indispensable pour le projet étant donnée que l'environnement de diagnostic se situera sur une carte électronique qui sera reliée au circuit véhicule à travers le bus CAN.

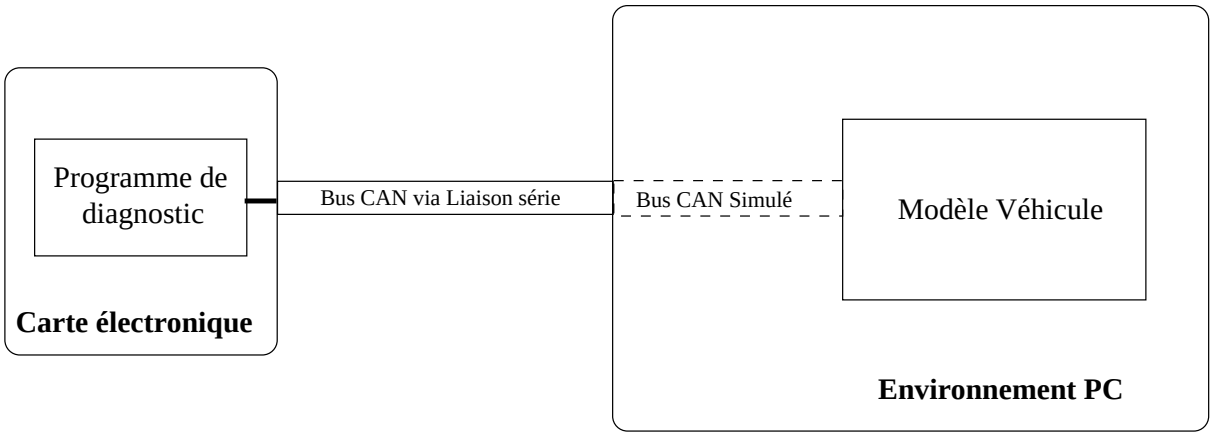

FIG. 2.1: Architecture générale

De plus, Renault doit fournir toutes les informations relatives au véhicule à diagnostiquer afin de concevoir le logiciel de diagnostic. Ainsi, les comportements de chacun des composants à diagnostiquer devront être formellement décrits ou du moins être renseignés afin d'établir, dans un premier temps, leurs comportements internes puis leurs interactions avec le reste du système. Il est à noter que par description formelle du comportement d'un composant, nous entendons, de préférence, une équation analytique reliant les entrées du

composant à ses sorties et par le terme « renseignés » toutes informations susceptibles d'aboutir à l'établissement de cette équation analytique.

Finalement, Renault devra tester et valider un démonstrateur, à partir de scénarios d'utilisation du système de diagnostics du véhicule élaborés conjointement avec les autres partenaires du projet. Ce démonstrateur matérialisera l'achèvement du projet RESEDA et dans le cas où le démonstrateur se montrerait à la hauteur des exigences de Renault : le projet pourrait être approfondi en vue d'une mise en production sur véhicules en séries.

La société Trialog devra définir la spécification de l'environnement matériel et logiciel pour le projet. Une analyse des besoins en ressources mémoire et temps de calculs devrait permettre d'établir un environnement, le plus adéquat possible, afin d'accueillir les logiciels de diagnostics.
De plus, un protocole et une interface devront être spécifiés afin de permettre l'interconnexion entre le bus CAN et l'environnement de diagnostic d'une part et entre l'environnement et les logiciels de diagnostics d'autre part.

En outre, la spécification de l'environnement devra tenir compte du coté évolutif du système car l'un des points principaux de ce projet est, bel et bien, son caractère évolutif. Durant le cycle de vie d'une voiture, les connaissances acquises, par retour d'expériences notamment auprès des services après-vente lors des constatations de pannes par les clients, permettent d'étudier et d'identifier les types de pannes ainsi que leurs symptômes. Il serait donc tout à fait naturel d'envisager d'exploiter ces connaissances « en temps réel » en avertissant, en amont, le conducteur d'une éventuelle défaillance, ainsi que de sa gravité et en indiquant, en aval, les composants défectueux au garagiste afin d'optimiser le temps de réparation tout en réduisant les coûts.
Ainsi, le système de diagnostic doit pouvoir évoluer en fonction du temps au fur et à mesure des connaissances acquises sur les types de pannes. Il s'agit essentiellement pour l'environnement d'accueillir de nouveaux logiciels de diagnostics pouvant être coopératifs.

En plus des spécifications, Trialog devra conjointement avec Renault mettre en place des scénarios de tests, pour le démonstrateur, simulant l'utilisation d'un véhicule par un conducteur et permettant de mettre en avant les capacités du système de diagnostics.

La société Silicomp sera chargée de l'étude et de la réalisation de la plate forme matériel ainsi que de l'environnement logiciel spécifiés par Trialog. En plus des spécifications de Trialog, Silicomp devra tenir compte des exigences du constructeur Renault. Ces exigences peuvent se résumer en deux points principaux :

- Le volume : la carte électronique, plate forme matériel de RESEDA, doit être la plus petite possible afin de ne pas encombrer la plate forme électronique du véhicule.
- Le coût : la carte doit avoir un coût de revient très faible de l'ordre de l'euro afin de ne pas influer sur le prix du véhicule.

Afin de satisfaire ces exigences, la puissance de calcul sera limitée au strict minimum : le calcul des virgules flottantes ne sera même pas pris en charge par le processeur. Ainsi, ces calculs seront effectués par des algorithmes logiciels quelque peu gourmands en temps de calcul. Cela implique que les logiciels de diagnostics devront être optimisés aussi bien

au niveau mémoire qu'au niveau calcul.

Le Laboratoire d'Informatique de Paris Nord (LIPN) aura pour tâche la recherche et le développement d'outils de diagnostics. L'objectif sera essentiellement de bien comprendre la structure matérielle du véhicule à diagnostiquer, d'en extraire les différents composants clés et d'établir les équations analytiques régissant le comportement et les interactions de ces composants. Dans une seconde étape, une étude de diagnosticabilité devra être effectuée afin de différencier les composants pouvant être diagnostiqués de ceux ne pouvant l'être et dans le cas échéant il faudra prévoir d'identifier les causes et les moyens d'y remédier. Puis, il s'agira de mettre en place des méthodes formelles de diagnostics pouvant aboutir à un bon niveau de génération de candidats (ensemble de composants supposés fautifs) expliquant la défaillance observée.

La dernière étape consistera en la réalisation du logiciel implémentant la méthode de diagnostic et devant être embarquée sur le support électronique prévu par Silicomp. Ainsi, le code devra respecter la spécification définie par Trialog tout en respectant les contraintes matériels et logiciels émanant de l'environnement de diagnostics réalisé par Silicomp.

2.5 Stage

La première phase du stage est l'étude et la réalisation d'un moteur de diagnostic utilisant les techniques de diagnostics issues de l'intelligence artificielle et intégrant les modèles fautes.

Les modèles de fautes permettent de décrire les différents comportements pour chacun des composants, à diagnostiquer, et pour chaque type de panne pouvant l'affecter. Cette information permet alors d'étudier la pertinence d'un composant présumé fautif et de l'éliminer de l'espace des candidats possibles si l'hypothèse de sa défaillance se révèle non pertinente.

De plus, dans le cadre de ce projet, certaines variables, issues des calculateurs, ont été jugées « intéressantes » pour l'environnement externe au calculateur et circulent, par conséquent, sur le bus multiplexé CAN. Cela permet à tout programme d'avoir accès librement à ces variables. Néanmoins, d'autres variables « moins intéressantes » ne circulent pas sur le bus mais peuvent être interrogées sur requête. Cependant, cette interrogation à un coût temporel. Ainsi, l'interrogation de ces variables ne doit avoir lieu qu'à certain instants « clés » de l'étape de diagnostic et par parcimonie.
La deuxième phase sera alors d'étudier l'utilisation des variables interrogeables et de son intégration au sein du moteur de diagnostic.

Le stage devra aboutir sur la réalisation d'un environnement de diagnostics (framework) permettant l'expérimentation de différentes techniques de diagnostics. Cet environnement devra être réalisé en java en vue d'une intégration au sein de l'environnement

de diagnostic embarqué du projet.
De plus l'environnement devra assurer les fonctionnalités suivantes :

- Le framework doit pouvoir être intégré à l'environnement de RESEDA.
- La définition ou description du modèle à diagnostiquer doit être simple et intuitive pour un non développeur sans avoir besoin de programmer ou de recompiler.
- Les séquences de méthodes de diagnostics doivent être paramétrables.
- De nouvelles méthodes de diagnostics doivent pouvoir être ajoutées au framework sans grande difficulté.

Chapitre 3

Diagnostic

De nos jours, les industriels s'intéressent de plus en plus au cycle de vie des produits, se rendant compte que la construction n'est que le point de départ de la chaîne commerciale. Ainsi, le concept de service après-vente est apparu récemment. Ce service devant accompagner le client tout au long de la vie du produit. Plus qu'un simple argument de vente, il répond à une réelle nécessité. En effet, à la sortie de l'usine, aucun produit n'est immunisé contre les défauts de fabrication.

De plus, avec l'apparition de systèmes complexes issus de domaines très variés tel que la mécanique, l'informatique, la télécommunication et l'électronique, les constructeurs ont observé une augmentation du coût des réparations. Cette augmentation est provoquée par la difficulté rencontrée par les techniciens de localiser précisément les composants défectueux. La conséquence est, que trop souvent, des composants, de plus en plus onéreux, en bon état de fonctionnement sont remplacés.

Ainsi, un besoin s'est fait sentir au niveau d'outils de diagnostic. Ce besoin a crée une dynamique active de recherche dans le domaine du diagnostic.

3.1 État de l'art du diagnostic

Pour répondre au besoin d'outils automatiques de diagnostic, la recherche, essentiellement en informatique et automatique, a établi des théories de base qui servent de références pour les différentes approches tentant d'apporter une contribution au problème du diagnostic.

3.1.1 Définition

Système

Nous utiliserons le terme « système » pour désigner la partie physique ou logique qui est la cible ou sujet du diagnostic. Le système peut être un système physique tel un circuit électronique ou un réseau, ou un système logique tel un programme informatique. Quelque soit le système, cible du diagnostic, la théorie du diagnostic reste identique, étant donné qu'elle a été établie de façon générale.

De plus, les composants du système, du moins ceux à diagnostiquer, doivent être formellement identifiés au niveau du système.

Modèle

Le terme « modèle » désigne la description théorique d'un système. Le modèle permet de décrire de manière formelle le comportement d'un système. Le terme « modèle » est synonyme de description du système.

Tâche de diagnostic

La tâche de diagnostic consiste à détecter puis à expliquer les différences constatées entre le comportement obtenu et le comportement prévu du système à diagnostiquer.
Le raisonnement sur la tâche de diagnostic, comme indiqué dans [de Kleer and Williams, 1987], consiste à assigner un crédit ou un blâme à un composant ou un ensemble de composants faisant partie du système à diagnostiquer.

Les objectifs du diagnostic peuvent être, selon [de Kleer and Williams, 1987] :

- La détection et l'explication des défaillances. Dans ce cas, le comportement prévu est supposé prédéfini et exempt d'erreurs. Ainsi, toutes différences observées incrimineront le système.
- La modélisation d'un système. Il s'agit, alors, d'établir un modèle comportemental du système. Le système est considéré comme fonctionnant correctement. Ainsi, toutes différences impliqueront une erreur au niveau du modèle.

Conflit

Le terme « conflit » désignera une différence entre un comportement observé et un comportement prédit ou attendu. La tâche de diagnostic tentera alors d'expliquer le conflit.
Le conflit pourra aussi désigner un ensemble de composants du système contenant au moins un composant fautif.

Diagnostic

D'après [Reiter, 1987], un diagnostic est un ensemble de composants supposés fautifs expliquant un conflit.

3.1.2 Diagnostic à base de modèles

Le diagnostic à base de modèles s'appuie sur une description formelle du système à diagnostiquer afin de prédire le comportement du système. Le modèle est définie dans un langage de formalisme compréhensible par le programme de diagnostic. Il définit le comportement du système en décrivant les comportements internes des composants, ainsi, que leurs interactions au niveau du système.
Le modèle est utilisé pour déterminer le comportement attendu du système partant d'une observation. Une observation étant un ensemble de données, issues en général de capteurs. Ces observations sont reliées à des variables du modèle. Le conflit est détecté lorsque des variables calculées par le modèle, à partir d'une observation, diffèrent de celles observées sur le système. Autrement dit, un conflit apparaît lorsqu'une observation et le modèle sont inconsistants.

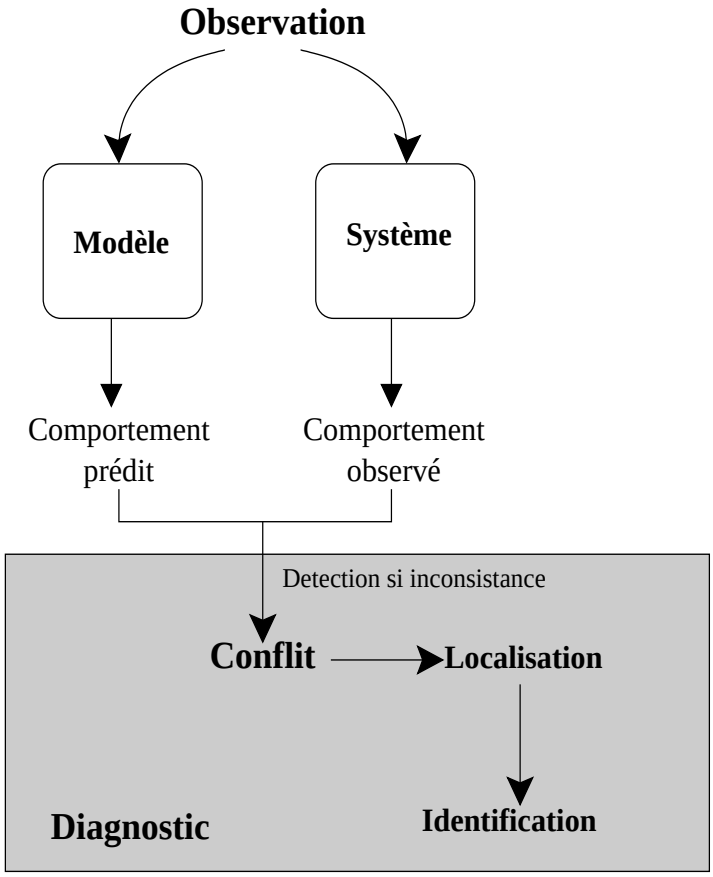

FIG. 3.1: Schéma du diagnostic à base de modèles

Le conflit détecté, la tâche de diagnostic consiste à expliquer la défaillance en proposant des diagnostics, à savoir des ensembles de composants fautifs, expliquant la défaillance.

Dans le domaine du diagnostic à base de modèles, il existe principalement deux approches. Ces approches proviennent de deux domaines scientifiques différents. L'une étant le domaine du contrôle automatique et l'autre étant le domaine de l'informatique et plus précisément de l'intelligence artificielle.

Approche FDI

FDI, Fault Detection and Isolation, est une communauté appartenant au domaine du contrôle automatique. Son objectif est d'utiliser le domaine de l'automatique afin de proposer des méthodes de diagnostic efficaces et optimisées en matière de temps de calculs pour une application à l'industrie.

Le modèle est décrit par des relations de redondances analytiques (RRA ou ARR en anglais).

Une relation de redondance analytique, selon [Cordier et al., 2000], est définie comme étant une relation ne contenant que des variables observables, dont les valeurs appartiennent à l'observation, et pouvant être calculée à partir de l'observation. Le résultat de l'évaluation de la relation est appelé résidu. Un résidu non nul indique un conflit. Le conflit est, alors, l'ensemble des composants dont les comportements ont été utilisés pour établir la RRA. L'ensemble des composants intervenant dans une RRA est appelé support de cette RRA.
Ainsi, lorsque le résidu d'une RRA est non nul : une différence entre le comportement prédit et le comportement observé est mise à jour. Le conflit est, alors, le support de la RRA dont le résidu est non nul. La signification de ce résidu est qu'il existe au moins un composant fautif parmi les composants intervenant dans la RRA.

L'étape de détection de fautes consiste simplement à évaluer les RRA. Un résidu non nul suffit alors pour détecter une défaillance. Cette étape permet, en cas de défaillance, d'identifier les conflits qui sont les supports des RRA non satisfaites (résidus non nuls).

Exemple 1 *Prenons l'exemple d'un additionneur complet composé de deux portes « et » logiques (A1 et A2), de deux portes « ou-exclusif » et d'une porte « ou »tel que défini dans [Reiter, 1987] :*

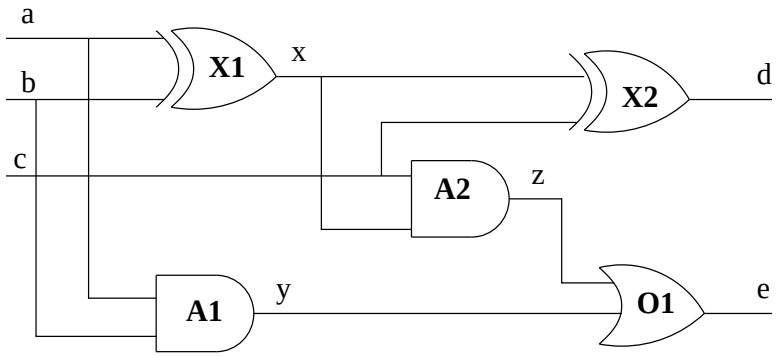

FIG. 3.2: Additionneur complet.

Les variables observables sont $\{a, b, c, d, e\}$. *Le modèle de l'additionneur complet peut être défini par les RRA suivants :*

- RRA_1 *:* $d \oplus (c \oplus (a \oplus b)) = 0$.
- RRA_2 *:* $e \oplus ((a \cdot b) + (c \cdot (a \oplus b))) = 0$.
- RRA_3 *:* $(e \cdot (c \cdot (a \oplus b))) \oplus (c \cdot (a \oplus b)) = 0$.

Les supports pour chacun des RRA étant :

- $supp(RRA_1) = \{X1, X2\}$.
- $supp(RRA_2) = \{X1, A1, A2, O1\}$.
- $supp(RRA_3) = \{X1, A2, O1\}$.

Supposons que les capteurs du système à un instant donné, nous fournissent les sorties $1, 0$ *pour les entrées suivantes* $1, 0, 1$. *L'observation sera ainsi représentée : obs* $= \{a =$

$1, b = 0, c = 1, d = 1, e = 0\}$.
Pour cette observation, la RRA_1, la RRA_2 et la RRA_3 ne sont pas satisfaites, leurs résidus valant, chacun, 1*. Cette observation nous indique donc que le système est défaillant.*

Une fois la détection effectuée, l'étape suivante est la localisation des fautes. Pour cela, la notion de signature doit être introduite. La signature d'une faute F_J (J étant un ensemble de composants du système tel que $\forall c \in J, AB(c)$, le prédicat $AB(c)$ indiquant une défaillance sur le composant c) est représentée par un vecteur colonne S_{F_J} tel que :

$$S_{F_J}(i) = \begin{cases} 1 & si\ \exists c \in J \mid c \in supp(RRA_i) \\ 0 & sinon \end{cases}$$

Les signatures des fautes à détecter seront disposées sous forme matricielle afin d'optimiser les calculs. Pour l'additionneur de la figure 3.2, nous obtenons la matrice du tableau 3.1, appelée matrice de signatures ou matrice d'incidences.

	$S_{F_{\{X1\}}}$	$S_{F_{\{X2\}}}$	$S_{F_{\{A1\}}}$	$S_{F_{\{A2\}}}$	$S_{F_{\{O1\}}}$
RRA_1	1	1	0	0	0
RRA_2	1	0	1	1	1
RRA_3	1	0	0	1	1

TAB. 3.1: Matrice de signatures pour l'exemple de l'additionneur complet

Il est possible d'étendre cette matrice aux fautes multiples tel que :

$$S_{F_{\{x_1,\cdots,x_n\}}} = \begin{pmatrix} S_{F_{\{x_1\}}}(1) \vee \cdots \vee S_{F_{\{x_n\}}}(1) \\ \vdots \\ S_{F_{\{x_1\}}}(k) \vee \cdots \vee S_{F_{\{x_n\}}}(k) \end{pmatrix}$$

La signature d'une observation *obs* est, quant à elle, définie par la formule :

$$S_{obs} = \begin{pmatrix} 0 & si\ RRA_1\ satisfaite\ par\ obs\ ou\ 1\ sinon \\ & \vdots \\ 0 & si\ RRA_k\ satisfaite\ par\ obs\ ou\ 1\ sinon \end{pmatrix}$$

La localisation des fautes se fait alors en comparant la signature de l'observation représentée par le vecteur S_{obs} à la matrice de signatures. Un ensemble de fautes $\mathcal{F}$ pouvant expliquer la défaillance est alors déterminé :

$$\forall F_J \in \mathcal{F}, S_{obs} = S_{F_J}$$

Il est important de noter que cette méthode utilise deux hypothèses [Cordier et al., 2000] :

- L'hypothèse de non compensation : un composant fautif devra laisser apparaître son comportement défectueux. Cela implique que son comportement ne pourra être masqué par le comportement d'un ou plusieurs autres composants. Ainsi, un ou plusieurs composants fautifs entraîneront la non satisfaction des RRA dans lesquelles ils sont engagés. Cette hypothèse permet d'introduire l'hypothèse qui suit.
- L'hypothèse d'exonération : les composants appartenant aux supports d'une RRA satisfaite (résidu nul) sont exonérés. Ces composants sont considérés comme fonctionnant correctement et ne pourront pas apparaître dans l'espace de diagnostic.

Exemple 2 *(Suite de l'exemple 1) En prenant l'exemple précédent avec obs* $= \{a = 1, b = 0, c = 1, d = 1, e = 0\}$*, nous obtenons la signature suivante :*

$$S_{obs} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

L'espace de diagnostic contient alors une faute simple et deux fautes doubles si nous nous limitons aux fautes doubles :

$$\mathcal{F} = \{\{X1\}, \{X2, A2\}, \{X2, O1\}\}$$

Approche DX

La communauté DX [1] est issue du domaine de l'intelligence artificielle. La particularité de cette approche est l'importance qu'elle accorde à la partie explication de la défaillance. Il s'agit de fournir les meilleurs explications (diagnostics) dans un cadre général.

De même que pour l'approche FDI, l'approche DX utilise le diagnostic à base de modèles. Ainsi, le modèle est décrit dans un langage de formalisme le plus adéquat à l'application du diagnostic. De ce fait, la logique du premier ordre est utilisée dans [Reiter, 1987], un langage de contraintes dans [de Kleer and Williams, 1987] ou encore des relations de redondances analytiques dans [Cordier et al., 2000].

Étant donné, une description du système SD, l'ensemble $\mathcal{C}$ des composants du système, un prédicat AB indiquant le mauvais fonctionnant d'un comportement et une observation OBS, le problème de détection d'erreur peut être formalisé par l'expression [Reiter, 1987] :

$$SD \cup OBS \cup \{\forall c \in \mathcal{C} \mid \neg AB(c)\}$$

Si l'expression est inconsistante alors une erreur a été détectée et (C) est un conflit.

Exemple 3 *En reprenant l'exemple 1 avec obs* $= \{a = 1, b = 0, c = 1, d = 1, e = 0\}$ *et* $SD = \{RRA_1, RRA_2, RRA_3\}$*, l'expression suivante :*

$$RRA_1 \cup obs \cup \{\forall c \in supp(RRA_1) \subseteq \mathcal{C} \mid \neg AB(c)\}$$

est inconsistante indiquant que $supp(RRA_1)$ *est un conflit. De même pour* RRA_2 *et* RRA_3*.*

[1] Nom issu du workshop international consacré au diagnostic utilisant, essentiellement, les techniques IA.

L'étape de localisation permet d'expliquer la faute. Ainsi, un espace de diagnostic sera généré. Un diagnostic, $\Delta \subseteq \mathcal{C}$, est défini comme étant un ensemble de composants tous fautifs et expliquant la défaillance du système tel que l'expression suivante est consistante :

$$SD \cup obs \cup \{\neg AB(c) \mid c \in \mathcal{C} - \Delta\}$$

Le principe de parcimonie permet d'introduire la notion de diagnostic minimal [2]. Un diagnostic est dit minimal s'il ne contient aucun sous-ensemble tel que ce sous-ensemble soit aussi un diagnostic. En outre, le diagnostic minimal Δ peut être formalisé par la consistance de l'expression suivante [Reiter, 1987] :

$$SD \cup obs \cup \{AB(c) \mid c \in \Delta \subseteq \mathcal{C}\} \cup \{\neg AB(c) \mid c \in \mathcal{C} - \Delta\}$$

Il est préférable de travailler avec des diagnostics minimaux étant donné que la tâche de diagnostic consiste à expliquer une défaillance en incriminant un minimum de composants.

Le calcul des diagnostics s'effectue par la génération des « hitting-sets » des conflits. Nous avons vu qu'un conflit, ψ_i, est un ensemble de composants supposés corrects, servant d'hypothèse d'inférence et entraînant une inconsistance de l'expression :

$$SD \cup obs \cup \{\neg AB(c) \mid c \in \psi_i \subseteq \mathcal{C}\}$$

Un « hitting-set » est défini comme étant un ensemble h de composants tel que :

$$\forall i, \psi_i \cap h \neq \{\}$$

De plus, un « hitting-set » est dit minimal s'il ne contient aucun sous-ensemble qui soit lui-même un « hitting-set ».
Un « hitting-set » minimal, h, intersectant tous les conflits ψ_j soulevés par la défaillance, satisfait l'expression :

$$SD \cup obs \cup \{\neg AB(c) \mid c \in \mathcal{C} - h\}$$

Les « hitting-set » minimaux sont donc des diagnostics minimaux.

Ils existent plusieurs types d'algorithmes pour calculer les hitting-sets, notamment le HS-tree présenté dans [Reiter, 1987] dont la version corrigée est HS-dag décrit dans [Greiner et al., 1989] ou encore des algorithmes génétiques. Nous présenterons, par la suite, un algorithme basé sur l'algèbre booléenne [Lin and Jiang, 2003] utilisé dans le cadre de ce stage.

[2] La notion de diagnostic minimal se base sur le principe qu'un composant est présumé correct tant qu'aucun soupçon ne le remet en cause [Raiman, 1992]

3.1.3 Diagnostic à base de modèles avec modèles de fautes

L'objectif des techniques de diagnostic avec modèles de fautes est d'affiner la localisation des composants fautifs. En effet, le diagnostic basé sur un modèle de bon fonctionnement permet de détecter et de localiser le dysfonctionnement en générant un espace de candidats pouvant expliquer la défaillance. L'espace obtenu est souvent vaste. Il est alors nécessaire d'utiliser diverses techniques afin d'éliminer des candidats qui ont une probabilité de pertinence moindre[Struss and Dressler, 1989]. Parmi ces techniques, nous pouvons citer :

- L'utilisation d'hypothèses telles que :
  - La non compensation : le comportement fautif ne peut être masqué par le comportement d'un ou plusieurs autres composants fautifs.
  - L'exonération : tout composant fautif affecte, de manière visible, les parties du système dans lequel il intervient. Si ces parties ne révèlent aucune anomalie alors les composants de ces parties sont mises hors d'état de cause.
- La limitation des fautes : la cardinalité des diagnostics est restreinte. Ainsi, les fautes étudiées sont limitées à un certain nombre de composants fautifs (plus généralement les diagnostics sont limités aux fautes simples voir doubles, la probabilité d'avoir des fautes faisant intervenir un ensemble de composants de cardinalité supérieure à 2 est supposée faible). Cette technique permet de limiter de façon considérable la combinatoire.
- L'utilisation de modèles de fautes : ces modèles permettent d'identifier le mode de fonctionnement des composants de chaque candidat. Un candidat est éliminé de l'espace de diagnostic lorsqu'aucun des modèles de fautes, pour ce candidat, ne permet d'expliquer la défaillance (aucun des modèles n'est satisfait par l'observation ayant révélée le dysfonctionnement).

Ces techniques peuvent être combinées afin de réduire considérablement les candidats potentiels mais au risque d'éliminer des candidats pertinents.

Ainsi, le modèle de bon fonctionnement est utilisé, essentiellement, pour détecter et localiser une défaillance. Les modèles de fautes, ou modèles pour modes de comportement [de Kleer and Williams, 1989], permettent une identification des composants fautifs. De ce fait, l'utilisation des modèles de fautes intervient dans la partie « identification » de la tâche de diagnostic, après la détection et la localisation.

Une fois, la localisation effectuée et l'espace de diagnostic généré, l'identification consiste à attribuer à chaque composant un mode de comportement et cela indépendamment pour chaque diagnostic de l'espace. Un candidat étant un diagnostic, aucun de ses composants ne pourra se voir attribuer le mode de fonctionnement correct. De plus, si aucune combinaison de mode de fonctionnement n'est pertinente alors certain composants peuvent être placés en mode de fonctionnement inconnu et se voir éliminés de l'espace de diagnostic sous l'hypothèse que le mode de fonctionnement inconnu a une probabilité très faible d'apparaître.

Bien que l'utilisation de modèles de fautes, dans le diagnostic à base de modèles, permette une meilleure discrimination des candidats pouvant expliquer le mauvais fonctionnement du système, ils présentent quelques inconvénients dont les plus notables sont :

- la difficulté, voire l'impossibilité, de modéliser le comportement des composants en cas de fautes.
- le problème de l'explosion combinatoire. En effet, pour chaque composant possédant k modes de fonctionnement et pour chaque candidat, de l'espace de diagnostic, constitué en moyenne de n composants, le nombre de combinaison , à considérer, est alors de l'ordre de $(k-1)^n$.

3.2 Diagnostic dans RESEDA

Dans le cadre du projet RESEDA, le langage choisi pour la modélisation du système est à base d'équation analytiques de redondances (RRA). Cette représentation a été choisie pour son avantage à pouvoir rapidement être traitée par un système informatique.

La modélisation nécessite une étude préalable du système à diagnostiquer. Cette étude doit permettre de mettre à jour des variables dites redondantes. Ces variables ont la caractéristique, non seulement, de pouvoir être observées (généralement à partir de capteurs) mais aussi de pouvoir être calculées par des relations mathématiques.

3.2.1 Description du modèle

La description du système fournie par le constructeur automobile Renault est un ensemble constitué de :

- Documents de spécifications sur le FReinage ÉLectrique (projet FREL) qui devrait équiper les prochaines séries de véhicules. Ces documents concernent plus particulièrement la fonction « Gestion de la Décélération Proche de l'Arrêt » (GDPA).
- Un modèle conceptuel des fonctions de calculs des divers prestations du freinage Électrique. Ce modèle a été réalisé avec le logiciel Simulink de MatLab. Simulink permet la modélisation de systèmes complexes à l'aide de diagrammes (voir figures 3.3 et 3.4).

Partant de ces documents, l'objectif est d'extraire des informations nous permettant d'identifier les éléments diagnosticables. La méthodologie à suivre serait :

- Identifier et dresser la liste des capteurs et composants pouvant être diagnostiquer.
- Extraire les informations reliant les composants à diagnostiquer dans le système pour pouvoir établir des relations analytiques de redondances en éliminant les variables intermédiaires non observables.

3.2.2 Fonction GDPA

Le système à diagnostiquer, à savoir le véhicule, étant trop vaste et les informations fournies par Renault insuffisantes, le démonstrateur, pour le projet RESEDA, a été limité,

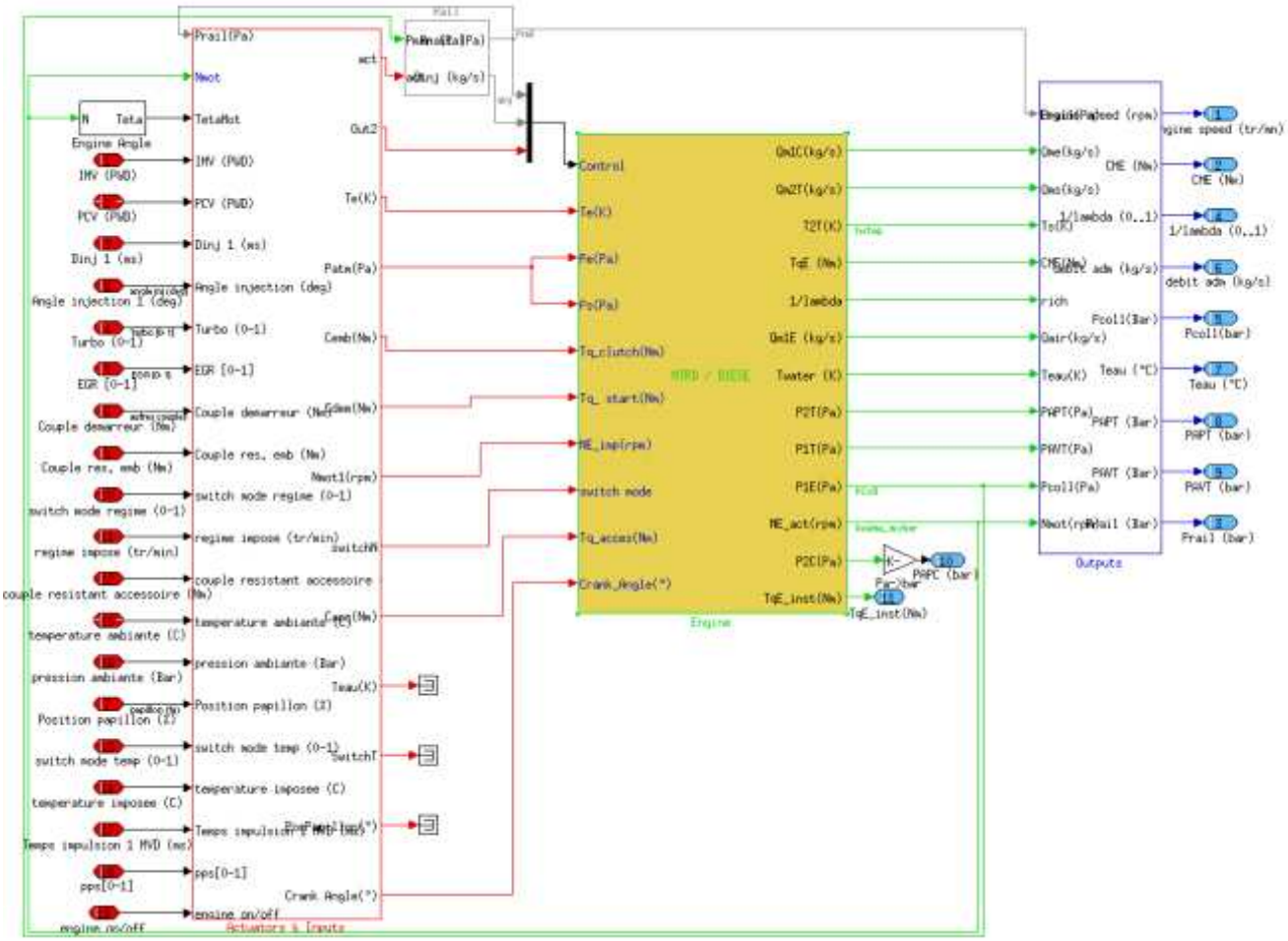

FIG. 3.3: Exemple de diagramme Simulink : Schéma global du moteur

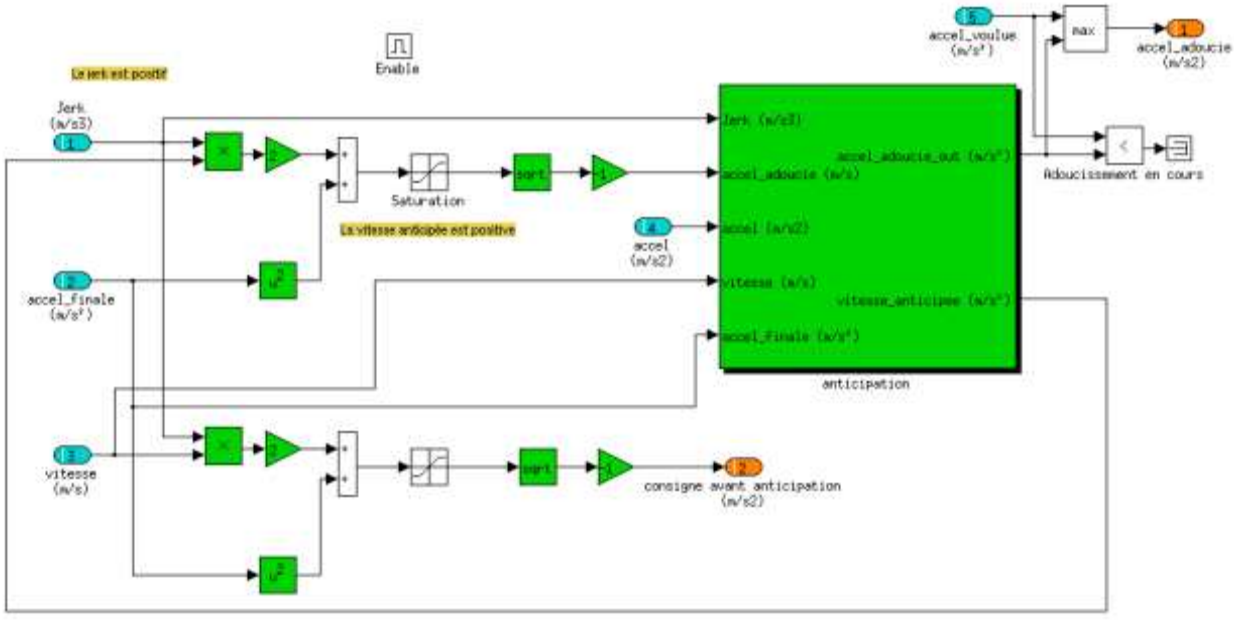

FIG. 3.4: Exemple de diagramme simulink : Calcul de l'accélération adoucie par le GDPA

dans un premier temps, à l'une des fonctions du freinage électrique à savoir la fonction GDPA.

La fonction GDPA (Gestion de la Décélération Proche de l'Arrêt) est une fonction de confort permettant d'éviter les « à-coups » plus ou moins brutaux dus à l'arrêt du véhicule, plus précisément lors du passage d'une vitesse non nulle à la vitesse nulle. Afin d'éviter ces « à-coups », le conducteur doit relâcher la pédale de frein au fur et à mesure que la voiture s'immobilise. Cette action nécessite, de la part du conducteur, une attention particulière. Ainsi, le GDPA se propose de décharger le conducteur de cette action en la prenant entièrement en charge.

Le principe du GDPA est de suivre une courbe de décélération permettant de passer progressivement d'une accélération initiale constante à une accélération finale immobilisant la voiture tout en maintenant une augmentation de la distance d'arrêt constante par rapport à la distance d'arrêt sans adoucissement. La courbe d'accélération sera linéairement dépendant du temps, la dérivée temporelle de l'accélération sera donc constante pendant toute la phase d'adoucissement. Cette courbe est caractérisée par trois paramètres :

- une accélération initiale γ_C
- une accélération finale γ_F
- une distance X telle que $Distance\ d'arrêt_{adoucie} = Distance\ d'arrêt + X$

3.2.3 Description du modèle pour le diagnostic

L'étude de la fonction GDPA (voir annexe A) a permis d'extraire des équations mathématiques pouvant être utilisées pour établir des relations de redondances. De plus, ces équations sont calculées directement par un calculateur embarqué dans le véhicule. Les résultats de ces équations étant régulièrement émis sur le bus CAN du véhicule, le programme de diagnostic est en mesure d'utiliser ces valeurs pour évaluer les RRA.
L'équation de redondance peut alors se résumer à la forme simplifiée : $Valeur_{observée} - Valeur_{calculée} = 0$, la valeur observée étant issue d'un capteur et la valeur calculée étant déterminée et émise sur le CAN par le calculateur.

Ainsi, cette étude a permis de dresser la liste des composants pouvant être diagnostiqués (tableau 3.2) ainsi que d'établir les relations de redondances.

Symbole du composant	Description
C_{emb}	Capteur indiquant la position de l'embrayage.
C_{V_m}	Capteur indiquant le régime moteur.
C_{pente}	Capteur indiquant la valeur de la pente (inclinaison du véhicule par rapport à l'horizontal).
C_V	Capteur indiquant la vitesse du véhicule.
C_γ	Capteur indiquant l'accélération du véhicule.
$A_{actuator}$	Capteur indiquant la force de freinage appliquée par les actionneurs au niveau des roues.
$C_{accélérateur}$	Capteur indiquant l'accélération exigé par le conducteur au niveau de la pédale d'accélération.
C_{levier}	Capteur indiquant la valeur du rapport engagé au niveau de la boîte de vitesse.

TAB. 3.2: Liste des composants diagnosticables

De plus, les quatre relations de redondances, suivantes, ont pu être établies :

1. $r_1 = Vitesse\_véhicule - Vitesse\_Véhicule_{estimée}$
2. $r_2 = Accélération\_véhicule - Accélération_{estimée}$
3. $r_3 = Pente\_véhicule - Pente_{estimée}$
4. $r_4 = Rapport\_boîte\_vitesse - Rapport\_boîte\_vitesse_{calculé}$

Les supports associés à chacun des quatre RRA sont :

1. $supp(RRA_1) = \{C_{emb}, C_{V_m}, C_V, C_\gamma, A_{actuator}, C_{accélérateur}, C_{levier}\}$
2. $supp(RRA_2) = \{C_{emb}, C_{V_m}, C_V, C_\gamma, A_{actuator}, C_{accélérateur}, C_{levier}\}$
3. $supp(RRA_3) = \{C_{emb}, C_{V_m}, C_{pente}, C_V, C_\gamma, A_{actuator}, C_{accélérateur}, C_{levier}\}$
4. $supp(RRA_4) = \{C_{emb}, C_{V_m}, C_V, C_\gamma, A_{actuator}, C_{accélérateur}, C_{levier}\}$

La matrice de signatures des fautes simples pour le système GDPA est :

	$S_{F_{\{C_{emb}\}}}$	$S_{F_{\{C_{V_m}\}}}$	$S_{F_{\{C_{pente}\}}}$	$S_{F_{\{C_{V,}\}}}$	$S_{F_{\{C_\gamma\}}}$	$S_{F_{\{A_{actuator}\}}}$	$S_{F_{\{C_{accélérateur}\}}}$	$S_{F_{\{C_{levier}\}}}$
RRA_1	1	1	0	1	1	1	1	1
RRA_2	1	1	0	1	1	1	1	1
RRA_3	1	1	1	1	1	1	1	1
RRA_4	1	1	0	1	1	1	1	1

TAB. 3.3: Matrice de signatures pour la fonction GDPA

Chapitre 4

Contribution

Dans le cadre du projet RESEDA, le système à diagnostiquer se limitera au système devant prendre en charge la fonction GDPA. Ce système est, ainsi, décrit par quatre relations de redondances analytiques (voir section 3.2.3).
Toutefois, en faisant abstraction du système spécifique au projet, nous adopterons une approche générale qui pourra ensuite être adaptée à n'importe quel système pour le peu que ce système puisse être correctement décrit et que les informations requises par les différents algorithmes puissent être disponible.

4.1 Détection

La détection des erreurs s'effectue en évaluant les équations de redondances selon l'approche FDI (voir 3.1.2).

Les erreurs sont détectées, pour une observation donnée, lorsqu'il existe au moins une RRA du modèle telle que le résultat [1] de son évaluation par l'observation est non nul.

4.2 Localisation

La localisation, quant à elle, s'effectue en utilisant l'approche DX (voir section 3.1.2).

Les supports des RRA non satisfaites (résidus non nuls) deviennent alors des ensembles de conflits. Chacun de ces supports contient, au moins, un composant défectueux.

La génération des diagnostics (la localisation) s'effectue en calculant les « hitting-sets » de ces conflits.

L'algorithme utilisé pour le calcul des « hitting-sets » est basé sur l'utilisation de l'algèbre booléenne [Lin and Jiang, 2003].
Les conflits sont représentés sous forme normale conjonctive dont tous les atomes sont négatives. Les atomes représentent les composants du système.

[1] le résultat de l'évaluation d'une RRA est appelée résidu de la RRA

Exemple 4 *Dans l'exemple 3, la RRA_1, la RRA_2 et la RRA_3 sont non satisfaites. $supp(RRA_1)$, $supp(RRA_2)$ et $supp(RRA_3)$ sont des conflits. La forme normale conjonctive de ces conflits est alors :*

$$FNC(conflits) = (\neg X1 \wedge \neg X2) \ \vee \ (\neg X1 \wedge \neg A1 \wedge \neg A2 \wedge \neg O1) \ \vee \ (\neg X1 \wedge \neg A2 \wedge \neg O1)$$

Un « hitting-set » h est une conjonction d'atomes positives représentant les composants.

Exemple 5 *Un « hitting-set » h=$\{X2, A2\}$ peut être représenté sous la forme $h = X2 \wedge A2$.*

De plus, h a la propriété suivante :

$$FNC(conflits) \ \wedge \ h \ = 0$$

Exemple 6 *$h = X2 \wedge A2$ est un hitting set pour l'exemple 4 car :*

$FNC(conflits) \ \wedge \ h$
$= ((\neg X1 \wedge \neg X2) \ \vee \ (\neg X1 \wedge \neg A1 \wedge \neg A2 \wedge \neg O1) \ \vee \ (\neg X1 \wedge \neg A2 \wedge \neg O1)) \ \wedge \ (X2 \wedge A2)$
$= 0$

L'algorithme permettant de calculer les « hitting-sets » à partir d'une forme normale conjonctive des conflits est donnée par l'algorithme 4.1.

Exemple 7 *En appliquant l'algorithme 4.1 à l'exemple 4, on obtient :*
$h \ = \ Hitting\text{-}Set((\neg X1 \wedge \neg X2) \ \vee \ (\neg X1 \wedge \neg A1 \wedge \neg A2 \wedge \neg O1) \ \vee \ (\neg X1 \wedge \neg A2 \wedge \neg O1))$
$h \ = \ X1 \vee \ Hitting\text{-}Set((\neg X2) \ \vee \ (\neg A1 \wedge \neg A2 \wedge \neg O1) \ \vee \ (\neg A2 \wedge \neg O1))$
$h \ = \ X1 \ \vee \ X2 \wedge \ Hitting\text{-}Set((\neg A1 \wedge \neg A2 \wedge \neg O1) \ \vee \ (\neg A2 \wedge \neg O1))$
$h \ = \ X1 \ \vee \ X2 \wedge (A2 \ \vee \ O1)$
$h \ = \ X1 \ \vee \ (X2 \wedge A2) \ \vee \ (X2 \wedge O1)$

Les « hitting sets » sont $\{\{X1\}, \{X2, A2\}, \{X2, O1\}\}$

L'algorithme 4.1 a la propriété d'être l'un des algorithmes les plus rapides pour le calcul des « hitting-sets » [Lin and Jiang, 2003]. Cela justifie, ainsi, son utilisation dans le cadre de ce projet.

De plus, si nous envisageons le pire des scénarios, à savoir :
- n RRA non satisfaites, donc n conflits.
- l'intersection des conflits deux à deux donne un ensemble vide
- m composants par support (un conflit étant le support d'une RRA non satisfaite)

La taille de l'espace de diagnostic est alors m^n (m^n diagnostics) et chaque diagnostic contient n composants. L'espace de diagnostic pouvant croître de manière exponentielle en fonction des RRA non satisfaites, il est important d'affiner cette espace.

Algorithme 4.1 Hitting-Set

Entrée: $\mathcal{H}$ sous forme normale conjonctive
Entrée: $\mathcal{F}$, l'ensemble des formules bien formées
Entrée: $\mathcal{A}$, l'ensemble des atomes (composants) du système
Sortie: $h = h_1 \vee ... \vee h_n$ et $h \wedge \mathcal{H} = 0$

1: Éliminer, de $\mathcal{H}$, tout $c_1 \in \mathcal{H} \mid \exists c_2 \in \mathcal{H} - \{c_1\}, c2 \subseteq c_1$
2: **si** $\mathcal{H} = false$ **alors**
3: $h \leftarrow true$
4: **sinon**
5: **si** $\mathcal{H} = true$ **alors**
6: $h \leftarrow false$
7: **sinon**
8: **si** $\exists e \in \mathcal{A} \mid \mathcal{H} = \neg e$ **alors**
9: $h \leftarrow e$
10: **sinon**
11: **si** $\exists e \in \mathcal{A}, \exists C \in \mathcal{F} \mid \mathcal{H} = \neg e \wedge C$ **alors**
12: $h \leftarrow e \vee$Hitting-Set(C)
13: **sinon**
14: **si** $\exists e \in \mathcal{A}, \exists C \in \mathcal{F} \mid \mathcal{H} = \neg e \vee C$ **alors**
15: $h \leftarrow e \wedge$Hitting-Set(C)
16: **sinon**
17: $\{\mathcal{H} = C, C \in \mathcal{F}\}$
18: choisir $\neg e$ un atome arbitraire de $\mathcal{H}$
19: $C_1 \leftarrow \{c \mid c \in C$ et $\neg e \notin c\}$
20: $C_2 \leftarrow \{c \mid c \cup \{\neg e\} \in C$, ou $c \in C$ et $\neg e \notin c\}$
21: $h \leftarrow (e \wedge$Hitting-Set$(C_1)) \vee ($Hitting-Set$(C_2))$
22: **fin si**
23: **fin si**
24: **fin si**
25: **fin si**
26: **fin si**

4.3 Modèles de fautes

Comme indiqué dans la section 3.1.3, l'utilisation des modèles de fautes permet d'affiner le diagnostic en réduisant considérablement l'espace de diagnostic généré par la localisation.
Pour cela, on tente d'affecter à chaque composant d'un diagnostic, un mode de comportement [2]. Dans le cas où aucune combinaison n'est jugée cohérente alors le diagnostic est éliminé de l'espace.

Ainsi, un mode de comportement devra être associé à chaque composant ; Nous introduirons, donc, la notation suivante : c_{n_j} où c_n est un composant et j le mode de

[2] Le mode de comportement correct ne peut être attribué à aucun composant du diagnostic. En effet, par définition, un diagnostic (ou candidat) est un ensemble de composants dont tous les éléments sont fautifs.

comportement de c_n ($j = 0$ sera utilisé pour le mode correct). De plus, nous introduirons la notion de support d'un modèle qui sera l'ensemble des composants, avec indication sur les modes de comportements, utilisés par le modèle. Soit pour un modèle m décrit par des relations de redondances analytiques :

$$supp(m) = \bigcup_{RRA_i \in m} supp(RRA_i)$$

De plus, soit l'ensemble $\mathcal{R}$ des RRA d'un système composé d'un ensemble $\mathcal{C}$ de n composants tel que $\mathcal{C} = \{c_1, ..., c_n\}$, nous pouvons définir un modèle comportemental m_d, pour un ensemble de composants $d = \{c_{1_{i_1}}, ..., c_{n_{i_n}}\} \mid i_j$ est un mode de comportement pour le composant c_j ($mode(c_{j_{i_j}}) = i_j$), par l'expression :

$$m_d = \{RRA_i \mid \forall RRA_i \in \mathcal{R} \; et \; \forall c_{j_k} \in supp(RRA_i), mode(c_{j_k}) = k = 0(correct) \; où \; c_{j_k} \in d \; et \; \forall c_{j_k} \in d, c_{j_k} \in supp(m_d)\}$$

De même, soit $\mathcal{M}$ l'ensemble des modèles comportementaux du système, nous définissons la fonction $Model$ comme étant une fonction prenant en entrée un ensemble de composants d et retournant l'ensemble $\mathcal{M}_d$ des modèles comportementaux m_{d_I} :
$I = \{i_1, ..., i_n\}$, i_j étant le mode de faute i_j pour le composant j
$d_I = \{k_{i_k} \mid i_k \in I$ et $\exists j \mid k_j \in d\}$

$$Model(d) = \mathcal{M}_d = \bigcup_{\forall I \mid 0 \notin I} \{m_{d_I}\}$$

Une combinaison de fautes $I = \{i_1, ..., i_n\}$ pour un diagnostic d et une observation *obs* est dite cohérente si :

$$\forall RRA_i \in m_{d_I}, eval(RRA_i, obs) = 0$$

De même, un diagnostic d est dit cohérent avec une observation *obs* s'il existe au moins une combinaison de faute cohérente soit :

$$\exists m \in Model(d) \mid \forall RRA_i \in m, eval(RRA_i, obs) = 0$$

Un diagnostic cohérent ne peut être éliminé de l'espace de diagnostic, par contre si le diagnostic est incohérent le diagnostic peut être éliminé.

Ainsi, toutes les combinaisons doivent être testées pour pouvoir éliminer un diagnostic. Dans ce cas, si un diagnostic est composé de n composants et chaque composant ayant k modes de fonctionnements ($k - 1$ modes de fautes + le mode correct [3] alors le nombre de combinaisons est $(k - 1)^n$.
Si nous reprenons le pire des scénarios défini à la section 4.2 : le nombre de cas à considérer pour affiner l'espace serait de l'ordre de $m^n * (k - 1)^n$.

[3]Nous pouvons aussi dire, pour généraliser et par abus de langage, k modes de fautes, le mode de fonctionnement étant alors un mode de fautes particulier.

De plus, il peut s'avérer que l'espace de diagnostic généré lors de la phase de localisation ne contienne pas le « véritable diagnostic ». Dans ce cas, la phase d'identification avec modèles de fautes éliminera tous les diagnostics de l'espace étant donnée qu'aucun n'est pertinent. Ainsi, l'espace de diagnostic final sera vide (voir exemple 8).
Pour faire face à cette éventualité, il peut être intéressant d'explorer les sur-ensembles des diagnostics éliminés (diagnostics jugés non cohérents).
Donc lorsqu'un diagnostic est jugé incohérent, il est éliminé de l'espace de diagnostic. Cependant, les sur-ensembles de ce diagnostic seront explorés et si l'un des sur-ensembles est cohérent alors il devient un diagnostic qui doit être ajouté dans l'espace de diagnostics (voir exemple 9). De plus, les sur-ensembles de ce dernier ne seront ni explorés ni ajoutés à l'espace de diagnostic. En effet, seuls les diagnostics minimaux doivent être ajoutés à l'espace.
L'algorithme 4.2 résume ce principe.

Algorithme 4.2 Modèle de fautes avec exploration de sur-ensembles

Entrée: $\mathcal{D}$ = l'espace de diagnostic généré par la localisation.
Entrée: *obs* = Observation pour laquelle une défaillance a été détectée.
Sortie: $\mathcal{D}_f$ = l'espace de diagnostic final.

1: $\mathcal{D}_f \leftarrow \{\}$
2: **pour tout** $d \in \mathcal{D}$ **faire**
3: **si** d cohérent avec *obs* **alors**
4: $\mathcal{D}_f \leftarrow \mathcal{D}_f \cup \{d\}$
5: **sinon**
6: **pour tout** $d' \in sur-ensemble(d)$ **faire**
7: **si** d' cohérent avec *obs* **alors**
8: **si** $\nexists e \in \mathcal{D}_f \mid e \subseteq d'$ **alors**
9: **pour tout** $e \in \mathcal{D}_f \mid d' \subset e$ **faire**
10: $\mathcal{D}_f \leftarrow \mathcal{D}_f - \{e\}$
11: **fin pour**
12: $\mathcal{D}_f \leftarrow \mathcal{D}_f \cup \{d'\}$
13: **fin si**
14: **fin si**
15: **fin pour**
16: **fin si**
17: **fin pour**

Exemple 8 *Prenons l'exemple du polybox (figure 4.1) qui est un système composé de 3 multiplicateurs et de 2 aditionneurs [de Kleer and Williams, 1987] :*

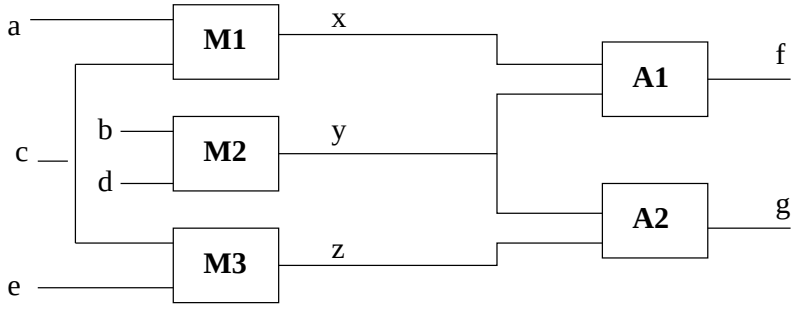

FIG. 4.1: Polybox.

Le modèle de bon fonctionnement est décrit par les RRA suivantes [Cordier et al., 2000] :

- $RRA_1 : f - (a * c + b * d) = 0$
- $RRA_2 : g - (b * d + c * e) = 0$
- $RRA_3 : f - g - a * c + c * e = 0$

Les supports de ces RRA sont :

- $supp(RRA_1) = \{M1, M2, A1\}$.
- $supp(RRA_2) = \{M2, M3, A2\}$.
- $supp(RRA_3) = \{M1, M3, A1, A2\}$.

De plus, nous fixons pour chaque composant, pour cet exemple, trois modes de fonctionnement :

- *0 étant le mode de bon fonctionnement.*
- *1 étant le mode de faute qui associe la sortie du composant à sa première entrée (par exemple pour $M2 : y = b$).*
- *2 étant le mode de faute qui associe la sortie du composant à sa deuxième entrée (par exemple pour $M2 : y = d$).*

Pour les différentes combinaisons de faute, les RRA seront extraites. Ainsi, pour le modèle $m_{I=\{1,0,0,0,0\}}$ associé à la combinaison de faute $\{M1_1, M2_0, M3_0, A1_0, A2_0\}$, la description est :

- $RRA_1 : f - (a + b * d) = 0$ *et* $supp(RRA_1) = \{M1, M2, A1\}$.
- $RRA_2 : g - (b * d + c * e) = 0$ *et* $supp(RRA_2) = \{M2, M3, A2\}$.
- $RRA_3 : f - g - a + c * e = 0$ *et* $supp(RRA_3) = \{M1, M3, A1, A2\}$.

Supposons que nous ayons l'observation suivante : obs $= \{a = 4, b = 5, c = 3, d = 7, e = 2, f = 4, g = 37\}$.
La RRA_1, la RRA_2 et la RRA_3 ne sont pas satisfaites, indiquant, ainsi, une défaillance. La localisation par calcul des hitting-set de l'ensemble des conflits :

$$conflits = \{supp(RRA_1), supp(RRA_2), supp(RRA_3)\}$$

permet de générer l'espace de diagnostics suivant :

$$\mathcal{D} = \{\{M1, M2\}, \{M1, M3\}, \{M1, A2\}, \{M2, M3\}, \{M2, A1\},$$

$$\{M2, A2\}, \{M3, A1\}, \{A1, A2\}\}$$

En utilisant les modèles de fautes, aucun diagnostic de $\mathcal{D}$ n'est cohérent avec l'observation obs. Ainsi, tous les diagnostics de $\mathcal{D}$ sont éliminés.

Exemple 9 *Dans l'exemple 8, l'espace de diagnostics suivant a été déterminé :*

$$\mathcal{D} = \{\{M1, M2\}, \{M1, M3\}, \{M1, A2\}, \{M2, M3\}, \{M2, A1\},$$

$$\{M2, A2\}, \{M3, A1\}, \{A1, A2\}\}$$

L'utilisation des modèles de fautes a permis de montrer qu'aucun de ces diagnostics ne s'avérait être cohérent avec obs. L'exploration des sur-ensembles de chacun de ces diagnostics est alors nécessaire. Pour cela, l'algorithme 4.2 est utilisé.
En utilisant cet algorithme, un unique diagnostic, cohérent avec obs, a été trouvé :

$$\mathcal{D}_f = \{\{M1, M3, A1\}\}$$

4.4 Variables interrogeables

Les relations de redondances analytiques font intervenir uniquement des variables observables provenant, par exemple, de capteurs. De plus, ces relations exploitent des redondances de variables. En effet, si la valeur d'une variable peut être obtenue de divers manières (par exemple une valeur provenant d'un capteur et une autre d'un calculateur) et qu'il existe une incohérence au niveau de cette variable, à un même instant, alors il est évident qu'au moins un des composants, ayant fourni une valeur pour cette variable, est défaillant.
Ainsi, les RRA sont caractérisées par l'utilisation exclusive de variables observables et par l'exploitation des redondances au niveau des variables.

Dans le cadre du projet RESEDA, il existe deux types de variables :

- les variables observables : ces variables circulent librement sur le bus CAN du véhicule et peuvent être exploitées dans le cadre du diagnostic.
- les variables interrogeables : ces variables ne sont pas disponibles librement sur le bus. Toutefois, elles peuvent être interrogées par une demande explicite du programme de diagnostic. L'un des inconvénients majeurs de ces variables est le coût temporel de l'interrogation.
  En effet, la requête d'interrogation doit être émise par le programme de diagnostic sur le bus puis le composant, calculant la variable, doit émettre sur le même bus, après réception de la requête, la valeur de la variable interrogée.
  Ainsi, une interrogation de variable peut être plus ou moins longue.

L'utilisation des variables interrogeables peut permettre d'affiner le diagnostic afin d'obtenir, dans le meilleur des cas, un espace contenant un unique diagnostic.
En effet, les variables interrogeables apportent des informations supplémentaires qui pourraient se révéler utiles pour discriminer les diagnostics. Toutefois, ces variables devant être utilisées avec parcimonie, il est important de déterminer les variables pouvant apporter

une information supplémentaire tout en maîtrisant le coût temporel.

Soit $\mathcal{C}$ l'ensemble des composants intervenant dans l'espace de diagnostics $\mathcal{D}$ tel que :

$$\mathcal{C} = \bigcup_{d \in \mathcal{D}} d$$

Toute variable provenant de composants appartenant à $\mathcal{C}$ peut permettre d'incriminer ces composants.

Nous introduirons de plus la notion de coût temporel d'une relation de redondance analytique.
Pour cela, nous définissons $\mathcal{V}_{t_i}$ comme étant l'ensemble des variables utilisées par la RRA_i. Soit $\mathcal{V}_i$ l'ensemble des variables observables utilisées par RRA_i et $\mathcal{V}_q$ l'ensemble des variables interrogeables utilisées par RRA_i, nous avons alors :

$$\mathcal{V}_{t_i} = \mathcal{V}_i \cup \mathcal{V}_{q_i}$$

Le coût de la RRA_i est alors :

$$coût(RRA_i) = \sum_{v \in \mathcal{V}_q} coût(v)$$

Le coût d'une RRA, dont l'ensemble $\mathcal{V}_q$ est vide, est nul. En effet, la RRA ne fait intervenir, alors, que des variables observables.

L'ensemble des relations de redondance $\mathcal{R}_t$ du système peut alors être divisé en deux sous-ensembles $\mathcal{R}$ et $\mathcal{R}_q$:

$$\mathcal{R} = \{RRA_i \in \mathcal{R}_t \mid coût(RRA_i) = 0\}$$

$$\mathcal{R}_q = \mathcal{R}_t - \mathcal{R}$$

L'ensemble des RRA susceptible d'apporter des informations supplémentaire pour la discrimination est :

$$\mathcal{R}_c = \{RRA_i \in \mathcal{R}_q \mid supp(RRA_i) \cap \mathcal{C} \neq \{\}\}$$

L'ensemble $\mathcal{R}_c$ déterminé ; La contrainte de coût temporel doit être prise en compte en réduisant $\mathcal{R}_c$.
Le problème pourrait être traduit en un problème d'optimisation combinatoire qui serait le problème du sac à dos : il s'agit de conserver le maximum d'éléments de $\mathcal{R}_c$ tout en minimisant le coût temporel.
Le problème peut être énoncé sous la forme suivante :

$$\begin{array}{c|l} \max & x_1 + ... + x_n \\ \text{sous contrainte} & x_1 \times coût(RRA_1) + ... + x_n \times coût(RRA_n) \leq coût\_max \\ & \mathcal{R}_c = \{RRA_1, ..., RRA_n\} \text{ et } \{x_1, ..., x_n\} \in \{0,1\}^n \end{array}$$

Algorithme 4.3 glouton

Entrée: $\mathcal{R}_c = \{RRA_1, ..., RRA_n\}$
Entrée: $coût\_max \geq 0$
Sortie: $\sum_{RRA_i \in \mathcal{R}_f} coût(RRA_i) \leq coût\_max$

```
1: Trier R_c dans l'ordre croissant
2: R_f ← {}
3: coût_{R_f} ← 0
4: i ← 1
5: tant que i ≤ n et (coût_{R_f} + coût(RRA_i) ≤ coût_max) faire
6:    R_f ← R_f ∪ {RRA_i}
7:    coût_{R_f} ← coût_{R_f} + coût(RRA_i)
8:    i ← i + 1
9: fin tant que
```

Pour résoudre le problème du sac à dos, l'algorithme glouton (voir algorithme 4.3) sera utilisé. L'ensemble $\mathcal{R}_f$ contiendra alors les RRA pouvant être utilisées pour discriminer l'espace original.

L'ensemble $\mathcal{R}_f$ évalué ; Le problème, dés lors, est d'utiliser les RRA afin de discriminer l'espace de diagnostic.
Une solution est d'utiliser l'étape de détection (voir section 4.1) et de localisation (voir section 4.2) en utilisant les RRA appartenant à $\mathcal{R}_f$ et l'observation *obs*, ayant permis de générer l'espace de diagnostic $\mathcal{D}$, associée à l'observation obs_q issue de l'interrogation des variables utilisées par les RRA de $\mathcal{R}_f$.
Un espace de diagnostic $\mathcal{D}'$ est alors obtenu par l'étape de localisation.
Le problème consiste à relier $\mathcal{D}$ et $\mathcal{D}'$ pour obtenir l'espace de diagnostics final.
Les RRA utilisées pour générer $\mathcal{D}'$ modélisent une partie, au moins, du système et il en est de même pour les RRA utilisées pour générer $\mathcal{D}$ [4]. Il existe un chevauchement ou une inclusion entre les deux parties du système car :

$$\forall RRA_i \in \mathcal{R}_f, \exists RRA_j \in \mathcal{R} \mid supp(RRA_i) \cap supp(RRA_j) \neq \{\}$$

Ainsi, les diagnostics de l'espace final $\mathcal{D}_f$ sont des sur-ensembles (au sens large) des diagnostics de $\mathcal{D}$ et $\mathcal{D}'$:

$$\mathcal{D}_f = \{d \in \mathcal{D} \cup \mathcal{D}' | \exists d_1 \in \mathcal{D}, d_1 \subseteq d \; et \; \exists d_2 \in \mathcal{D}', d_2 \subseteq d\}$$

La relation précédente ne prend pas en compte tous les cas, elle est, essentiellement, vérifiée lorsqu'une partie du système est incluse dans l'autre. En effet, l'hypothèse selon laquelle les diagnostics finaux doivent être des sur-ensembles des diagnostics de $\mathcal{D}$ et de $\mathcal{D}'$ élimine des diagnostics potentiels. Une étude, plus générale, est nécessaire pour relier correctement $\mathcal{D}$ et $\mathcal{D}'$ notamment par l'utilisation de hitting-sets.

[4] En général, les RRA utilisées pour générer $\mathcal{D}$ proviennent de l'ensemble $\mathcal{R}$ des RRA du système qui sont supposées modéliser tout le système. Ainsi, les RRA utilisées pour générer $\mathcal{D}'$ ne peuvent modéliser qu'une partie du système.

L'algorithme 4.4 illustre l'utilisation des variables interrogeables dans le cas du projet RESEDA.

Algorithme 4.4 Variables interrogeables

Entrée: $\mathcal{D}$: espace de diagnostic initial non vide
Entrée: $\mathcal{R}_f$: ensemble de RRA utilisant des variables interrogeables
Entrée: *obs* : observation ayant permis le calcul de $\mathcal{D}$
Entrée: obs_q : valeurs des variables interrogeables utilisées par les RRA de $\mathcal{R}_f$
Sortie: $\mathcal{D}_f$: espace de diagnostic final

1: $conflit \leftarrow \{supp(RRA_i) \mid RRA_i \in \mathcal{R}_f \text{ et } eval(RRA_i, obs \cup obs_q) \neq 0\}$ {Étape de détection}
2: $\mathcal{D}' \leftarrow Hitting - Set(conflit)$ {Étape de localisation}
3: **si** $\mathcal{D}' \neq \{\}$ **alors**
4: **pour tout** $d \in \mathcal{D}$ **faire**
5: **pour tout** $d_2 \in \mathcal{D}' | d \subseteq d_2$ **faire**
6: $\mathcal{D}_f \leftarrow \mathcal{D}_f \cup \{d_2\}$
7: **fin pour**
8: **pour tout** $d_2 \in \mathcal{D}' | d_2 \subseteq d$ **faire**
9: $\mathcal{D}_f \leftarrow \mathcal{D}_f \cup \{d\}$
10: **fin pour**
11: **fin pour**
12: **sinon**
13: $\mathcal{D}_f \leftarrow \mathcal{D}$
14: **fin si**

4.5 Fenêtre temporelle

La détection de défaillance dans un système s'effectue, à un instant t, en prélevant une observation *obs*. S'il existe, au moins, une RRA non satisfaite par *obs* alors la défaillance est détectée puis le processus de diagnostic se poursuit par la tentative d'explication de la défaillance, notamment par la localisation.
Or, il est tout à fait possible, qu'une erreur fugitive, qui n'est pas, par ailleurs, significative d'une défaillance du système, ou même un bruit soit apparu à l'instant t. De même, une perturbation, due par exemple à un grain de poussière sur le circuit ou une fluctuation de courant, a pu provoquer une défaillance du système. Une telle perturbation ne devrait pas être prise en compte par le diagnostic étant donné qu'aucun des composants du système n'est fautif.
Il s'agirait donc d'améliorer la détection afin de tenir compte de ce type de phénomène. Ainsi, il est nécessaire de tenir compte de plusieurs observations prélevées successivement dans le temps.

Pour cela, une fenêtre de temps pourrait être utilisée afin de tenir compte des perturbations. Cette fenêtre stockerait, essentiellement, les observations prélevées ainsi que tous les résidus des RRA du système [5] (voir figure 4.2). En ce qui concerne les résidus,

[5] Un résidu est la valeur de l'évaluation de la RRA en appliquant une observation.

nous nous limiterons aux valeurs $\{0, 1\}$ à savoir si les RRA sont satisfaites ou non.

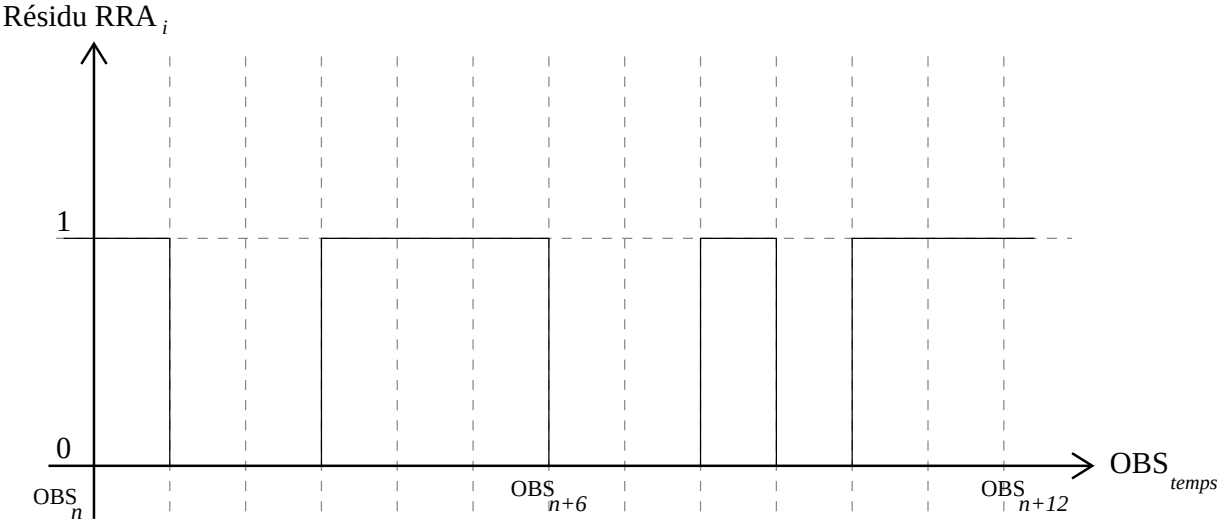

FIG. 4.2: Exemple de fenêtre temporelle de taille 13 pour la RRA_i.

La méthode la plus simple pour la détection serait de se fixer un seuil d'activation, S_A, puis pour chaque RRA de déterminer le nombre de fois dont la RRA n'a pas été satisfaite dans la fenêtre temporelle [6], et dans le cas où ce nombre serait supérieur au seuil d'activation des RRA, la RRA devrait être placée dans un ensemble $\mathcal{R}_d$.

Après traitement, l'ensemble $\mathcal{R}_d$ contiendra toutes les RRA non satisfaites. Cet ensemble servira à poursuivre le diagnostic en fournissant un ensemble de conflits qui pourra être utilisé par la localisation (voir section 4.2) :

$$conflits = \{supp(RRA_i) \mid RRA_i \in \mathcal{R}_d\}$$

Exemple 10 *Pour la figure 4.2, la RRA_i a été non satisfaite 8 fois dans les intervalles de temps suivants :*

$$\{[t_{n+12}, t_{n+10}[, [t_{n+9}, t_{n+8}[, [t_{n+6}, t_{n+3}[, [t_{n+1}, t_n]\}$$

L'algorithme 4.5 illustre l'utilisation d'une fenêtre de temps simple.

[6] Nous commencerons toujours les traitements temporelles en partant du dernier instant pour redescendre progressivement dans le temps.

Algorithme 4.5 Fenêtre temporelle simple

Entrée: Fenêtre temporelle
Entrée: $\mathcal{R}$: ensemble des RRA dont les résidus sont disponibles dans la fenêtre
Entrée: S_A : seuil d'activation pour les RRA
Sortie: $\mathcal{R}_d$: ensemble des RRA non satisfaites

```
1: R_d ← {}
2: pour tout RRA_i ∈ R faire
3:    n_RRA_i ← 0
4:    pour t ← fenêtre_temps_max à fenêtre_temps_min faire
5:       si RRA_i non satisfaite pour obs_t alors
6:          n_RRA_i ← n_RRA_i + 1
7:       fin si
8:    fin pour
9:    si n_RRA_i ≥ S_A alors
10:      R_d ← R_d ∪ {RRA_i}
11:   fin si
12: fin pour
```

Une autre technique d'utilisation des fenêtres temporelles consisterait, cette fois non plus à considérer toute la fenêtre, mais à considérer seulement la partie débutant à l'instant courant [7] (instant auquel la dernière observation a été effectuée) et se terminant au premier instant, antérieur, auquel l'observation associée satisfait la RRA.
Soit une fenêtre f d'intervalle de temps $[t_0, t_n]$, nous déterminons l'intervalle I_i, pour la RRA_i :

$$I_i = [t_k, t_n] \subseteq [t_0, t_n] \mid \forall t \in I_i, eval(RRA_i, obs_t) = 1$$
$$et\ si\ t_k > t_0\ eval(RRA_i, obs_{t_{k-1}}) = 0$$

Ainsi, seul l'intervalle I_i sera pris en compte pour la RRA_i lors de la détection.
Si le nombre d'observations dans l'intervalle de temps I_i est supérieur ou égal au seuil d'activation, S_A des RRA alors la RRA_i est considérée comme étant non satisfaite.

Exemple 11 *Pour la figure 4.2, I_i pour la RRA_i correspondant à :*

$$I_i = [t_{n+11}, t_{n+12}]$$

Si $S_A \leq 2$ alors la RRA_i est considérée comme non satisfaite.

De plus, une amélioration à cette dernière technique consisterait à prendre en compte les bruits pouvant apparaître et provoquer la satisfaction d'une RRA.

Prenons le cas où une RRA est insatisfaite sur toute la fenêtre temporelle exceptée à un instant (figure 4.3). Il serait, alors, légitime de penser qu'un bruit s'est glissé à cet instant donné. Ainsi, cet instant ne devrait pas être pris en compte.

Afin de permettre d'ignorer ces types de bruits, nous ajoutons un seuil de tolérance S_T à la technique précédente. Ainsi, si nous détectons des intervalles I'_i, pour une fenêtre

[7] si l'observation correspondant à cet instant ne satisfait pas la RRA.

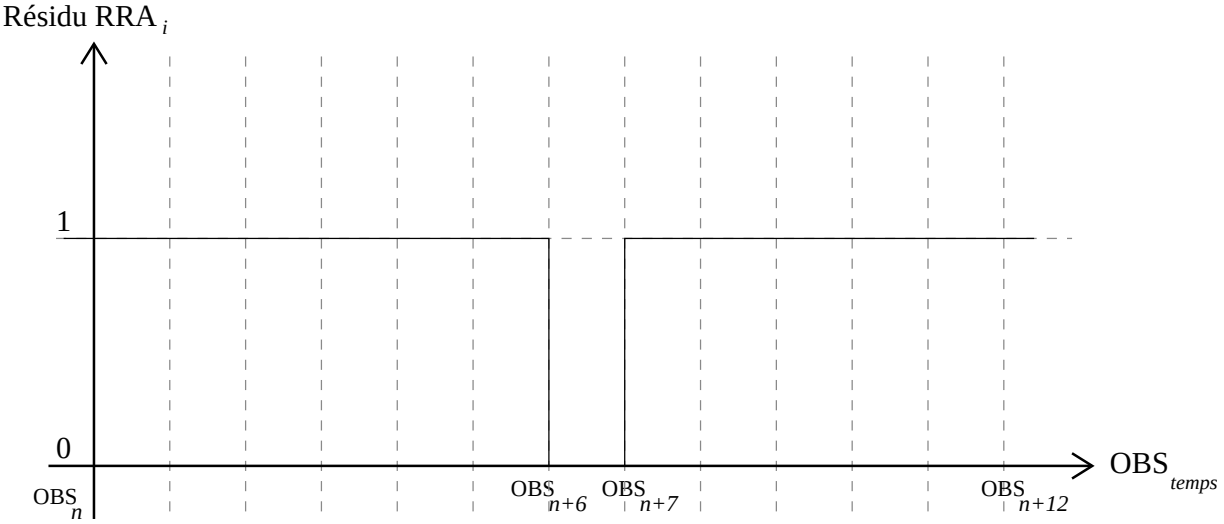

FIG. 4.3: Perturbation du résidu de la RRA_i.

f couvrant l'intervalle de temps $[t_0, t_n]$ tel que :
soit les fonctions :

- $nombre_0(RRA_i, I)$ indiquant le nombre de fois dont la RRA_i a été satisfaite sur l'intervalle I
- $nombre_1(RRA_i, I)$ indiquant le nombre de fois dont la RRA_i a été non satisfaite sur l'intervalle I

$$I'_i = [t_k, t_n] \subseteq [t_0, t_n] \mid nombre_1(RRA_i, I'_i) \geq S_A$$
$$et \quad \nexists J \subseteq I'_i \mid nombre_1(RRA_i, J) = 0 \wedge nombre_0(RRA_i, J) > S_T$$

et I'_i le plus grand intervalle vérifiant la propriété précédente alors la RRA_i peut être considérée comme non satisfaite..

L'algorithme 4.6 illustre l'application de cette technique.

Exemple 12 *Pour la figure 4.4, si nous fixons la zone de tolérance S_T à 2, nous obtenons un unique intervalle I'_i :*

$$I'_i = [t_{n+3}, t_{n+12}]$$

Si $nombre_1(RRA_i, I'_i) = 4 \geq S_A$ alors la RRA_i est considérée comme non satisfaite.

A partir de cette dernière technique généralisée de l'utilisation des fenêtres temporelles appliquées à la détection (algorithme 4.6), on peut revenir à la première technique simple (algorithme 4.5). Pour cela, il suffit de fixer la taille de la zone de tolérance à la taille de la fenêtre temporelle. Ainsi, l'intervalle I'_i va correspondre à l'intervalle de temps couvert par la fenêtre.

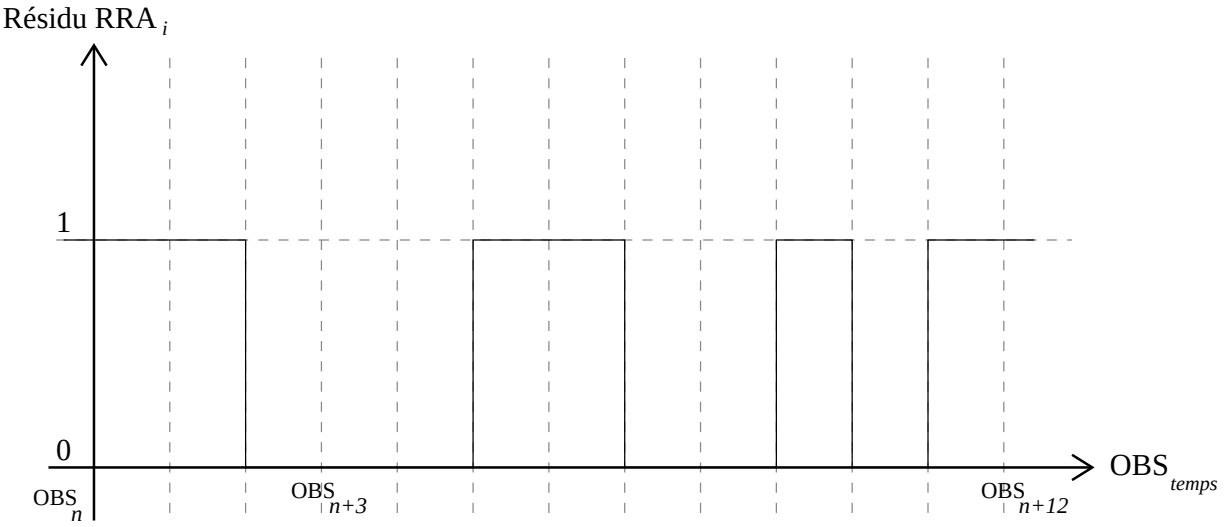

FIG. 4.4: Perturbation masquée du résidu de la RRA_i.

Algorithme 4.6 Fenêtre temporelle avec gestion de bruits

Entrée: Fenêtre temporelle
Entrée: $\mathcal{R}$: ensemble des RRA dont les résidus sont disponibles dans la fenêtre
Entrée: S_A : seuil d'activation pour les RRA
Entrée: S_T : taille de la zone de tolérance
Sortie: $\mathcal{R}_d$: ensemble des RRA non satisfaites

1: $\mathcal{R}_d \leftarrow \{\}$
2: **pour tout** $RRA_i \in \mathcal{R}$ **faire**
3: $n_{RRA_i} \leftarrow 0$
4: $Z_T \leftarrow 0$
5: **pour** $t \leftarrow$ fenêtre_temps_max à fenêtre_temps_min et $Z_T \leq S_T$ **faire**
6: **si** RRA_i non satisfaite pour obs_t **alors**
7: $n_{RRA_i} \leftarrow n_{RRA_i} + 1$
8: $Z_T \leftarrow 0$
9: **sinon**
10: $Z_T \leftarrow Z_T + 1$
11: **fin si**
12: **fin pour**
13: **si** $n_{RRA_i} \geq S_A$ **alors**
14: $\mathcal{R}_d \leftarrow \mathcal{R}_d \cup \{RRA_i\}$
15: **fin si**
16: **fin pour**

Chapitre 5

Framework de diagnostics

L'objectif du framework est de fournir un environnement de travail afin d'élaborer et d'expérimenter des techniques de diagnostic. Ainsi, cet environnement doit permettre l'ajout de nouveaux outils de diagnostic sans grande difficulté.

L'expérimentation doit permettre de choisir et d'améliorer les techniques de diagnostic pour un système en particulier. Suite à cette étape, des connaissances pourront être acquises. Ces connaissances pourront être « compilées » sous forme de structure de données optimisée telle les diagrammes de décisions binaires (BBD) [Bryant, 1992]. Ces structures peuvent ensuite être embarquées dans des systèmes temps réel critique.

5.1 Conception

Afin de respecter les contraintes liées au projet RESEDA, le langage de programmation Java a été choisi.

Dans la suite de cette section, la conception UML du framework sera présentée.

5.1.1 Diagramme de packages

Le programme est composé de quatre modules ou paquetages :

- **io** : module offrant des services d'entrées/sorties.
- **model** : module permettant de décrire le système à diagnostiquer.
- **reseda** : module offrant les services de diagnostic.
- **util** : module offrant des services de facilités pour l'utilisation du framework.

Le schéma représenté sur la figure 5.1 présente les différents modules ou paquetages du programme ainsi que les liens entre eux.

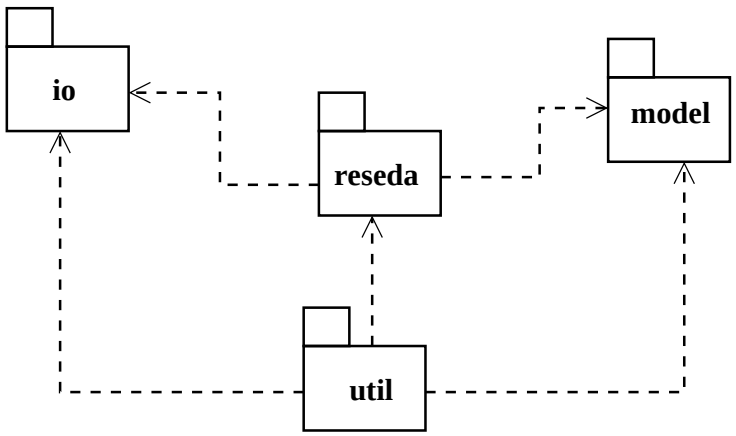

FIG. 5.1: Diagramme UML de packages

5.1.2 Diagramme de classes pour le package io

Le module « io » permet de connecter le framework avec l'environnement extérieur. Ainsi, les données de diagnostic pourront être envoyées par des canaux adéquats pour alimenter le moteur de diagnostic. De même, les sorties du framework, concernant les résultats des diagnostics, pourront utilisées des canaux menant du framework vers l'extérieur.

Les canaux d'entrées doivent dériver, tous, de la classe *InputConnector* (voir figure 5.2). De même, les canaux de sorties doivent dériver de la classe *OutputConnector*.
Ce module fournit par défaut des connecteurs d'entrées/sorties vers des fichiers textes.

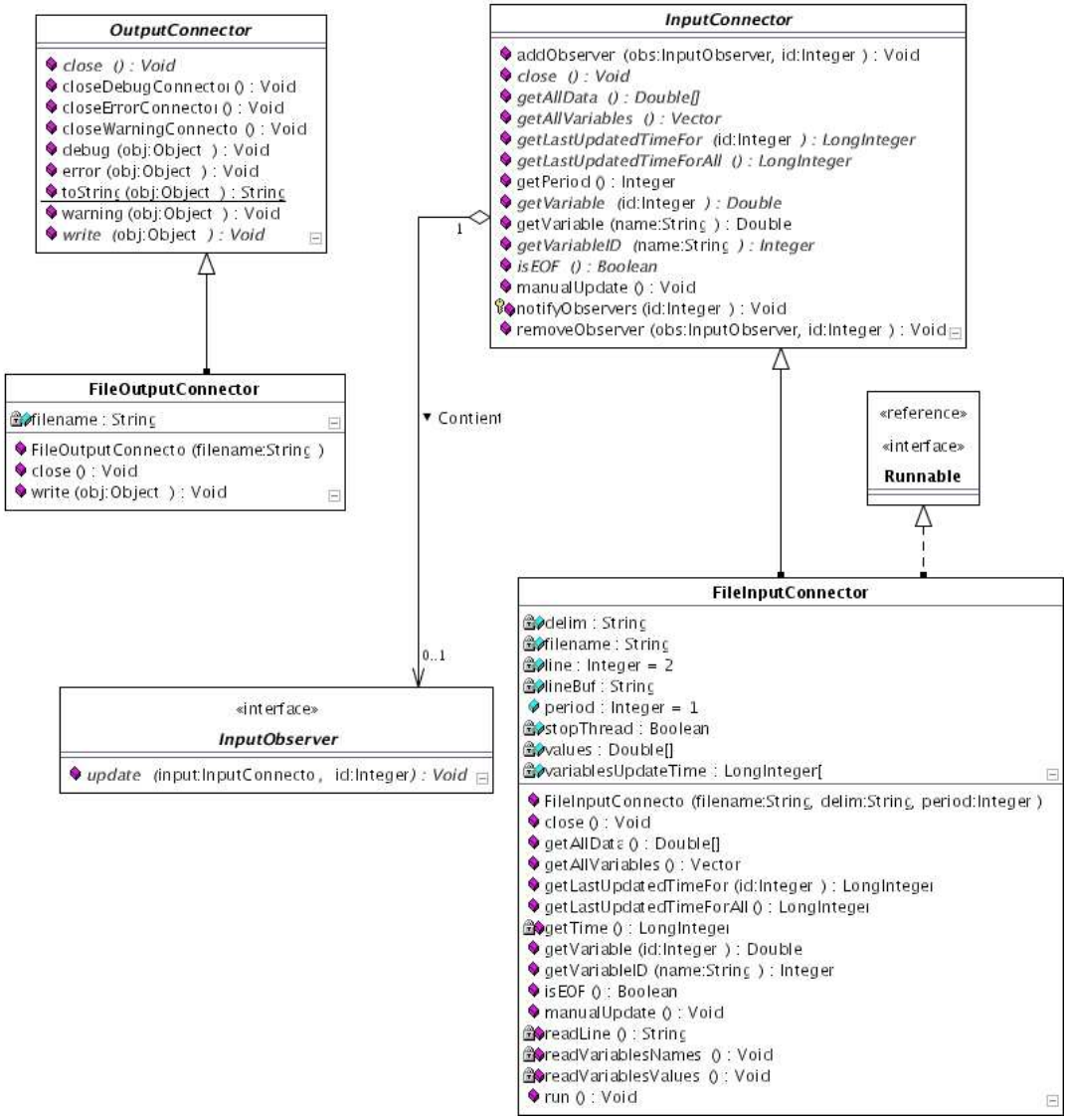

FIG. 5.2: Diagramme UML de classes pour le package io

5.1.3 Diagramme de classes pour le package model

Le module « model » permet de modéliser le système cible du diagnostic. Le système doit être décrit en fournissant les informations suivantes :

- une liste de variables décrites par ses propriétés telles que :
  - le nom de la variable
  - indication sur son observabilité

- coût de l'observation

• une liste de composants avec les informations suivantes :

- le nom du composant
- les variables d'entrée du composant
- les variables de sortie du composant
- le nombre de fautes possibles pour ce composant

• une liste de relations de redondances analytiques avec les informations suivantes :

- le support de la RRA (liste de composants engagés dans la RRA)
- la liste de variables utilisées dans le calcul de la RRA

• un programme permettant de calculer les résidus de chaque RRA en fonction d'une observation

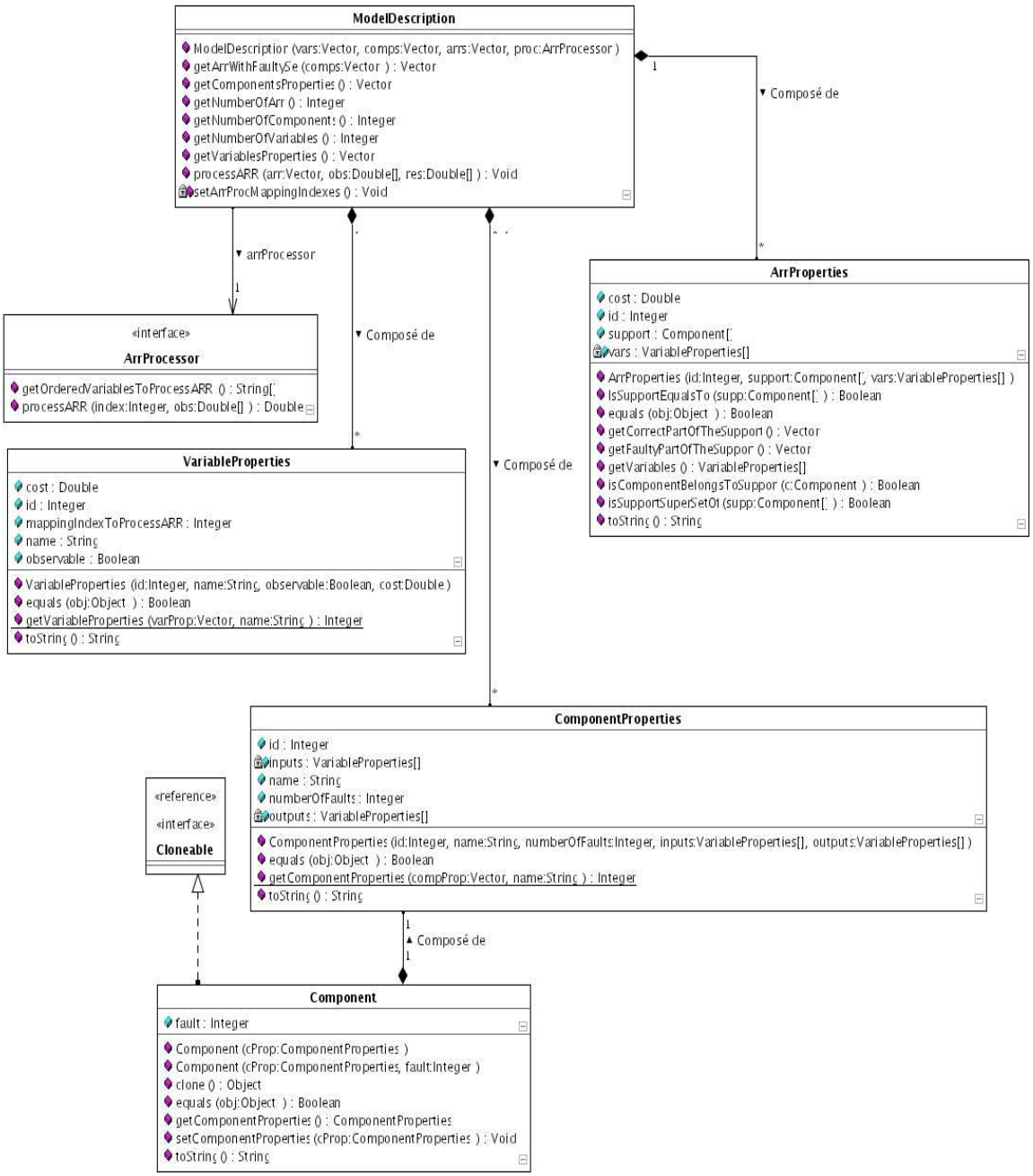

FIG. 5.3: Diagramme UML de classes pour le package model

5.1.4 Diagramme de classes pour le package reseda

Le module « reseda » implémente les diverses techniques de diagnostic décrites dans le chapitre 4.

L'implémentation et l'ajout de nouvelles techniques de diagnostic s'effectuent en implémentant la classe interface *AbstractDiagnosticProcessor* (voir figure 5.4).

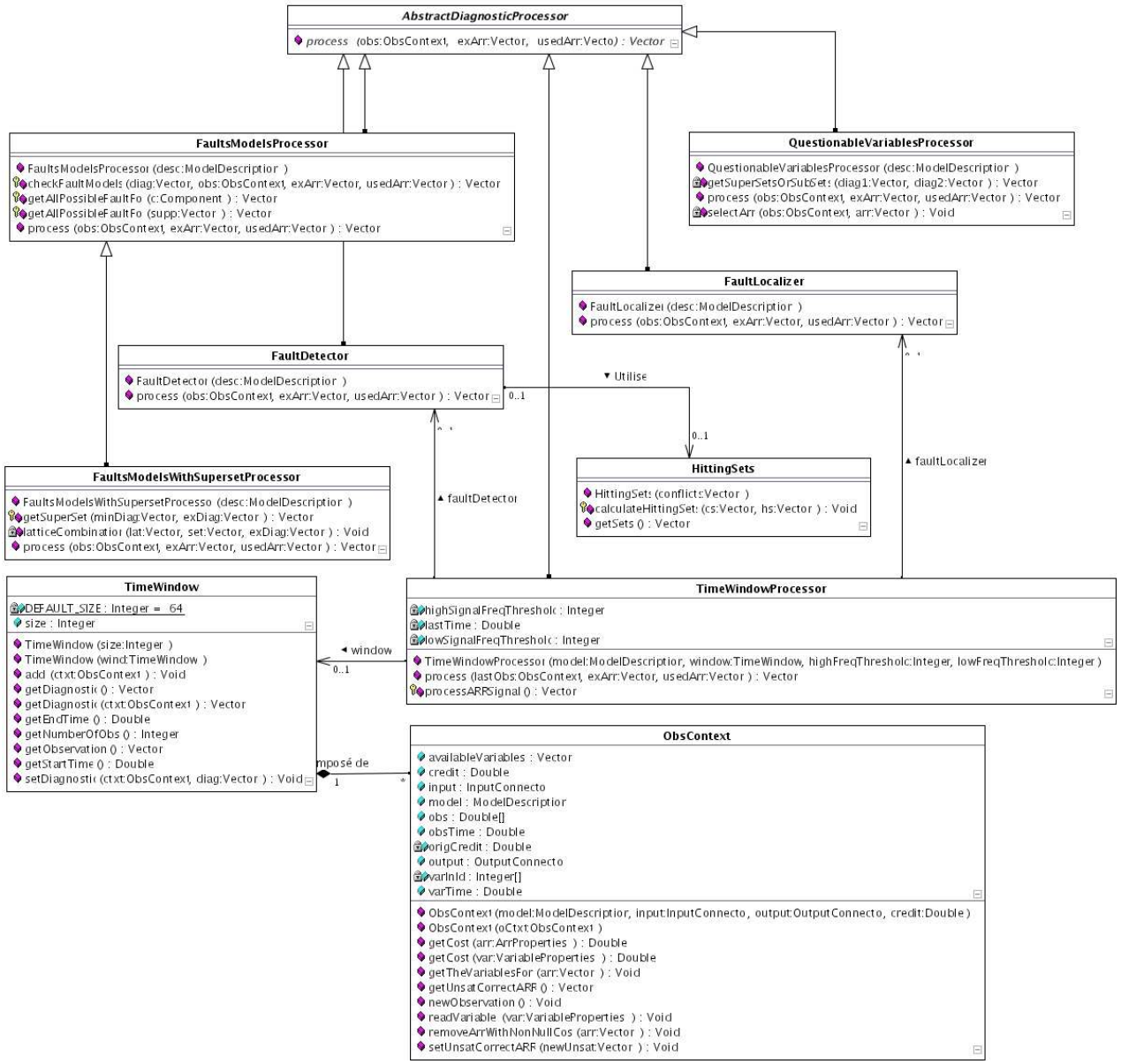

FIG. 5.4: Diagramme UML de classes pour le package reseda

5.1.5 Diagramme de classes pour le package util

Le module « util » fournit des fonctions permettant de faciliter l'utilisation du framework. Ainsi, les fonctions suivantes sont offertes :

- un programme « lanceur » permettant de démarrer le framework en utilisant un fichier de configuration (voir annexe B). Ce fichier de configuration fournit les informa-

tions suivantes :
- le nom du fichier de description du modèle
- les noms des fichiers d'entrées/sorties
- une séquence de techniques de diagnostics à utiliser pour la session de diagnostic
- divers paramètres pour les techniques de diagnostics

• une classe permettant de construire le modèle d'un système à partir d'un fichier de description de modèle (voir annexe C). Ce fichier contient les informations suivantes :
- une liste de variables utilisées par le système
- une liste de composants intervenant dans le système
- une liste de RRA
- le nom du programme chargé de calculer les RRA

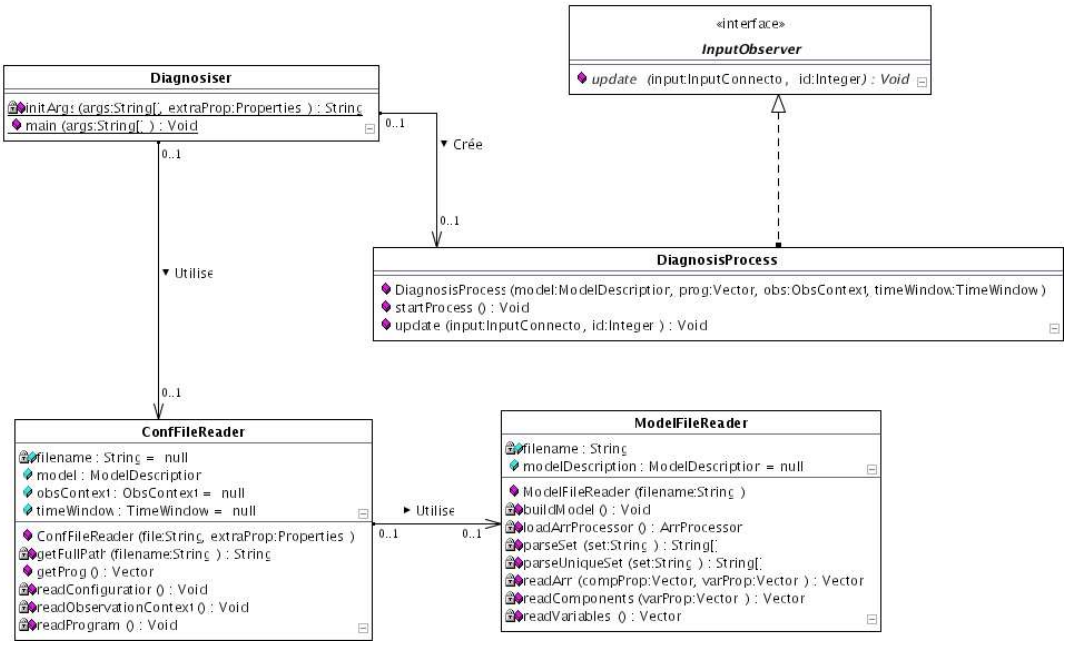

FIG. 5.5: Diagramme UML de classes pour le package util

5.2 Application

Les informations concernant les modèles de fautes du système GDPA n'étant pas disponible, le framework a été appliqué sur le système Polybox (voir figure 4.1) pour le besoin des tests et validation.

5.2.1 Polybox

Le framework ayant été paramétré avec les informations suivantes :

- Système : Polybox
- Nombre de fautes possible par composant : 3 (mode correct + 2 modes de fautes :$sortie = entrée_1$ et $sortie = entrée_2$)
- Séquence de diagnostic :
  1. détecteur de défaillance (voir section 4.1)
  2. localisateur de fautes (voir section 4.2)
  3. identificateur basé sur les modèles de fautes (voir section 4.3

Les résultats sur une série de dix observations sont donnés par le tableau 5.1.

Fautes injectées	**Fautes trouvées sans exploration des sur-ensembles**	**Fautes trouvées avec exploration des sur-ensembles**
$\{M1_2, M2_1, M3_1\}$	$\{\}$	$\{\{M1, M2, M3\}\}$
$\{M1_1\}$	$\{\{M1\}\}$	$\{\{M1\}\}$
$\{M1_2, M2_2, A2_2\}$	$\{\}$	$\{\{M1, M2, A2\}\}$
$\{M1_1, M2_1, M3_2, A1_1, A2_1\}$	$\{\}$	$\{\{M1, M2, A1\}\}$
$\{M1_2, M2_1, A1_1, A2_1\}$	$\{\}$	$\{\{M1, M2, A1, A2\}\}$
$\{M1_2, M2_2, M3_1, A2_1\}$	$\{\{M2, A2\}\}$	$\{\{M2, A2\}\}$
$\{M1_1, M3_2, A1_2, A2_2\}$	$\{\{M3\}\}$	$\{\{M3\}\}$
$\{M1_2, M3_2\}$	$\{\{M3\}\}$	$\{\{M3\}\}$
$\{M1_1, M2_1, M3_2\}$	$\{\}$	$\{\{M1, M2, M3\}\}$
$\{A1_1, A2_2\}$	$\{\{A1, A2\}\}$	$\{\{A1, A2\}\}$

TAB. 5.1: Résultat d'une séquence de diagnostic sur le polybox

Nous pouvant noter, à partir du tableau 5.1, que dans la majorité des cas les diagnostics minimaux, générés lors de la phase de localisation, n'expliquent pas la défaillance. En effet, aucune combinaison de fautes pour ces diagnostics n'est cohérente avec les observations. Ainsi, l'identification, utilisant les modèles de fautes, renvoie un espace de diagnostic vide. Il est alors nécessaire d'explorer les sur-ensembles des diagnostics minimaux. Cette exploration fournit alors un diagnostic unique.

Toutefois, on peut noter que les diagnostics finaux obtenus, après exploration des sur-ensembles de diagnostics minimaux, ne correspondent pas, tout à fait, aux fautes injectées. Cependant, les diagnostics correspondent toutefois à un sous-ensemble de l'ensemble des fautes injectées.
Il existe deux explications pour la remarque précédente :

- Le diagnostic est un diagnostic minimal. Ainsi, ce diagnostic est le plus petit ensemble expliquant la défaillance. Or, pour le polybox, il existe des possibilités de court-circuitage de certains composants. Ainsi, si le composant $A1$ est en mode de faute 2, alors le composant $M1$ est court-circuité (isolé) du système. L'unique moyen de détecter une telle faute est alors d'avoir accès à la variable de sortie du composant court-circuité (dans le dernier cas, il s'agit de la variable x).

• Certaines valeurs pour les observations peuvent masquer le comportement fautif d'un composant. Prenons le cas du composant $M1$, si la première entrée de ce composant est égale à 0 ($a = 0$) alors les comportements de ce composant en mode correct et en mode fautif 1 ($x = a$) sont identiques. La discrimination est alors impossible pour ce composant.

5.2.2 GDPA

Renault n'a pas pu fournir les informations concernant les comportements fautifs des composants. De plus, à ce jour, les plateformes matérielles n'étant pas disponibles et aucune donnée de tests (observations) n'ayant été fournie, le framework n'a pas pu être testé.
Toutefois, le framework a été utilisé pour produire un code pouvant être embarqué. Certaines parties du framework ont été retirées et d'autres optimisées afin de satisfaire les contraintes temps réels de l'environnement matérielles.

Ainsi, la séquence de diagnostic embarquée consiste en :

1. une fenêtre temporelle
2. une étape de détection décrite dans la section 4.1
3. une étape de localisation utilisant l'approche FDI [1] décrite dans la section 3.1.2 avec la matrice d'incidence donnée par le tableau 3.3

[1] Pour des raisons temps critiques, l'approche DX définie à la section 4.2 et implémentée dans le framework n'a pas été utilisée

Chapitre 6

Conclusion

Dans le cadre de ce stage, les techniques de bases du diagnostic ont été étudiées. Cette étude a permis de révéler des points qui nécessitent des améliorations pour l'application au domaine automobile qui se trouve être le cadre applicatif du stage.
Ainsi, parmi les principaux points nécessitant une amélioration figurent la phase de détection et la phase d'identification.

L'étape de détection a pour objectif de surveiller le système afin de détecter les anomalies. Dans le but de minimiser les « fausses alarmes », nous avons proposé, dans le cadre de ce stage, un algorithme de détection utilisant une fenêtre temporelle.

L'étape d'identification, quant à elle, est chargée d'affiner l'espace d'explications (diagnostics) d'une anomalie détectée.
Pour expliquer l'anomalie, l'étape de localisation génère un espace de diagnostic. Cet espace, pouvant être, plus ou moins vaste, une étape d'identification est nécessaire pour isoler le diagnostic le plus pertinent.
Ainsi, nous avons proposé un algorithme d'identification basé sur l'utilisation des modèles de fautes.

Outre les différentes améliorations proposées, nous avons développé des algorithmes intégrant, au processus de diagnostic, des informations spécifiques au véhicule Renault telles que les variables interrogeables.

Le stage a, finalement, abouti à la réalisation d'un environnement de diagnostic intégrant les différentes techniques présentées.

Le processus de diagnostic pourrait être amélioré en effectuant, à priori, une étude de diagnosticabilité.
Une des techniques de diagnosticabilité consisterait à convertir le modèle quantitatif, utilisé dans le cadre de ce stage, en un modèle qualitatif. Ainsi, cette étude pourrait mettre à jour des composants à incriminer pour certains types d'anomalies [Dressler and Struss, 2003].

Bibliographie

R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computer Surveys*, 24(3), September 1992.

M-O. Cordier, P. Dague, M. Dumas, F. Lévy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès. A comparative analysis of ai and control theory approaches to model-based diagnosis. In *Proc. of the 14th ECAI*, pages 134–140, Berlin, 2000.

J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32 : 97–130, 1987.

J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *Proc. of the 11th IJCAI*, pages 1324–1330, Detroit, MI, 1989.

O. Dressler and P. Struss. A toolbox integrating model-based diagnosability analysis and automated generation of diagnostics. In *Proc. of the 14th International Workshop on Principles of Diagnosis*, Washington, 2003.

R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *Artificial Intelligence*, 41 :79–88, 1989.

Li Lin and Yunfei Jiang. The computation of hitting sets : review and new algorithms. *Inf. Process. Lett.*, 86(4) :177–184, 2003. ISSN 0020-0190.

O. Raiman. The alibi principle. *Readings in model-based diagnosis*, pages 66–70, 1992.

R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32 :57–95, 1987.

P. Struss and O. Dressler. "physical negation" : Integrating fault models into the general diagnostic engine. In *Proc. of the 11th IJCAI*, pages 1318–1323, Detroit, MI, 1989.

Annexe A

Rapport d'étude de la fonction GDPA

Sujeevan ASEERVATHAM

La fonction GDPA -Gestion de la Décélération Proche de l'Arrêt- est une fonction de confort permettant d'éviter les « à-coups » plus ou moins brutaux dus à l'arrêt du véhicule, plus précisément lors du passage d'une vitesse non nulle à la vitesse nulle. Afin d'éviter ces « à-coups », le conducteur doit relâcher la pédale de frein au fur et à mesure que la voiture s'immobilise. Cette action nécessite, de la part du conducteur, une attention particulière. Ainsi, le GDPA se propose de décharger le conducteur de cette action en la prenant entièrement en charge.

A.1 Principe théorique du GDPA

Le principe du GDPA est de suivre une courbe de décélération permettant de passer progressivement d'une accélération initiale constante à une accélération finale immobilisant la voiture tout en maintenant une augmentation de la distance d'arrêt constante par rapport à la distance d'arrêt sans adoucissement.
La courbe d'accélération sera linéairement dépendant du temps, la dérivée temporelle de l'accélération sera donc constante pendant toute la phase d'adoucissement.
Cette courbe est caractérisée par trois paramètres :

- une accélération initiale γ_C,
- une accélération finale γ_F,
- une distance X - garantie par $\frac{d\gamma}{dt} = w = cst$ - telle que $DA_{adoucie} = DA_{ref} + X$.

L'accélération finale γ_F est un paramètre de réglage qui suit une loi dépendante de :

- l'accélération initiale voulue par le conducteur γ_C,
- la distance DA_{ref},
- (la perturbation d'accélération tendant à accélérer le véhicule donc positif),
- (le freinage demandé à l'aide de la pédale),
- (la vitesse d'appui sur la pédale).

Les paramètres entre parenthèse étant des données utilisées pour tenir compte des contraintes d'environnement.

De même, l'augmentation de la distance d'arrêt X suit une loi dépendante de :
- l'accélération initiale voulue par le conducteur γ_C,
- la distance DA_{ref},
- (la vitesse d'appui sur la pédale).

A l'aide de cette courbe, il est possible de déterminer la décélération nécessaire pour stopper le véhicule en fonction de la vitesse par la formule :

$$\gamma(t) = -\sqrt{\gamma_F^2 + 2V(t) \times w},\ avec\ w = \frac{d\gamma}{dt} = cst$$

A.1.1 Condition d'adoucissement

L'adoucissement de la décélération par le GDPA ne débute que lorsque les différentes conditions suivantes sont satisfaites :
- l'utilisation du GDPA a été activée au préalable par le conducteur,
- le véhicule est en mode nominal - aucune défaillance détectée -,
- le conducteur appuie sur la pédale de frein - freinage en cours -,
- le freinage demandé par le conducteur est inférieur ou égale à P_S - freinage seuil au dessus duquel on considère être en freinage « d'urgence »-,
- la décélération finale est inférieure à la décélération initiale : $\mid \gamma_F \mid \leq \mid \gamma_C \mid$,
- la vitesse du véhicule est inférieure ou égale à V_S - vitesse seuil en dessous de laquelle on considère que le véhicule est proche de l'arrêt -,
- l'ABS n'est pas en cours d'utilisation.

A.1.2 Algorithme de calcul

▷ $V_0 = V(t_0)$ et $\gamma_C = \gamma(t_0)$

- Calcul de la distance d'arrêt sans adoucissement (à γ_C constant) :

$$DA_{ref} = -\frac{V_0^2}{2\gamma_C}$$

- Calcul de γ_F et de X
- Si conditions d'adoucissement - cf. A.1.1 - non satisfaites alors :
  - $\gamma(t)$ = décélération demandée par le conducteur
- Sinon
  - Calcul du jerk :

$$w = \sqrt{\frac{1}{24X\gamma_C}(\gamma_F - \gamma_C)^3(\gamma_C + 3\gamma_F)}$$

  - Calcul de la décélération adoucie

$$\gamma(t) = -\sqrt{\gamma_F^2 + 2V(t) \times w}$$

A.2 Implémentation au niveau du véhicule

A.2.1 Calcul de la distance d'arrêt DA_{ref}

$$\begin{aligned} DA_{ref} &= DA_{parcourue} + DA_{restante} \\ &= \int_{T_0} \mid v \mid dt + \frac{1}{2}\frac{v^2}{max(|\gamma_{obs}|;0.05))} \end{aligned}$$

T_0 étant le dernier instant où :

- soit le signe de la vitesse à changé
- soit il y a eu appui ou relâchement de la pédale de frein

A.2.2 Calcul de l'accélération finale γ_F et de l'augmentation de la distance d'arrêt X

1. Lors d'un appui rapide sur la pédale de frein - arrêt rapide souhaité - l'adoucissement devra être faible afin de ne pas allonger la distance d'arrêt, l'accélération finale devra donc être proche de l'accélération voulue par le conducteur :

$$\begin{aligned} v_{pédale} &:= \max_{\text{depuis l'appui sur la pédale}} \frac{dX_{pédale}}{dt} \\ v_{pédale} &\geq 0.150 \Rightarrow \begin{cases} \gamma_F := 0.5\gamma_C \\ X := 0.02 \end{cases} \\ v_{pédale} &< 0.150 \Rightarrow \begin{cases} \gamma_F := 0.4\gamma_C \\ X := 0.02 \end{cases} \end{aligned}$$

2. En fonction de la longueur de la distance d'arrêt - sans adoucissement -, on calcule l'accélération finale, en sachant que pour une distance d'arrêt courte l'accélération finale doit être proche de l'accélération conducteur et l'augmentation de la distance d'arrêt plus faible - adoucissement plus faible - que pour une longue distance - adoucissement plus important -.

$$\begin{aligned} DA_{ref} &\leq 1.2m \Rightarrow \gamma_F := \gamma_C \\ DA_{ref} &\geq 10m \Rightarrow \begin{cases} \gamma_F := 0.3 \times \gamma_C \\ X := 0.15 \end{cases} \\ gain &:= \frac{1-0}{10-1.2}(DA_{ref} - 1.2) \\ &= \frac{DA_{ref}-1.2}{10-1.2} \\ \gamma_F &:= (1 - gain)\gamma_F + gain \times 0.3\gamma_C \\ X &:= (1 - gain)X + gain \times 0.15 \end{aligned}$$

3. Lors de freinage important, l'accélération finale doit diminuer pour atteindre la valeur de l'accélération conducteur. Ainsi, l'adoucissement va progressivement être désactivé lors d'un appui important sur la pédale de freinage - supérieur à un certain seuil -.

$$\begin{array}{lcl} \mid \gamma_{pédale} \mid & \leq & 5 \Rightarrow gain := 0 \\ \mid \gamma_{pédale} \mid & \geq & 6 \Rightarrow gain := 1 \\ gain & := & \frac{1-0}{6-5}(\mid \gamma_{pédale} \mid -5) + 0 \\ & = & \gamma_{pédale} - 5 \end{array}$$

$$\gamma_F := \gamma_F + gain \times \gamma_C$$

4. Lors de la montée, l'accélération finale doit être inférieur à la perturbation de la pente qui freine le véhicule - multipliée par un coefficient de sécurité -, sinon le véhicule aura tendance à repartir en arrière obligeant le conducteur à ré-appuyer, plus fortement, sur la pédale de frein :

$$\gamma_F := \min(\gamma_F, (2.3 \times \gamma_{perturbation}))$$

5. L'accélération finale doit être supérieure à l'accélération voulue par le conducteur pour qu'il y ait adoucissement sinon on applique l'accélération voulue par le conducteur :

$$\gamma_F := \max(\gamma_F, \gamma_C)$$

A.2.3 Calcul du jerk w

$$w := \frac{d\gamma_{consigne}}{dt} = \sqrt{\frac{1}{24X\gamma_C}(\gamma_F - \gamma_C)^3(\gamma_C + 3\gamma_F)}$$

A.2.4 Calcul de l'accélération adoucie $\gamma_{consigne}$

L'accélération adoucie doit être supérieure à l'accélération voulue par le conducteur :

$$\gamma_{consigne} = \max(-\sqrt{\gamma_F^2 + 2vw}; \gamma_{voulue})$$

Or, il existe un temps de retard entre la commande de freinage et l'accélération réelle du véhicule. Ce retard est essentiellement dû à deux facteurs :

- l'échantillonnage du calculateur dont la période est de $T_e = 0.02s$,
- le temps de réponse des actionneurs : $T_{FREINAGE} = \frac{0.150}{3}s$

Pour le retard dû à l'échantillonnage, il suffit de calculer $\gamma_{consigne}(v(t+T_e))$ au lieu de $\gamma_{consigne}(v)$, d'où :

$$\begin{array}{lcl} \gamma_{consigne}(v(t+T_e)) & := & -\sqrt{\gamma_F^2 + 2v(t+T_e) \times w} \\ & = & -\sqrt{\gamma_F^2 + 2w(v(t) + T_e \times \gamma_{obs})} \\ & = & -\sqrt{\gamma_F^2 + 2w(v + T_e\gamma_{obs})} \end{array}$$

Quant au retard dû aux actionneurs, nous devons calculer $\gamma_{consigne}(1+\tau s)$ à la place de $\gamma_{consigne}(v)$.

$$\begin{aligned}
\gamma_{consigne}(1+\tau s) &:= \gamma_{consigne} + \tau\frac{d\gamma_{consigne}}{dt} \\
&= \gamma_{consigne} + \tau\frac{d\gamma_{consigne}}{dv}\frac{dv}{dt} \\
&= \gamma_{consigne} + \tau\frac{d\gamma_{consigne}}{dv}\gamma_{obs} \\
&= \gamma_{consigne} + \tau\frac{w}{\gamma_{consigne}}\gamma_{obs} \\
&= \gamma_{consigne} + T_{FREINAGE}\frac{w}{\gamma_{consigne}}\gamma_{obs} \text{ car } \tau = T_{FREINAGE}
\end{aligned}$$

D'où l'accélération adoucie anticipée :

$$\gamma_{consigne} := -\sqrt{\gamma_F^2 + 2w(v + T_e\gamma_{obs})} + T_{FREINAGE}\frac{w}{-\sqrt{\gamma_F^2 + 2w(v + T_e\gamma_{obs})}}\gamma_{obs}$$

On veillera à ce que le dénominateur ne soit jamais nul et que l'accélération adoucie ne soit jamais supérieure à l'accélération finale. De plus, l'accélération adoucie ne devra pas être inférieure à l'accélération voulue par le conducteur. D'où :

$$\begin{aligned}
\gamma_{consigne} &:= -\sqrt{\gamma_F^2 + 2w(v + T_e\gamma_{obs})} + T_{FREINAGE}\frac{w}{-\max(\sqrt{\gamma_F^2+2w(v+T_e\gamma_{obs})};0.05)}\gamma_{obs} \\
\gamma_{consigne} &:= \min(\gamma_{consigne};\gamma_F) \\
\gamma_{consigne} &:= \max(\gamma_{consigne};\gamma_C)
\end{aligned}$$

A.2.5 Calcul de la commande de freinage

La commande de freinage doit tenir compte, non seulement, de l'accélération adoucie mais aussi des perturbations qui ont tendances à accélérer ou à freiner le véhicule. De plus, lors de l'arrêt du véhicule, la commande de freinage doit tendre et atteindre la valeur de la décélération demandée par le conducteur au niveau de la pédale de frein. La valeur de la commande peut donc être obtenue par la formule suivante :

$$\begin{cases}
t = 0 \text{ si le véhicule est en mouvement} \\
t = \int_{T_0}\frac{1}{T_C}dt, T_0 \text{ étant le dernier instant où le véhicule est passé du mouvement à l'arrêt} \\
\text{si } t > 1 \Rightarrow t = 1
\end{cases}$$

$$commande = (-\gamma_{consigne} + \gamma_{perturbation}) \times (1-t) + t\times \mid \gamma_{pédale} \mid$$

A.2.6 Désactivation et limitation de la fonction GDPA

Sous certaines conditions, la fonction GDPA doit être désactivée, l'accélération appliquée au véhicule sera alors égale à la valeur de la décélération demandée par le conducteur au niveau de la pédale de frein.
Ces conditions sont les suivantes :

- la fonction GDPA a été désactivée par le conducteur,
- le conducteur n'a pas appuyé sur la pédale de frein - arrêt non sollicité -,
- l'ABS s'est déclenché,
- l'accélération finale est inférieure à l'accélération initiales - $\mid \gamma_F \mid \geq \mid \gamma_C \mid$ -. L'adoucissement est alors inutile,
- la vitesse est supérieure à un certain seuil - $\mid v \mid > \frac{20}{3.6} m.s^{-1}$ -, le véhicule n'est donc pas proche de l'arrêt,
- la demande de décélération au niveau de la pédale est supérieure à un certain seuil - $\mid \gamma_{pédale} \mid > 6N.Kg^{-1}$ -, le conducteur peut alors être en freinage d'urgence.

De plus, on limitera la fonction GDPA dans les cas suivants :

- la commande de freinage adoucie est supérieure ou égale à la valeur de la décélération demandée par le conducteur au niveau de la pédale de frein - $commande \geq \mid \gamma_{pédale} \mid$-. L'accélération appliquée sera alors celle demandée par le conducteur,
- la commande de freinage est inférieure ou égale à un certain seuil - $commande \leq x \times \mid \gamma_{pédale} \mid, x \in [0; 1]$ -. De même, l'accélération appliquée sera celle demandée par le conducteur.

Annexe B

Fichier de configuration du framework pour le Polybox

```
##
# A simple configuration file for diagnosis
# experimentation purpose on the polybox system.
#
# Author: Sujeevan ASEERVATHAM
###


################################################
#
# model description file
file.model = polybox.model


################################################
#
#input definition

# input data file name or stdin for standard input
file.input = data10.csv
# field separator
file.input.delim = ;

# period of the variables' update
file.input.refresh_period = -1


################################################
#
# Output definition
#
# stdout = standard output
# stderr = standard error

# results output
file.output = diagRes.txt


#message output
# defout = default output (= file.output)
# stdout = standard output
# stderr = standard error
# none = no output
# filename

file.output.debug = defout
file.output.warning = defout
file.output.error = defout



################################################
#
# Diagnosis program sequence

# available diagnostic components
# debug = output of the current diagnostic
# FaultDetector, FaultLocalizer, FaultsModelsProcessor, FaultsModelsWithSupersetProcessor
# QuestionableVariablesProcessor, TimeWindowProcessor

diagnostic_program = debug; FaultDetector; debug; FaultLocalizer; FaultsModelsWithSupersetProcessor; debug



################################################
```

```
#
# Parameter used by QuestionableVariablesProcessor and other
observation.credit = 40


##############################################
#
# Parameters used to define time window
TimeWindow.size = 10
# Activation threshold
TimeWindowProcessor.highFreq = 1
# Tolerance threshold
TimeWindowProcessor.lowFreq = 0
```

Annexe C

Fichier de description du modèle Polybox

```
###
# A system description of the Polybox system.
# This is an automatically generated code.
#
# Author: Sujeevan ASEERVATHAM
###


##############################################################################

###
# Definition of the system's variables
###

variables = {a; b; c; d; e; f; g; x; y; z }

variables.a.observable=true
variables.a.observation_cost = 0

variables.b.observable = true
variables.b.observation_cost = 0

variables.c.observable = true
variables.c.observation_cost = 0

variables.d.observable = true
variables.d.observation_cost = 0

variables.e.observable = true
variables.e.observation_cost = 0

variables.f.observable = true
variables.f.observation_cost = 0

variables.g.observable = true
variables.g.observation_cost = 0

variables.x.observable = true
variables.x.observation_cost = +10

variables.y.observable = true
variables.y.observation_cost = +15

variables.z.observable = true
variables.z.observation_cost = +15



##############################################################################

###
# Definition of the system's components
###

components = {M1; M2; M3; A1; A2}

components.M1.inputs = {a; c}
components.M1.outputs = {x}
components.M1.number_of_faults = 3

components.M2.inputs = {b; d}
components.M2.outputs = {y}
components.M2.number_of_faults = 3
```

```
components.M3.inputs = {c; e}
components.M3.outputs = {z}
components.M3.number_of_faults = 3

components.A1.inputs = {x; y}
components.A1.outputs = {f}
components.A1.number_of_faults = 3

components.A2.inputs = {y; z}
components.A2.outputs = {g}
components.A2.number_of_faults = 3



################################################################################

###
# Definition of the system's ARRs
###

arr.program = PolyboxModel


arr.0.support = {M1; M2; A1}
arr.0.faults = {0; 0; 0}
arr.0.variables = {a; b; c; d; f}

arr.1.support = {M2; M3; A2}
arr.1.faults = {0; 0; 0}
arr.1.variables = {b; c; d; e; g}

arr.2.support = {M1; M3; A1; A2}
arr.2.faults = {0; 0; 0; 0}
arr.2.variables = {a; c; e; f; g}

arr.3.support = {M1}
arr.3.faults = {0}
arr.3.variables = {a; c; x}

arr.4.support = {M2}
arr.4.faults = {0}
arr.4.variables = {b; d; y}

arr.5.support = {M3}
arr.5.faults = {0}
arr.5.variables = {c; e; z}

arr.6.support = {A1}
arr.6.faults = {0}
arr.6.variables = {f; x; y}

arr.7.support = {A2}
arr.7.faults = {0}
arr.7.variables = {g; y; z}

arr.8.support = {M1; M2; A1}
arr.8.faults = {1; 0; 0}
arr.8.variables = {a; b; d; f}

arr.9.support = {M1; M3; A1; A2}
arr.9.faults = {1; 0; 0; 0}
arr.9.variables = {a; c; e; f; g}

arr.10.support = {M1}
arr.10.faults = {1}
arr.10.variables = {a; x}

arr.11.support = {M1; M2; A1}
arr.11.faults = {2; 0; 0}
arr.11.variables = {b; c; d; f}

arr.12.support = {M1; M3; A1; A2}
arr.12.faults = {2; 0; 0; 0}
arr.12.variables = {c; e; f; g}

arr.13.support = {M1}
arr.13.faults = {2}
arr.13.variables = {c; x}

arr.14.support = {M1; M2; A1}
arr.14.faults = {0; 1; 0}
arr.14.variables = {a; b; c; f}

arr.15.support = {M2; M3; A2}
arr.15.faults = {1; 0; 0}
arr.15.variables = {b; c; e; g}

arr.16.support = {M2}
arr.16.faults = {1}
arr.16.variables = {b; y}

arr.17.support = {M1; M2; A1}
arr.17.faults = {1; 1; 0}
arr.17.variables = {a; b; f}
```

```
arr.18.support = {M1; M2; A1}
arr.18.faults = {2; 1; 0}
arr.18.variables = {b; c; f}

arr.19.support = {M1; M2; A1}
arr.19.faults = {0; 2; 0}
arr.19.variables = {a; c; d; f}

arr.20.support = {M2; M3; A2}
arr.20.faults = {2; 0; 0}
arr.20.variables = {c; d; e; g}

arr.21.support = {M2}
arr.21.faults = {2}
arr.21.variables = {d; y}

arr.22.support = {M1; M2; A1}
arr.22.faults = {1; 2; 0}
arr.22.variables = {a; d; f}

arr.23.support = {M1; M2; A1}
arr.23.faults = {2; 2; 0}
arr.23.variables = {c; d; f}

arr.24.support = {M2; M3; A2}
arr.24.faults = {0; 1; 0}
arr.24.variables = {b; c; d; g}

arr.25.support = {M1; M3; A1; A2}
arr.25.faults = {0; 1; 0; 0}
arr.25.variables = {a; c; f; g}

arr.26.support = {M3}
arr.26.faults = {1}
arr.26.variables = {c; z}

arr.27.support = {M1; M3; A1; A2}
arr.27.faults = {1; 1; 0; 0}
arr.27.variables = {a; c; f; g}

arr.28.support = {M1; M3; A1; A2}
arr.28.faults = {2; 1; 0; 0}
arr.28.variables = {c; f; g}

arr.29.support = {M2; M3; A2}
arr.29.faults = {1; 1; 0}
arr.29.variables = {b; c; g}

arr.30.support = {M2; M3; A2}
arr.30.faults = {2; 1; 0}
arr.30.variables = {c; d; g}

arr.31.support = {M2; M3; A2}
arr.31.faults = {0; 2; 0}
arr.31.variables = {b; d; e; g}

arr.32.support = {M1; M3; A1; A2}
arr.32.faults = {0; 2; 0; 0}
arr.32.variables = {a; c; e; f; g}

arr.33.support = {M3}
arr.33.faults = {2}
arr.33.variables = {e; z}

arr.34.support = {M1; M3; A1; A2}
arr.34.faults = {1; 2; 0; 0}
arr.34.variables = {a; e; f; g}

arr.35.support = {M1; M3; A1; A2}
arr.35.faults = {2; 2; 0; 0}
arr.35.variables = {c; e; f; g}

arr.36.support = {M2; M3; A2}
arr.36.faults = {1; 2; 0}
arr.36.variables = {b; e; g}

arr.37.support = {M2; M3; A2}
arr.37.faults = {2; 2; 0}
arr.37.variables = {d; e; g}

arr.38.support = {M1; A1}
arr.38.faults = {0; 1}
arr.38.variables = {a; c; f}

arr.39.support = {A1}
arr.39.faults = {1}
arr.39.variables = {f; x}

arr.40.support = {M1; A1}
arr.40.faults = {1; 1}
arr.40.variables = {a; f}

arr.41.support = {M1; A1}
arr.41.faults = {2; 1}
arr.41.variables = {c; f}
```

```
arr.42.support = {M2; A1}
arr.42.faults = {0; 2}
arr.42.variables = {b; d; f}

arr.43.support = {M3; A1; A2}
arr.43.faults = {0; 2; 0}
arr.43.variables = {c; e; f; g}

arr.44.support = {A1}
arr.44.faults = {2}
arr.44.variables = {f; y}

arr.45.support = {M2; A1}
arr.45.faults = {1; 2}
arr.45.variables = {b; f}

arr.46.support = {M2; A1}
arr.46.faults = {2; 2}
arr.46.variables = {d; f}

arr.47.support = {M3; A1; A2}
arr.47.faults = {1; 2; 0}
arr.47.variables = {c; f; g}

arr.48.support = {M3; A1; A2}
arr.48.faults = {2; 2; 0}
arr.48.variables = {e; f; g}

arr.49.support = {M2; A2}
arr.49.faults = {0; 1}
arr.49.variables = {b; d; g}

arr.50.support = {M1; A1; A2}
arr.50.faults = {0; 0; 1}
arr.50.variables = {a; c; f; g}

arr.51.support = {A2}
arr.51.faults = {1}
arr.51.variables = {g; y}

arr.52.support = {M1; A1; A2}
arr.52.faults = {1; 0; 1}
arr.52.variables = {a; f; g}

arr.53.support = {M1; A1; A2}
arr.53.faults = {2; 0; 1}
arr.53.variables = {c; f; g}

arr.54.support = {M2; A2}
arr.54.faults = {1; 1}
arr.54.variables = {b; g}

arr.55.support = {M2; A2}
arr.55.faults = {2; 1}
arr.55.variables = {d; g}

arr.56.support = {A1; A2}
arr.56.faults = {2; 1}
arr.56.variables = {f; g}

arr.57.support = {M3; A2}
arr.57.faults = {0; 2}
arr.57.variables = {c; e; g}

arr.58.support = {A2}
arr.58.faults = {2}
arr.58.variables = {g; z}

arr.59.support = {M3; A2}
arr.59.faults = {1; 2}
arr.59.variables = {c; g}

arr.60.support = {M3; A2}
arr.60.faults = {2; 2}
arr.60.variables = {e; g}



################################################################################
```