RÉPUBLIQUE DU CAMEROUN
PAIX-TRAVAIL-PATRIE
Ministère de l'Enseignement Supérieur

REPUBLIC OF CAMEROON
PEACE– WORK–FATHELAND
Ministry of Higher Education

MÉMOIRE DE FIN D'ÉTUDES

VBD (Vic on-Board Diagnotic): Système embarqué temps réel d'aide au diagnostic des dysfonctionnements pour véhicules terrestres légers.

Soutenu en vu de l'obtention d'un Master en Génie Informatique et Systèmes:

Auteur :
M. Donald Patrice
NKOUAKEP DJAMPOU

Sous l'encadrement de :
Dr Christian FOTSING
M. Blaise TEGUIA

Année Académique 2014-2015

Dédicace

Je dédie ce travail: A mes parents

DJAMPOU Hilaire

et

NZINSOUE Amélie

Ceci est un pas vers l'accomplissement d'un acte encore plus grand

Remerciements

- Je rends premièrement grâce à Dieu le Tout Puissant pour m'avoir donné la force morale et physique afin d'achever ce mémoire.

- Mes remerciements s'adressent ensuite à ma famille, qui m'a aidé sur tous les plans et dont les mots ne sauraient exprimer ma reconnaissance, particulièrement à mon grand frère Fabrice DJAMPOU.
- Je remercie mon amie que j'affectionne tout particulierement Michèle TOUSSE, pour sa "présence" tout au long de la durée de ce projet.
- Je remercie la famille NONO, pour l'agréable séjour qu'elle m'a réservé durant mon stage à Douala.

- Je remercie le groupe GBEEC de Bangangté, pour leurs soutiens et conseils, durant toute la durée de ma formation.
- Je tiens à remercier aussi tous mes professeurs et le personnel académique et administratif de l'université des montagnes pour les connaissances acquises pendant ces 5ans et qui ont rendu ces années d'intenses efforts plus agréable.
- Mes remerciements aussi à mes camarades pour leur agréable collaboration afin de tendre ensemble vers l'excellence.

- C'est avec gratitude que j'exprime ma profonde reconnaissance à mon encadreur de stage, le docteur Christian FOTSING. J'ai beaucoup appris à son contact, tant sur le point de vue scientifique en matière de rigueur et d'organisation que sur le plan humain.
- Je tiens aussi à manifester ma profonde gratitude à monsieur Paul Guimezap, président fondateur de l'Institut Universitaire de la Côte (IUC), pour m'avoir offert l'opportunité de travailler sur un projet aussi innovant et répondant parfaitement à mes attentes.
- Enfin, sans donner une liste exhaustive ou sans m'embarquer dans un récit que seul les destinataires comprendraient et de peur d'en oublier certains, je tiens aussi à exprimer ma gratitude à tout le personnel de l'IUC, particulièrement à M. Blaise Teguia. Ceci pour la

bonne ambiance qu'ils ont su apporter tout au long de ma présence au sein de l'équipe du PRIE (Pôle Recherche Innovation et Entrepreneuriat) et pour les qualités personnelles de chacuns

- Mes remerciements à tous ceux dont les noms n'ont point été énoncés et qui ont donné de leur personne pour la réalisation de ce projet

Donald Patrice NKOUAKEP DJAMPOU

December 2015 Master II, GIS

Résumé

Les systèmes embarqués sont aujourd'hui quasi présents dans tous les véhicules récents, assurant ainsi la sécurité du conducteur et l'assistant par la même occasion tout au long de son trajet.
Dans le cadre de ce projet, nous nous intéresserons à la mise en œuvre d'un système embarqué intelligent, afin de prendre en compte de façon quasi automatique la gestion des pannes dans un véhicule.
Nous commençons par un état de l'art afin de recenser les systèmes existant qui pourront nous permettre de mener à bien ce travail. Il en ressort de là qu'il existe un standard international : OBD-II "On-Board Diagnostic 2"[27], qui est un système d'aide au diagnostic de bord orienté vers l'anti pollution, sur lequel les géants de la construction automobile greffent des modules optimisés et performants. Mais ils sont coûteux et propriétaires, non souvent adaptés au contexte local, la plupart de temps non configurable en fonction de nos préférences; autant de raisons qui justifient notre travail. Et par la suite nous présenterons les différentes technologie utilisées dans le cadre de ce projet.
Dans un deuxième temps, nous recenserons les pannes usuelles dans notre contexte "tropical", afin d'avoir une vue globale et détaillée du système d'auto-diagnostic que nous allons mettre en œuvre. Pour ce faire, nous avons mené une étude (par interview et revu de la littérature) qui a conduit à une analyse minutieuse des problèmes rencontrés par les conducteurs et garagistes dans notre contexte environnemental, tout en mettant un accent particulier sur leur fréquence d'apparition.

Dans un troisième temps, la phase conceptuelle du projet. Cette phase guidée par une approche dirigée par les modèles afin de garantir que notre implémentation sera le reflet de nos modèle. Elle a été essentiellement basée sur la modélisation AADL[AADL15] dont le principal avantage, est qu'il permet une conception unifiée au même moment de partie logicielle et de celle matérielle en plus d'être pris en charge par le simulateur Cheddar[6], qui nous a ainsi permis de valider les contraintes temps réel de notre système.
Par la suite, nous avons amorcé l'implémentation du système par une approche de co-design. L'approche matériel a été réalisée avec un microcontrôleur Arduino UNO connecté à un

smartphone par liaison sans fil Bluetooth, quant au coté logiciel sa mise au point a été faite avec l'exécutif temps réel ERIKA Enterprise.
L'idée générale est de mettre à disposition de l'utilisateur une plate-forme logiciel Open Source développée sous ERIKA Enterprise[24], la dite plate-forme sera flashée dans un microprocesseur Arduino Uno, connecté sur les capteurs adéquat de son véhicule ; dont les principales fonctionnalités sont :

- La supervision des fluides (carburant, huile et liquide de refroidissement) ;
- Décharge de la batterie ;
- Assurer la sécurité du conducteur par le contrôle de l'usure des freins et de la pression de l'air contenu dans les pneus;
- ...

Mots clés : Système embarqué, auto-diagnostic, AADL, Excutif Temps-Réel

Abstract

Embedded systems are now present in almost all recent vehicles, thus ensuring the safety of the driver and the assistant at the same time throughout its path.
As part of this project, we will focus on the implementation of a system embar- Intelligent qué, to take into account almost automatically fault management in a vehicle.
We begin with a state of the art to identify the existing systems that will enable us to carry out this work. It follows from this that there is a standard interface National: OBD-II "On-Board Diagnostic 2" [27], which is a diagnostic support system edge oriented anti pollution, on which the giants of the automotive grafted and optimized performance modules. But they are expensive and owners, not often adapted to the local context, most of the time not configurable according to our preferences; all reasons that justify our work. And then we will present the different technology used in this project.
Secondly, we will identify the usual failures in our context "tropical" to have a global and detailed view of the self-diagnostic system that we are going to implemented. To do this, we conducted a study (by interview and reviewed literature) which led to a careful analysis of the problems encountered by drivers and garages in our environmental context, while placing special emphasis on their frequency of occurrence.
Thirdly, the conceptual phase of the project. This guided phase by model-driven approach to ensure that our implementation will reflect our model. It was essentially based on AADL modeling [12] which the main advantage is that it allows a unified approach at the same time part software and the hardware in addition to being supported by Cheddar simulator [6] which has allowed us to validate the real-time constraints of our system.
Subsequently, we began the implementation of a system co-design approach. The hardware approach was performed with an Arduino UNO microcontroller connected to a smartphone by Bluetooth wireless link, as the software side its development was made with real-time executive ERIKA Enterprise.
The general idea is to provide the user a software platform Open Source developed under ERIKA Enterprise [23], said platform will be flashed in a Arduino Uno microprocessor connected to the appropriate sensors of the vehicle; whose Key features include:

- Supervision of fluids (fuel, oil and coolant);
- Discharge of the battery;
- To ensure driver safety by controlling brake wear and pressure the air contained in the tire;
- ...

Keywords: embedded system, self-diagnosis, AADL, excutif Real-Time

Table des matières

Liste des Figures

Introduction générale

Contexte

Afin de répondre aux nouveaux défis de sécurité et de confort, des systèmes avancés d'aide au conducteur ont été développés s'appuyant sur des nouvelles technologies et de nouveaux capteurs. Parmi ces technologies, les systèmes d'aide au diagnostic des pannes ont été implémentés et exploités à bord des véhicules. Une condition sine qua none pour garantir une bonne maintenance d'éventuelles pannes est d'avoir une localisation précise de ces dernières. Certaines prestations mécaniques ont besoin de connaître avec précision et de façon fiable le dysfonctionnement, à savoir : sa localisation, ses causes... Une localisation fiable et précise d'une panne est un composant essentiel pour tout conducteur et mécanicien.
Certaines voitures sont déjà équipées de modules d'assistance à la détection de dysfonctionnement. Mais ces derniers sont très récents (les modèles des dix dernières années) dans notre environnement (Cameroun), or notre flotte de véhicules est majoritairement constituée des modèles en deçà des années 2000. De surcroît la quasi-totalité de ces véhicules super équipés est hors de prix pour le camerounais véhiculé moyen. Bien qu'il soit vrai que de tels systèmes existent déjà sur des véhicules assez récents, le nôtre aura la particularité d'être :

- Peu coûteux ;
- Installable sur tous les véhicules quelles qu'en soient leurs années de sortie;
- Intégrera un module de statistique afin d'informer à l'utilisateur sur la mise en main, l'évolution et le changement de certains paramètres vitaux du véhicule ;
- Configurable (prendre en compte tout type de capteur) ;
- Open source, ceci dans l'optique de donner la possibilité aux utilisateurs avancés de l'étendre en fonction de leur besoin spécifique.
- Extensible.

Cadre du Mémoire

Le travail présenté dans ce mémoire est le résultat d'un stage de cinq mois à l'IUC (Institut Universitaire de la Côte). Il s'inscrit dans le cadre du projet de mise en place d'un véhicule (prototype) entièrement autonome et 100% camerounais mené par le Pôle Recherche Innovation Entrepreneuriat (PRIE). L'idée principale de ce projet consiste à partir d'une voiture basique de lui adjoindre tout les outils nécessaires, pour la rendre plus intelligente. Il s'agit principalement de concevoir et de développer un véhicule comme le montre la figure 1 donc les fonctionnalités seront les suivantes :

- Hybride (Biocarburant et énergie solaire);
- Géo localisation;
- Automatisme embarqué;
- Auto-diagnostic des pannes;
- ...

Fig. 1 Grandes branches du projet VIC

Problématique

Supposons que vous êtes au volant de votre voiture tout en écoutant paisiblement de la bonne musique et que tout à coup une panne se déclare; à partir de vos légères connaissances en

mécanique, vous ne réussirez pas à venir à bout de ce désagrément et vous vous dirigerez chez un mécanicien. Une fois arrivé chez ce dernier (qui utilise quelque fois des méthodes de diagnostic; son, vibration,... pas très formelles et qui s'acquièrent avec de l'expérience) vous n'avez pas d'autre choix que d'écouter et d'accepter sans riposter tout ce qu'il vous dira, quitte à ce qu'il soit véreux au point de simuler et de facturer quelques pannes en plus et il n'est pas exclut qu'il puisse vous surfacturer en dramatisant la panne initiale. Dans le meilleur des cas supposons que votre mécanicien soit très intègre, toute fois il peut arriver que déterminer de manière précise votre panne soit un exercice très compliqué pouvant durer plusieurs jours voir même des semaines. Ce manque d'outil de diagnostic adapté au contexte camerounais entraîne plusieurs pertes, dont nous pouvons recenser les suivantes (liste non exhaustive):

- Le mécanicien n'ayant pas pu maintenir le véhicule, car ne pouvant pas localiser la panne ne sera pas payé or il y a passé un bon bout de son temps.
- Le conducteur qui a perdu du temps pour ne pas être satisfait en fin de compte, bien sur sans faire allusion aux suppléments dus à ses moyens de transports auxiliaires qu'il a dû emprunter durant cette période.
- Le propriétaire du véhicule pourra être victime d'une surfacturation des frais de maintenance, ceci en fonction de l'humeur du mécanicien ;
- Le propriétaire du véhicule pourra être victime d'une surfacturation des frais de maintenance, ceci en fonction de l'humeur du mécanicien ;
- Toute fois il n'est pas exclu que le conducteur puisse sortir du garage avec de nouvelles pannes suite à un diagnostic et maintenance à tâtons de la part du mécanicien, comme ci les pannes initiales n'étaient pas déjà assez problématiques.

Nous pouvons ainsi constater que la détection précise et rapide de la panne est un problème réel dans notre contexte, quand elle est mal faite ou tout simplement pas faite, elle peut entraîner d'énormes pertes sur divers plans : économique, matériel, temporel,...
Nos problèmes sont donc :

- Comment détecter, de façon précise l'origine d'une panne afin de pouvoir en informer le conducteur ou le mécanicien sur son apparition?
- Comment suivre de façon précise et fiable l'évolution des composants vitaux d'un véhicule afin de prévenir le conducteur de l'arrivée probable d'une panne ?

Objectif

Objectif principal
Il consiste à fournir un kit d'auto-diagnostic permettant d'anticiper et de prendre en charge de façons réactive en temps réel la détection des pannes dans le contexte camerounais.
Cela suppose plusieurs objectifs spécifiques.
Objectifs spécifiques (figure 2)

- Comprendre le fonctionnement des systèmes embarqués;
- Programmer des applications à forte contraintes temporelles;
- Déployer un ensemble de capteurs permettant de monitorer plusieurs paramètres vitaux d'un véhicule (moteur, carburant, freinage,...);
- ...

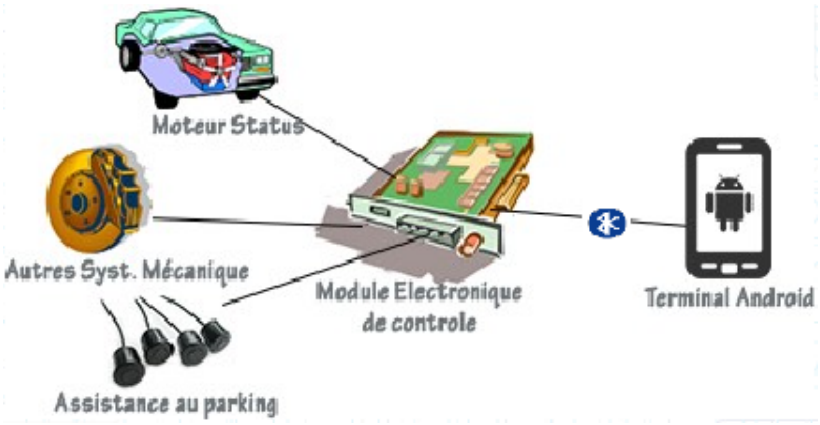


Fig. 2 Architecture générale du système

Méthodologie globale du travail

Nous rappelons que fournir un kit d'auto-diagnostic permettant d'anticiper et de prendre en charge la détection des pannes est notre préoccupation.
Pour ce faire, nous proposons :

- Une analyse du problème, laquelle sera effectuée par des interviews avec les experts du domaine et une revue de la littérature;
- Une approche conceptuelle basée sur les modèles, pour la conception du système embarqué, nous utiliserons en occurrence AADL;

- Une approche d'ingénierie dirigée par les modèles, par la génération automatique du squelette du code de l'application à partir des outils tel qu' OSATE et Ocarina;
- Une mise en œuvre, une simulation basée modèle par la simulation des résultats avant implémentation (CHEDDAR);
- Une mise en œuvre exécutive, en implémentant la solution avec un exécutif temps réel ERIKA Enterprise;
- Une implémentation effective des résultats sous les plate-forme Arduino;
- Une présentation des résultats et des statistiques sous Android;

Plan du mémoire

Le présent manuscrit est constitué de deux grandes parties :

1. La première partie présente l'état de l'art de l'existant et insiste sur les travaux qui ont déjà été menés pour cette thématique .
   Cette partie comporte deux chapitres :
   - Le premier présentera le système OBD[1] qui est la norme internationale en matière de réduction du taux de pollution automobile qui sera suivi d'un tour d'horizon concernant l'auto-diagnostic vu par certains grands constructeurs automobiles. Bien entendu, le panorama présenté n'a pas prétention à l'exhaustivité car les seules approches retenues ici sont celles jugées les plus intéressantes du point de vue de notre problématique;
   - Le deuxième chapitre présentera les grands axes logiciels qui ont permis de mener à bien ce projet.
2. La deuxième partie (contribution) qui constitue le "cœur" de ce travail, puisqu'elle s'intéresse à nos apports personnels dont l'objectif est de permettre la mise en œuvre de notre système d'auto-diagnostic. Elle est subdivisée en trois chapitres:
   - Présentation des pannes automobiles assez fréquentes dans notre contexte;
   - Présentation d'une approche d'ingénierie basée sur les modèles;
   - Présentation des résultats expérimentaux et une exploitation des données obtenues.

[1] OBD: On-Board Diagnostic.

3. Une conclusion générale récapitule le contexte de notre travail ainsi que notre contribution.

Partie I

État de l’art

Chapitre 1

Présentation des outils d'auto-diagnostics libres et propriétaires

Sommaire

Introduction

Dans cette partie, nous allons décrire les différents systèmes d'auto-diagnostics, et leurs concepts de fonctionnement, puis résumer leurs avantages et leurs inconvénients en vue d'une application dans le nôtre.

1.1 Présentation du standard OBD

Le système de diagnostic embarqué, ou On-Board Diagnostics (en abrégé OBD), est l'ensemble de dispositifs de diagnostic qui est embarqué dans la plupart des véhicules

à moteur thermique produits dans les années 2000.
L'OBD décrit les moyens à mettre en œuvre pour contrôler l'ensemble des composants du groupe motopropulseur affectant les émissions polluantes du véhicule tout au long de sa vie.[8]

1.1.1 Historique

À partir des année 1980, les constructeurs automobile ont commencé à intégrer massivement les systèmes électroniques dans leurs véhicules, en particulier à travers l'utilisation d'un calculateur de contrôle moteur (aussi appelé calculateur d'injection, initialement utilisé en essence) destiné à gérer le fonctionnement du moteur et à diagnostiquer ses défaillances.
Les diagnostics embarqués sont devenus progressivement de plus en plus sophistiqués, pour permettre aux moteurs de respecter les seuils d'émissions polluantes réglementées de plus c'est l'Agence de Protection de l'Environnement qui a fixé les premiers seuils standards.
Afin d'assurer le contrôle précis exigé par cette technologie moderne, les automobiles légers construits depuis le début des années 1980 disposent pratiquement tous d'un ordinateur de bord. Cet ordinateur contrôle le rendement du système d'injection de carburant et d'autres paramètres du moteur en régime dynamique. Ces systèmes informatiques « intelligents » sont aussi appelés "systèmes de diagnostic de bord", ou OBD.
Contrairement à la première génération OBD I, qui surveillait peu de paramètres d'antipollution, voire aucun; de surcroît elle était conçue sur mesure pour chaque modèle de véhicule, c'est-à-dire conformément aux exigences de chaque constructeur de véhicules en matière de conception, matériel et logiciel. Un technicien en réparation ne pouvait donc pas interroger le système OBD sans avoir accès au matériel de connexion et aux codes informatiques de la marque et du modèle du véhicule. Le système de deuxième génération OBD II, quant à lui, surveille les systèmes d'antipollution et les principaux composants du moteur. Ces systèmes OBD modernes, appelés OBD II, sont maintenant normalisés (1996). La normalisation a été fondamentale. Règle générale, peu importe le type de véhicule, les systèmes OBD II surveillent maintenant les mêmes composants, utilisent le même langage informatique et appliquent les mêmes critères pour évaluer les systèmes et signaler les problèmes au conducteur et au technicien en réparations.[4]

1.1.2 Description du système OBD II

Le système OBD II surveille le système de contrôle des émissions polluantes qui diagnostique la dégradation de la performance des dispositifs antipollution, d'alimentation, d'allumage et

leurs composants. Quand un problème (les émissions dépassent la norme, généralement 1,5 fois la norme élaborée par l'EPA[1]) susceptible d'entraîner une hausse notable d'émissions atmosphériques est décelé, le système OBD déclenche l'allumage du témoin d'anomalie dans le tableau de bord: le voyant MIL[2], pour indiquer au conducteur qu'il doit faire vérifier son véhicule par un technicien.
Le voyant MIL est un pictogramme représentant un bloc moteur de couleurs orange/jaune (le rouge étant interdit) spécifiées par l'ISO 2575.La figure ci dessous présente quelques exemples sur différents modèles de véhicules.

Fig. 1.1 Voyant MIL

Ce voyant sert à signaler un problème sur le système de dépollution du véhicule. Selon les défauts il y a plusieurs modes de fonctionnement possibles[15] :

- **Allumé en permanence :** un défaut affectant les émissions de polluants a été détecté et confirmé par le calculateur. Le véhicule peut continuer à rouler;
- **Clignotant :** un défaut pouvant provoquer la destruction de certains organes du véhicule a été détecté. Dans ce cas il est vivement recommandé d'arrêter le véhicule très rapidement. Ce mode s'accompagne la plupart du temps d'un passage du véhicule en mode dégradé (limitation du régime et de la puissance);
- **Fugitif :** un défaut a été détecté mais le système n'a pas confirmé la présence du défaut. Le voyant s'éteint donc lui même;
- **Éteint :** Dans ce dernier cas aucun défaut lié à la pollution n'est actif. Cela ne signifie pas pour autant qu'aucun défaut n'est présent. Certains d'entre eux ont très peu d'influence sur le fonctionnement du véhicule comme par exemple un dysfonctionnement des bougies de préchauffage.

[1]EPA: Environnement Protection Agency
[2]MIL : Malfunction Indicator Lamp.

La communication entre le calculateur OBDII et le technicien de maintenance, se fait à partir d'un connecteur de diagnostic. La norme stipule que le connecteur doit obligatoirement se situer dans l'habitacle. Généralement celui-ci se trouve sous le volant dans le compartiment à fusible ou sous le cendrier près du frein à main(cf. figure 1.2).

Fig. 1.2 Vue de face du connecteur OBD II [11]

1.1.3 Les avantages et les inconvénients du système OBD

Les systèmes OBD-II offrent plusieurs avantages mais aussi des inconvénients[4].

Avantages

- Plus de précision dans le diagnostic et la réparation des problèmes du système antipollution; Temps d'inspection réduit;
- Temps d'inspection réduit;
- Potentiel unique de réduction de l'évaporation de carburant grâce à la détection de certains défauts du système de contrôle de l'évaporation de carburant.

Inconvénients

- Un des inconvénients des défauts génériques c'est que pour être commun à tous les véhicules ils perdent parfois de leur clarté;
- Manque de fiabilité de la connectique générant des pannes aléatoires d'où la difficulté pour les mécaniciens d'appréhender les parties électroniques et informatiques;
- Obligation de disposer d'une valise de test pour diagnostiquer et effacer les défauts.

Cependant, même s'ils constituent une nouvelle arme puissante, les systèmes OBD II ne contribuent à réduire les émissions excessives des véhicules que dans la mesure où les

automobilistes et les propriétaires de véhicules réagissent de manière responsable quand s'allume le témoin d'anomalie du système OBD-II.
Malheureusement, l'expérience révèle que les automobilistes et les propriétaires de véhicules ignorent souvent les témoins lumineux.

1.2 Présentation des outils d'auto-diagnostics selon les grands constructeurs

Plusieurs géants de l'automobile (Mercedes, Hyndai,..) implémentent diverses solutions d'auto-diagnostic, lesquelles assistent le conducteur dans sa conduite.
Les grands constructeurs automobiles évoluent dans un contexte hautement concurrencé. Ce qui justifie le fait que ces solutions soient fermées et propriétaires; cette concurrence farouche a encore été présentée par le groupe automobile allemand Volkswagen qui sur l'immense pression de l'industrie automobile, s'est vu contraint de truquer leur module d'analyse des émissions en oxyde d'azote(NO_x) et carbone (CO_2) sur ses moteurs diesel et essence, lors des tests d'homologation[7].
En partant du fait que ces systèmes d'auto-diagnostic sont propriétaires, l'analyse conceptuelle et technique de ces derniers s'est avérée compliquée voir même impossible, dans le cadre de ce projet. De ce fait nous nous sommes appesantis à étudier ce que nous pouvons voir c'est à dire le tableau de bord. Ce dernier à partir de ces pictogrammes normalisés à l'international, renseignent sur l'état de certains paramètres vitaux (niveau des fluides de l'automobile, état de plaquette de frein,...) du véhicule.
La figure 1.3 ci-dessous présente un exemple de tableau de bord truffé de pictogrammes, qui présentent l'état du diagnostic de l'automobile. Ce diagnostic est effectué de manière automatique et transparente à l'utilisateur.

Fig. 1.3 Tableau de bord d'une voiture

1.3 Conclusion

Il en ressort de ce chapitre, qu'il éxiste deux grands groupes de système d'auto-diagnostic :

- Le standard international ***OBD*** oreinté vers l'*anti-polution*, actuellement dans sa version 2. Il est "*ouvert*" et utilisé par les grands constructeurs automobile ;
- Les systèmes d'auto-diagnostics fournis par les constructeurs automobiles, ces derniers sont par contre *fermés* et *propriétaires*.

Chapitre 2

Outils technologiques utilisés

.

Sommaire

.

Introduction

Ce chapitre présentera un condensé des grands axes technologiques utilisés pour la conception et l'implémentation de ce projet. Nous pouvons citer entre autre sur le point de vue logiciel: le langage de modélisation AADL, le simulateur d'ordonnancement Cheddar et l'outil d'implémentation ERIKA Enterprise. Quant à l'axe matériel on présentera la carte à

microprocesseur Arduino et l'OS Android, en passant par une brève présentation des modes de communication entre ces deux système hétérogènes. Il est important de préciser que chacun de ces choix sera justifié dans la partie « Contribution ».

2.1 Outils logiciels

Cette section présentera les outils tels que : AADL, Cheddar et ERIKA Enterprise.

2.1.1 Modèle de conception AADL

Présentation générale du langage AADL

AADL[1] a atteint sa maturité en 2006, c'est un langage de description d'architectures destiné aux systèmes embarqués ayant des contraintes de ressources strictes (taille, poids, et puissance), et qui sont soumis à des besoins de réponse temps réel. AADL est un langage textuel et graphique destiné à la modélisation des architectures logicielles et matérielles des systèmes embarqués temps réel et leurs caractéristiques de performances critiques. La description d'une architecture en AADL consiste en la description de ses composants et leurs compositions sous forme d'une arborescence. De multiples propriétés permettent de considérer les caractéristiques des composants telles que : les propriétés temporelles, flux de données, modes de fonctionnement, etc. [2].
Pour le langage AADL, les outils de modélisation textuelle et graphique, de simulation et de vérification sémantique sont fondés sur l'environnement OSATE (Open Source AADL Tool Environnement) [2] basé sur la plate-forme Eclipse. Quant au logiciel de simulation du comportement, d'ordonnancement et de validation d'architecture AADL, nous utiliserons : CHEDDAR.
Le langage AADL est essentiellement basé sur les composants et sur leurs interactions, ces derniers sont de type logiciel ou matériel.

Les composants AADL [18]

Un composant est la représentation d'une entité logicielle ou matérielle, réutilisable (composé d'un type/interface et d'une ou de plusieurs implantations). Ces propriétés, nous indiquent toutes informations nécessaires à sa mise en œuvre ou à son analyse. Il n'est pas exclu qu'il puisse avoir de sous-composant.
Les composants logiciels Ses principaux composants logiciels sont les suivants:

[1] AADL: Architecture Analysis & Design Language

- ***thread***: est un flot de contrôle qui exécute un programme;
- ***data***: c'est une structure de données implantée dans le langage cible (struct C, class C++/Java, ...);
- ***process***: il modélise un espace mémoire. Un *process* doit contenir au moins un *thread*;
- ***subprogram***: ce composant modélise un programme exécutable séquentiellement (fonction en C, méthode en Java,...).

Les composants matériels Ces derniers sont les suivants:

- ***processor/virtual processor***: c'est une abstraction du logiciel/matériel en charge de l'ordonnancement des threads. Un *processor* peut comporter plusieurs *virtual processors*;
- ***memory***: modélise toute entité de stockage physique de données (disque dur, mémoire vive, etc);
- ***device***: composant qui interagit avec l'environnement. On ignore sa structure interne (thread, data, ...). Ex : capteur, actionneur;
- ***bus***: entité permettant l'échange de donnée/contrôle entre *device*,*memory* ou *processor* (ex : réseau de communication, bus, etc).

On a aussi un autre type de composant: le composant ***system*** dont la particularité est qu'il est au dessus des deux types de composant présentés ci dessous. Il permet de structurer un modèle d'architecture tout en manipulant les composants: (*process*, *processor*, *device*, *bus*). Tout cet ensemble de composants de fonctionnent pas en vase clou, mais ils interagissement entre eux.

Connexions entre composant

La connexions entre composants, modélisent les interactions entre ces derniers (flot de contrôle et/ou échange de données).
Le ***features*** est l'interface du composant. Chaque *feature* est caractérisée par un nom, une catégorie, une orientation, un type de donnée, ...
les catégories de *feature* ou types d'interaction sont les suivantes :

- ***event port***: émission/réception d'un signal;

- ***data port/event data port***: échange synchrone/asynchrone d'un message;
- ***subprogram parameter***: paramètre lors d'appels de sous-programme;
- ***data access***: accès à une donnée partagée (composant data).

L'orientation des ports des features sont de trois types :

- ***in***: entrée;
- ***out***: sortie;
- ***in out***: entrée-sortie.

L'un des résultats d'une modélisation AADL, est un graphe où sont représentés les différents composants intervenant dans la modélisation et les connexions entre ces composants. Le respect des contraintes temporelles de ce modèle pourra se faire à partir d'un simulateur d'ordonnancement: Cheddar

2.1.2 Cheddar

Présentation générale du simulateur d'ordonnancement Cheddar

Cheddar est un outil d'ordonnancement temps réel gratuit. Il est conçus pour vérifier le respect des contraintes temps réel des tâches d'une application ou d'un système. Le système à analyser peut être décrit avec l'architecture AADL ou avec le langage de description d'architecte de Cheddar[6].
Cheddar met à disposition deux modes d'exécution principaux : un moteur de simulation et outils de tests de faisabilité.

Les outils de tests de faisabilité

Ces derniers permettent à l'utilisateur d'étudier le caractère temps réel d'un modèle. Ces outils utilisent des algorithmes pour de vérifier si toutes les contraintes temporelles sont respectées pour que c'est ensemble de tâches soient ordonnable par un ordonnanceur temps réel. Liu et Layland ont définis un simple modèle de tâche, appelé tâches périodiques, qui sont caractérisés par trois paramètres : une date de fin (Di), une période (Pi) et une capacité (Ci). Le modèle d'une tâche périodique est un système dont le caractéristique sont les suivantes :

- Chaque tâche est activée à une période Ti, pour tout Ti : Di = Pi ;

- Une tâche durant son activation doit finir sa durée total d'activation Ci avant sa date de fin Di.

Ces paramètres permettent de faire les tests de faisabilité. Ces tests évaluent les différents critères de performances : taux d'utilisation de processeur, pire temps réponse,l'inversion de priorité dû à l'accès aux données,... Liu et Layland ont proposés une simple équation de test de faisabilité (figure 2.1), permettant de calculer taux d'utilisation du processeur avec un ordonnancement EDF (Earliest Deadline First) [26].

$$\sum_{1}^{n} \frac{Ci}{Pi} \leq 1$$

Fig. 2.1

La figure 2.2 montre un exemple de validation des tâches présentées ci-dessus par AADL, par cette méthode:

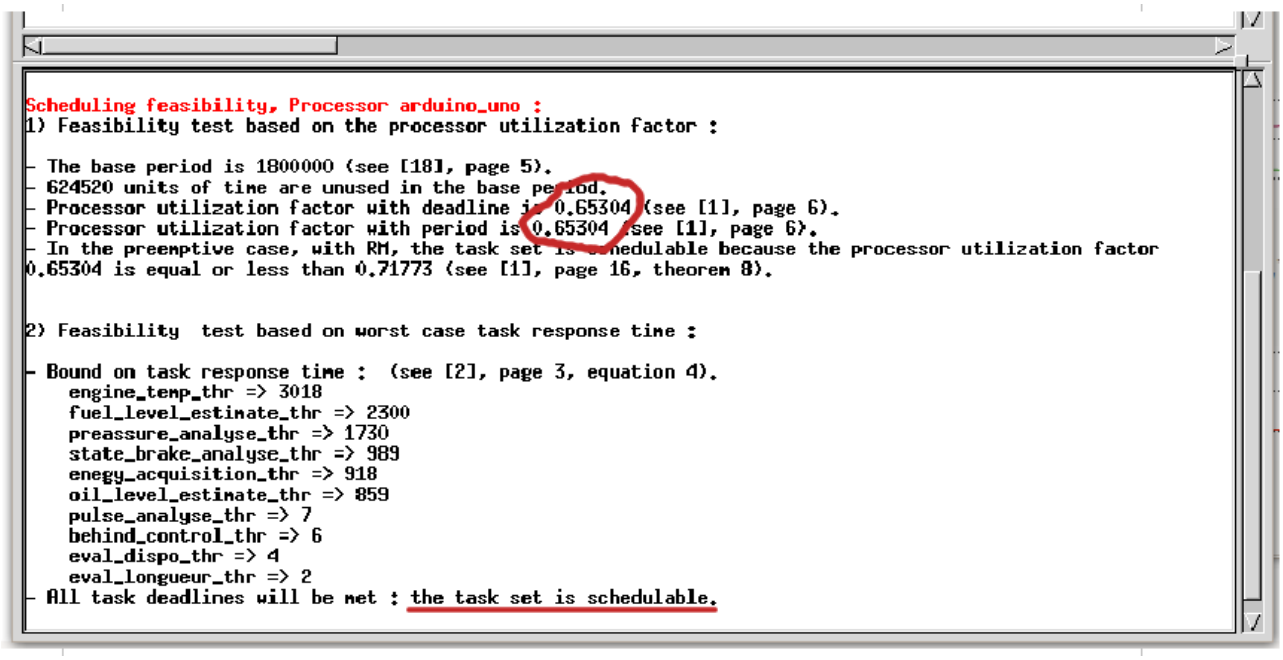


Fig. 2.2 Une vue de Cheddar: validation par test de faisabilité

A partir, de cette figure on peut lire que pour l'ordonnancement de ces tâches un processeur arduino Uno aura un taux d'utilisation d'environ 0,65 avec l'algorithme d'ordonnancement « Rate Monotonic », ce taux étant intérieur à 1, donc cet ensemble de tâches est ordonnable.

Les moteur de simulation

Ils utilisent un ordonnanceur et les propriétés des tâches.
Cette approche applique un algorithme d'ordonnancement sur une Hyper-période (Fenêtre temporelle minimale qui correspond à un intervalle sur lequel la validation du système garantit l'absence définitive de faute temporelle, c'est à dire une violation de contrainte temporelle) afin de trouver un ordonnancement [5].
La figure 2.3 montre un exemple de validation des tâches présentées ci-dessus par AADL, par cette méthode:

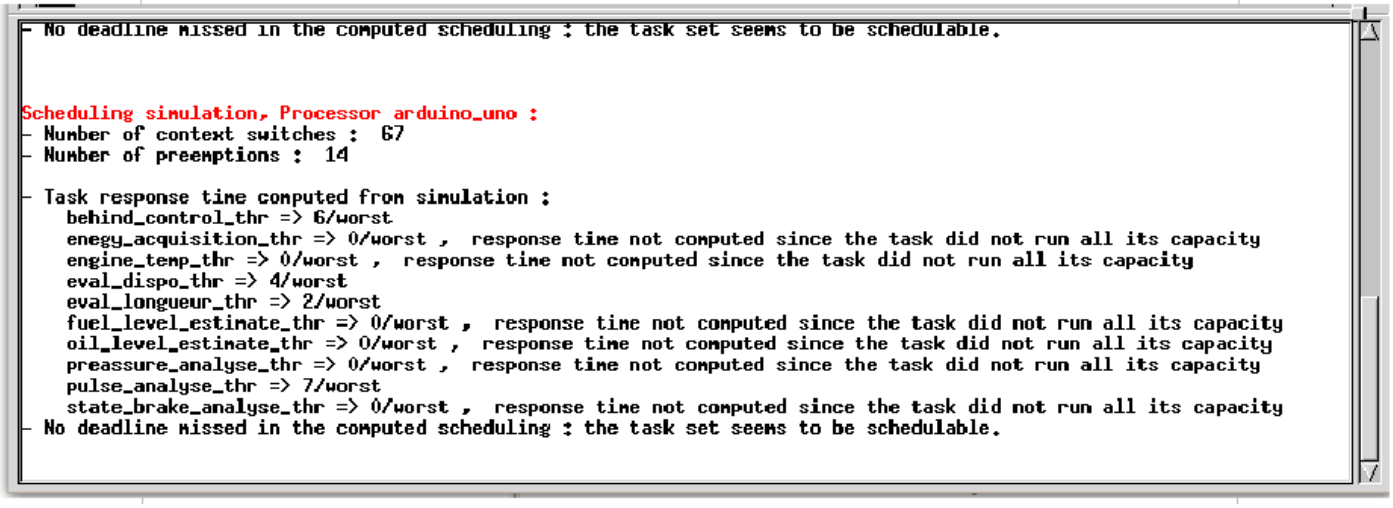


Fig. 2.3 Une vue de Cheddar: Simulation d'ordonnancement

Les principales fonctions fournies par Cheddar sont les suivantes[TCP 15] :

- Les fonctions du simulateur d'ordonnancement sont celles classique en temps réel :
  - Analyse Rate Monotonic (aussi appelé RMA, RM ou RMS) ;
  - Earliest Deadline First (EDF) ;
  - Deadline Monotonic (DM) ;
  - Least Laxity First (LLF) ;
  - ...
- Support des tâches périodiques ou apériodique ;
- Ressources partagées ;

- Simulateur d'ordonnancement d'applications distribuées sur plusieurs processeur avec messages partagés.

- ...

Après la validation d' ordonnançabilité des tâches de notre modèle AADL, par le simulateur Chaddar. Il convient de présenter un outil d'implémentation qui prend en compte la propriété multitâche du modèle AADL ainsi simulé.

2.1.3 ERIKA Enterprise

Présentation d'ERIKA Enterprise

L'environnement par défaut d'Arduino est conçu pour exécuter un seul processus. Or dans l'optique de fiabiliser notre système, en le rendant à l'écoute de tous les capteurs de notre réseau. Nous devons lui mettre à disposition plusieurs processus d'exécution simultanés. Pour ce faire la programmation multi-tâche sur Arduino se fera dans ce projet à partir de exécutif temps réel ERIKA Enterprise.
ERIKA Enterprise est sous la licence GPL, c'est un système d'exploitation temps réel pour les microcontrôleur basé sur l'API proposé par le consortium OSEK/VDX[1]. Dont le noyau donne le support pour le multitâche préemptif et non préemptif, et implémente quelques algorithmes d'ordonnancement comme la priorité fixe avec préemption, Stack Resource Policy (SRP), et Earliest Deadline First (EDF). Il implémente une pile partagée afin d'optimiser la RAM. L'API supporte les tâches, événements, alarmes, ressources, sémaphores et la gestion d'erreurs. Il est important de préciser qu'il a une faible occupation mémoire. Par exemple, une installation minimale du noyau consomme entre 800 et 2000 bit de code, pour une implémentation à ordonnancement à priorité fixe avec pile partagée et deux mutex. L'environnement de développement d'Erika Enterprise est basée sur la compilation croisée. Evidence srl fournit RT-Druid (basé sur Eclipse) est environnement de développement par défaut. Il implémente un compilateur du langage OIL, qui à parti des spécifications OIL pour générer les configuration du noyau [23].
L'installation de l'environnement complet de développement et de débogage des cartes embarquées. Ce fait à partir d'une machine virtuelle sous Linux (Figure 2.4), où les principaux logiciels sont préinstallés et près à l'emploi. Avec plusieurs exemples compilés sous OSEK/VDX pour plusieurs cartes.

[1] OSEK/VDX:Open systems and the corresponding interfaces for automotive electronics/Vehicle Distributed eXecutive

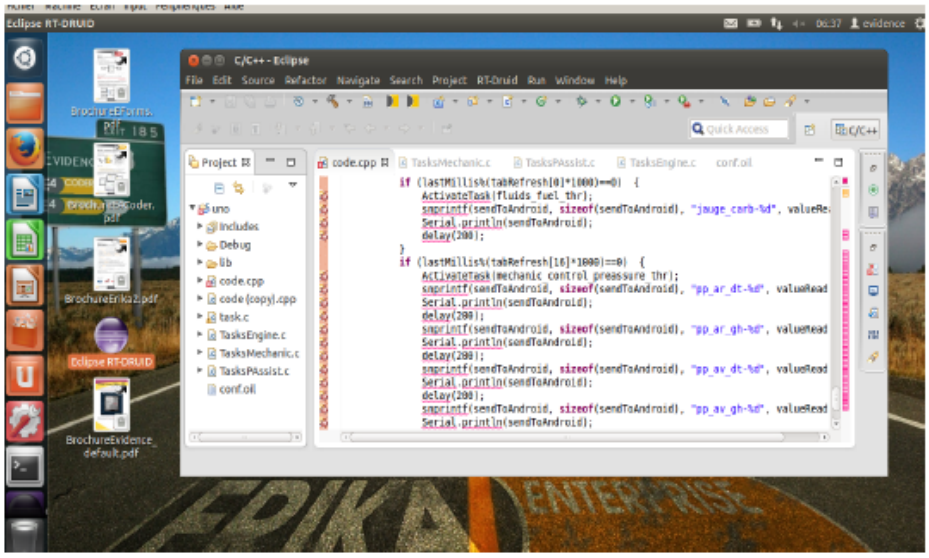


Fig. 2.4 Une vue de la machine virtuelle d'ERIKA Enterprise

En d'autre thermes, cette machine virtuelle offre un environnement complet dont les grandes lignes sont les suivantes :

- Édition des application OSEK/VDX en utilisant le noyau open-source OSEK/VDX d'ERIKA Enterprise;
- Compiler son application à partir des open-sources déjà pré-compilés ;
- ...

ERIKA Enterprise supporte plusieurs cartes embarquées, dont les principales sont les suivantes [24]:

- STM32F4Discovery;
- Arduino Uno;
- Cubieboard2;
- ...

Tout en respectant une méthodologie d'ingénierie dirigée par les modèles, lesquels modèles ont été conçu et simuler par les outils logiciels précédemment présentés. La section suivante présentera les outils et supports matériels sur lesquels seront implémentés et hébergés ces modèles.

2.2 Outils matériels

Cette section présentera : la carte à microprocesseur Arduino, le système d'exploitation mobile Android et les différents protocoles de communication entre ces deux entités.

2.2.1 Le microprocesseur Arduino

Connecté ou embarqué le contrôle des systèmes requiert souvent une plate-forme de traitement compatible avec la technologie temps réel. De ce fait on a besoin d'un microprocesseur pouvant prendre en charge la contrainte temps réel. Le microcontrôleur Arduino Uno qui a 14 ports digitaux dont 6 sur les 14 ports peuvent être utilisés comme PWM[1] et 6 ports analogiques; le tout manager pour un microcontrôleur ATmega328 cadencé à 16Mhz. Il a l'avantage d'être compatible avec le noyau temps réel ERIKA d'Evidence srl. [22] en plus d'être petit et léger ce qui minimise le poids et l'espace utilisé dans la voiture.

2.2.2 Le système d'exploitation mobile Android

Android est une pile de logiciel pour téléphones mobiles ; il inclut un système d'exploitation, et un ensemble d'applications clés. Le SDK Android fournit les outils et APIs nécessaire pour commencer le développement d'applications compatibles avec la plate-forme Android. Les applications Android sont écrit en langage de programmation Java. Le SDK Android compile le code avec des données et fichiers ressources d'Android, en un fichier archive avec l'extension *.apk*. Prêt à être installé dans un terminal Android [25].

2.2.3 Communication

Le microcontrôleur (Arduino) et le support d'IHM (Android) étant deux entités distinctes. Pour leur mise en communication nous avons deux grandes topologies qui s'offrent à nous: la topologie filaire et celle sans fil. Ainsi dit cette section sera consacrée à l'étude des éléments de chacune de ces topologies:

Filaire

La communication est assurée par un câble USB Standard B (côté Arduino) couplé avec un câble OTG (côté Android).

[1] PWM:largeur d'impulsion modulée

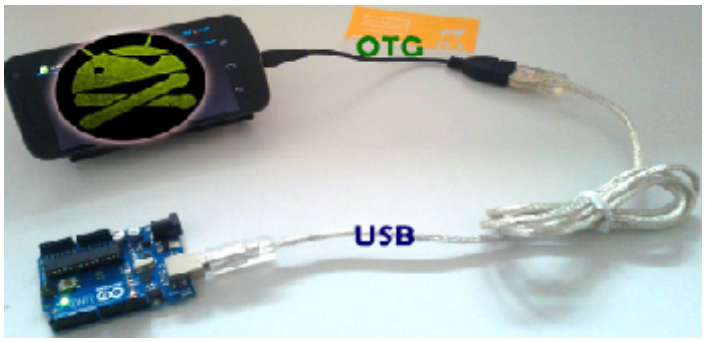


Fig. 2.5 Communication par USB Arduino – Android

Bien qu'ayant un débit théorique élevé allant jusqu'à 480Mbit/s, en plus elle n'est pas sensible aux radiations éventuellement présentent dans l'air comme ses homologues sans fil. Elle présente tout fois quelques inconvénients :

- Certains Smartphones nécessiterons les droits super utilisateurs (root), pour utiliser ce type de communication. L'attribution du droit super utilisateur pourrait endommager le téléphone de façon irréversible;
- Le câblage sera encombrant et limitera significativement la portée du système.

Ces inconvénients ont été suffisants pour nous permettre d'abandonner cette approche et de nous tourner vers une communication sans fil.

Communication sans fils

Cette topologie rassemble plusieurs technologies de communication en son sein dont les plus usuelles sont:GPRS, Wi-Fi et Bluetooth:

- ***Wi-Fi*** : ce mode de communication a une portée de près de 100m et un débit élevé plus de 11Mbits/s;
- ***Bluetooth*** : c'est un standard pour échanger les données entre appareils à une faible distance soit 10m;
- ***GPRS*** : Ce dernier a une portée sur toute l'étendue du réseau mobile de l'opérateur, qui s'étend en général à une échelle nationale et même internationale.

2.3 Conclusion

Ce chapitre était consacré à la présentation, des outils que nous utiliserons pour mener à bien ce travail. Ces derniers peuvent être divisés en deux catégories:

- Outils logiciels
  - Le langage ***AADL*** qui nous permettra de modéliser notre système;
  - Outil de simulation ***Cheddar***, validera le modèle obtenu par le précédent outil;
  - L'executif temps-réel ***ERIKA Enterprise***, pour l'implémentation algorithmique du modèle obtenu.
- Outils matériels
  - Le ***microcontrôleur Arduino***, c'est la carte physique sur laquelle sera hébergé le code généré par *ERIKA Enterprise*;
  - Le système d'exploitation mobile ***Android***, le "reporting" des traitements obtenus par le microcontrôleur ce fera sur un terminal "Android".

Partie II

Contribution

Chapitre 3

Analyse

Sommaire

Introduction

Ce chapitre présentera le condensé de nos travaux d'analyse, notamment: l'étude des pannes les plus fréquentes sur nos automobiles, les capteurs utilisables pour étudier les variations des

paramètres renseignant sur le déclenchement probable de ces pannes et pour finir nous présenterons et justifions le modèle de base de données choisi pour ce travail; où les informations sont censées varier très rapidement.

3.1 Pannes fréquentes

Après avoir mené de longs interviews auprès des garagistes et automobilistes expérimentés, nous avons pu classer les pannes en quatre grands groupes: démarrage impossible, mauvaise alimentation du moteur, moteur « coule », suspension et freinage.
Dans cette section, chacune de ces pannes sera passée au scanner: présentation de leurs causes possibles et éventuellement quelques conseils pour mieux les prévenir.

3.1.1 Démarrage impossible

C'est le cas le plus fréquent qui est déjà arrivé au moins une fois à un automobiliste ayant accumulé plusieurs années d'expérience au compteur. Soit le scenario suivant: vous vous levez de bonne humeur, parés à faire face à vos challenges de la journée, une fois dans votre véhicule vous essayez de le démarrer mais sans succès. Là vous vous posez une question: "c'est quoi le souci cette fois ci". Le causes majeures de ce désagrément peuvent être:
- Batterie trop faible pour démarrer le véhicule;
- Un peu surprenant mais on peut aussi noter une panne sèche;

"J'avais un problème de démarrage, je me suis rendu chez un garagiste qui après avoir passé plusieurs heures de recherche, s'est rendu compte que le souci venait de la batterie qui était trop faible pour un démarrage. Depuis cet intervention de maintenance, l'alarme de mon véhicule est hors d'usage". Témoigne un automobiliste attristé par ces pertes en temps et financières (frais de réparation de l'alarme endommagée) dues à cette maintenance qui s'avérait être une intervention de routine.
Ce qui vient une fois de plus tirer la sonnette d'alarme sur la nécessité d'un dispositif de diagnostic de bord et simple d'utilisation.

3.1.2 Mauvaise alimentation du moteur

Une mauvaise alimentation du bloc moteur en carburant conduit directement à une baisse de régime de ce dernier. Celle-ci pourrait être due:

- A une pompe défectueuse ne pompant pas assez de carburant, nécessaire au bon fonctionnement du moteur;

- Nous pouvons aussi pointer le doigt sur la «qualité du carburant».

3.1.3 Le moteur «coule»

Un moteur qui «coule» est complètement inutilisable, dont il faut inévitablement le remplacer. Cette panne est considérée comme une des plus critiques et ses causes peuvent être les suivantes:

- Température trop élevée du moteur soit entre 80 et 90°C;
- État (trop usé) de l'huile du moteur et son niveau bas.

Conseil: Contrôler régulièrement son niveau;

- Température élevée du liquide de refroidissement et son niveau bas;
- Tartre au niveau du radiateur qui empêche l'arrivée du liquide de refroidissement au moteur.

Conseil: Associer un liquide détartrant à l'eau de refroidissement.

- Usure des "coussinets" (composant séparant les parties mobiles des parties fixes du moteur). Il est recommandé d'être attentif au bruit du ronflement du moteur: ce bruit varie aussi avec l'état d'usure des "coussinets".

3.1.4 Suspension et freinage

La suspension

Cet ensemble peut être soumis à deux types de problèmes dont l'un est très rare: un ressort cassé et l'autre plus fréquent: amortisseur usé. Cette usure est dans la plupart des cas due à l'état des routes (trous sur la route, dos d'âne mal signalés,...).

Freinage

Les problèmes les plus fréquents sont l'usure des plaquettes de frein et la basse pression d'air dans les pneus, pouvant causer de nombreux désagrément sur la route dont la perte du contrôle de votre véhicule.

3.1.5 Autres

L'un des désagréments aussi observé mais qui pourrait paraître des moindres sont les rayures et pots de phares cassés, généralement dus à la manœuvre "créneaux", remarqués sur plusieurs véhicules.
Arrivé au terme de cette analyse, dont il était question d'étudier les pannes les plus fréquentes sur les véhicules terrestres légers dans un environnement tropical. Nous avons pu en recenser plusieurs, dont la plupart arrive de manière soudaine sans crier garde.
La section suivante sera consacrée à l'étude des différents capteurs pouvant être déployés au sein d'un véhicule, afin de permettre à l'automobiliste d'avoir un œil en permanence sur les paramètres sensibles de son automobile.

3.2 Capteurs adéquats

Avant de commencer il est important de préciser, qu'un capteur est un équipement permettant observer un phénomène physique et de transformer ses variations non-électriques en celles électriques. Comme le montre la figure 3.1 l'utilisation de capteurs en automobile a connu un essor à partir de 2001 et depuis lors elle n'a pas cessé d'évoluer.

Fig. 3.1 Evolution du marché des capteurs[3]

Revenons à nos bonnes vieilles pannes, les capteurs permettant de garder un œil sur les paramètres "clés" et influençant grandement sur ces dysfonctionnements seront présentés ci-dessous:

3.2.1 Démarrage impossible

Batterie trop faible

Le capteur électronique de batterie est un composant essentiel du système électronique de gestion énergétique. Implanté dans la cavité des bornes de la batterie, il enregistre dynamiquement et avec la plus grande précision les valeurs relatives à la batterie, comme le courant, la tension et la température et fait aussi des prévisions de son comportement électrique.

Fig. 3.2 Capteur Bosh des paramètres vitaux d'une batterie

Niveau de carburant

Le suivi de l'état du carburant dans le réservoir, se fait généralement à partir d'un dispositif à flotteur présenté à la Figure 3.3. Son principe de fonctionnement est assez simple; le capteur utilisé ici est une résistance variable entraînée par un flotteur, la variation de la valeur de la résistance en (est) fonction de (du) niveau d'essence présent dans le réservoir[10]. C'est le même principe que celui du variateur de lampe halogène de votre salon.

Mauvaise alimentation du moteur

- Une pompe défectueuse;
- Un débiteur n'éjectant pas le carburant nécessaire pour le moteur.

3.2.2 Le moteur "coule"

Température du moteur

Les capteurs de température mesurent électroniquement la température à partir des modifications de résistances au moyen des résistances NTC ou des résistances PTC. La plupart du

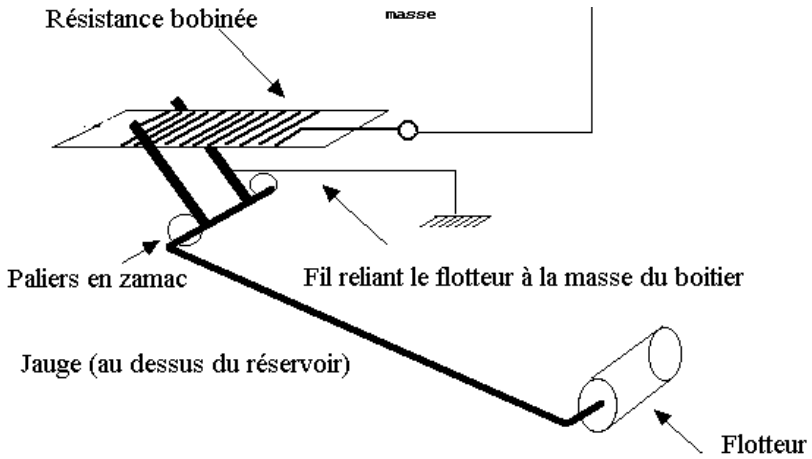


Fig. 3.3 Flotteur de niveau de carburant dans un réservoir

temps des (les) résistances NTC sont utilisées. Avec les NTC[1] en cas d'une augmentation de température la valeur de la résistance diminue. Par contre PTC[2] à l'augmentation de température la valeur de la résistance augmente. Les valeurs de résistances correspondantes aux valeurs de températures sont transmises à l'appareil de commande sous forme d'un signal de tension.[1]

Fig. 3.4 Capteur de température

Niveau d'huile moteur

Ce capteur de niveau permet de définir le niveau de remplissage de l'huile de moteur (pas de fluides conducteurs). Son principe de fonctionnement est décrit comme suit:

Le fil thermique du capteur de niveau est échauffé brièvement par un courant constant (1-2 sec env.). La valeur de résistance du fil augmente, ainsi que la chute de tension U1.

[1]NTC: Coefficient de Température Négatif.
[2]NTP: Coefficient de Température Positif.

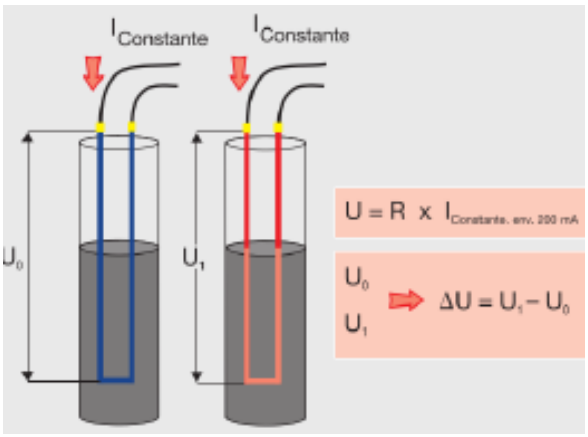


Fig. 3.5 Principe de mesure du niveau d'huile moteur

(L'augmentation de la résistance par auto-échauffement du fil est déterminée par le rapport de la profondeur d'immersion du fil dans l'huile et représente ainsi une grandeur proportionnelle au niveau.)
Si l'on mesure, lors de la mise sous tension, la chute de tension U0 (à froid), il découle du signal différentiel U1 – U0, une fonction proportionnelle au niveau d'huile.[17]

Température et niveau du liquide de refroidissement

Les spécifications de ce capteur seront semblables, à celles utilisées pour la mesure de la température du moteur.

3.2.3 Freinage

Un capteur est incorporé dans la garniture et lorsque les plaquettes sont usées, le contact ferme un circuit électrique, ce qui génère un signal électrique.

Fig. 3.6 Plaquette de frein avec capteur d'usure incorporé

3.2.4 Autres

Dans l'optique de diminuer la dégradation de la carrosserie lors des créneaux, nous allons mettre en place un système d'aide au stationnement.

Aide au stationnement

Le radar de recul est un système utilisé dans l'industrie automobile pour faciliter le stationnement même lorsque la visibilité arrière est nulle. Ce type de radar fonctionne sur le même principe qu'un radar classique, sans toutefois utiliser le même type d'ondes. On devrait donc l'appeler sonar et non radar [14].
Des émetteurs propulsent des ondes ultrasons dans l'air (ultrasons car il faut éviter qu'on les entende ! L'oreille humaine n'arrivant pas à déceler des sons aux fréquences trop élevées). Celles-ci sont réfléchies (rebondissent) lorsqu'elles rencontrent un obstacle et reviennent en partie vers le dispositif émetteur. Les ondes réfléchies par l'obstacle sont ensuite reprises par les capteurs, puis la centrale électronique incorpore ces signaux. Elle mesure ensuite le temps de réaction (temps mis entre émission et réception de l'écho : l'onde qui a rebondi contre l'obstacle et qui a fini par revenir), ainsi que la vitesse de propagation du son dans l'air, ensuite calcule la distance entre le véhicule et l'obstacle.[14]

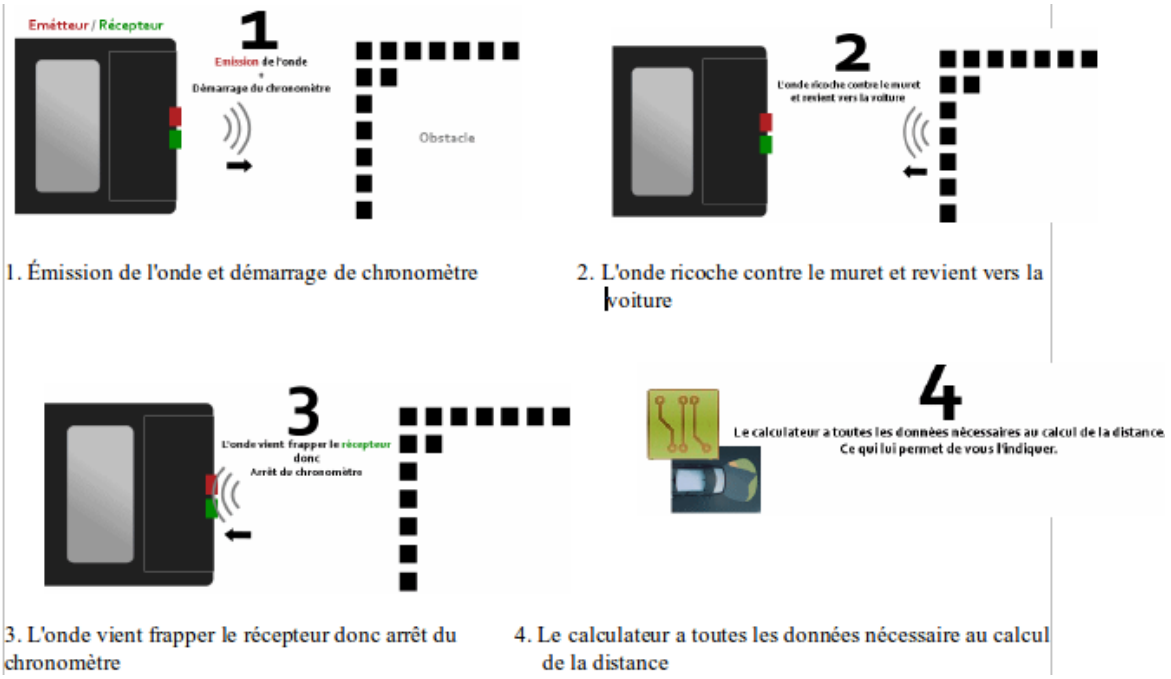


Fig. 3.7 Télémétrie par ultrasons

Dans l'optique d'optimiser cette fonction, nous nous inspirerons de la Citroën C4 Picasso pionnière de cette fonction; qui consiste en la mesure de l'espace de parking [13]. Ceci afin de s'assurer que l'espace de parking disponible est suffisamment large pour permettre au conducteur de s'y insérer. Elle est implémentée à partir d'un capteur à ultrason installé à la droite du véhicule, qui apprécie l'espace de parking en fonction de la vitesse du véhicule.

Fig. 3.8 Assistance au stationnement avec capteur ultrason à droite

État de la pression d'air dans les pneus

Les systèmes de contrôle permanents de la pression des pneumatiques équipent actuellement juste les véhicules de moyenne et de haute gamme.
A l'intérieur de chaque pneumatique, un capteur fournit les informations sur la pression à un micro-calculateur, associé à un émetteur RF (cf figure 3.9) Ce système est auto alimenté par une pile lithium-ion de durée de vie estimée à 10 ans[19].

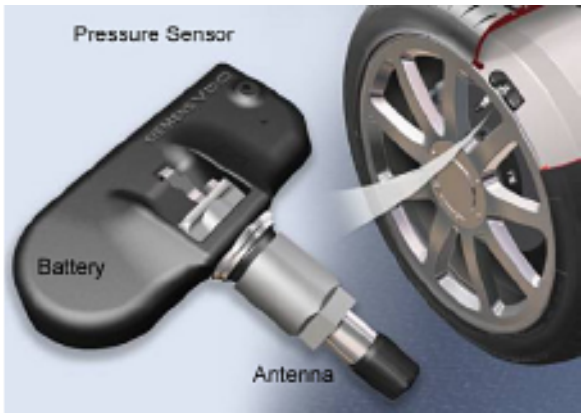


Fig. 3.9 Système TPMS (Tire Pressure Monitoring System)

Ce système a les avantages suivants:

- Détecte la base pression d'air dans les pneus;

- Accroît l'économie d'essence;
- Accroît la durée de vie des pneus;
- Réduit le risque d'accident.

3.3 Modèle de base de données

Pour une historisation et une analyse des paramètres qui nous permettrons d'anticiper sur d'éventuelles pannes, notre système doit être capable de sauvegarder et d'archiver sur des périodes relativement longues les informations en provenance de notre réseau de capteurs. L'implémentation de ce besoin doit se faire en gardant un œil sur deux principales contraintes à savoir:

- Les informations renvoyées par notre ensemble de capteurs peuvent varier très rapidement; ce qui entraîne immédiatement un accès très régulier à la base de données mise en place, ce qui n'est pas des moindres dans un environnement où les ressources sont aussi restreintes qu'un système embarqué.
- Le remplissage de notre BD avec les informations non structurées, nous conduira à un temps de traitement relativement long; toujours dans notre système très contraignant

La solution pour la prise en compte de ces contraintes d'archivage réside sur le choix du SGBD.

3.3.1 Choix du SGBD

La majorité des systèmes de gestion de base de données est construite selon le paradigme client-serveur, le moteur de base de données fonctionne dans un espace d'exécution qui lui est propre, voire sur une machine différente.
SQLite [9], au contraire, est directement intégrée dans l'application qui utilise son moteur de base de données. L'accès à une base de données SQLite se fait par l'ouverture du fichier qui lui est propre, avec ses déclarations, ses tables et ses index mais aussi ses données.
Cette caractéristique rend l'accès aux données:

- Plus sécurisé;
- Plus structuré;
- Ne portant pas atteinte à la facilité de déploiement de l'application qui l'utilise;

- La suppression de l'intermédiaire entre l'application et les données permet également de réduire la latence d'accès aux données comparée au paradigme client-serveur.

Les caractéristiques intrinsèques d'SQLite font de lui une solution incontournable, aux différentes contraintes de stockage que notre système devra faire face.

3.4 Conclusion

Ce chapitre, était consacré à l'analyse de notre système à mettre en oeuvre. Cette analyse s'est faite sur deux grands axes à savoir:

- Analyser les pannes assez fréquentes en automobile et trouver pour chaque panne un capteur adéquat pour son suivi;
- Le stockage des données, car l'une de nos problématiques est le suivi des pannes; le dit suivi passe par l'archivage structuré des informations renvoyées par notre ensemble de capteurs.

Le prochain chapitre, sera articulé autour de la modélisation, afin de faire cohabiter de façon harmonieuse les résultats de la présente analyse dans un système.

Chapitre 4

Conception

Sommaire

Introduction

Nous présentons dans ce chapitre, l'aspect conceptuel du système à mettre en œuvre suivant deux architectures à savoir celle logicielle et celle matérielle/technologique.
Pour ce qui est de l'architecture logicielle nous partirons des résultats obtenus à l'analyse pour une modélisation conceptuelle avec le langage de description logicielle/matérielle AADL. Quant'à celle matérielle/technologique nous présenterons les nœuds du système et les différentes technologies déployées pour leur mise en œuvre.

4.1 Architecture logicielle

4.1.1 Modèle de conception AADL

Nous présentons ici la conception AADL de notre application. Le point d'entré de la dite conception se fait par la déclaration d'un composant de type; ***systeme_VBD*** dont la présentation textuelle et celle graphique (figure 4.1) sont respectivement données ci-dessous:

system *systeme_VBD*
end *systeme_VBD*
Représentation textuelle du composant ***systeme_VBD***

Fig. 4.1 Représentation graphique de ***systeme_VBD***

Cette déclaration représente un modèle très abstrait où ce composant de catégorie ***system*** peut être vu comme une boite noire.
Ce composant est constitué de quatre sous-systèmes à savoir (figure 4.2):

- ***sub-system_Engine*** sous système où s'exécutent les commandes de contrôle du moteur;
- ***sub-system_Mechanic*** modélise le composant où s'exécutent les commandes de contrôle des paramètres mécaniques du véhicule;
- ***sub-system_PAssist*** gère les commandes d'aide au stationnement;
- ***sub-system_Communication*** assure la communication du système avec l'extérieur.

Le sous-système « *sub-system_Engine* »

Ce composant est lié en entrée à des capteurs, modélisé par des composants de la catégorie *device*, ceux-ci permettront de détecter:

- Le niveau du carburant (*fuel_level_sensor*);
- Le niveau d'huile (*oil_level_sensor*);

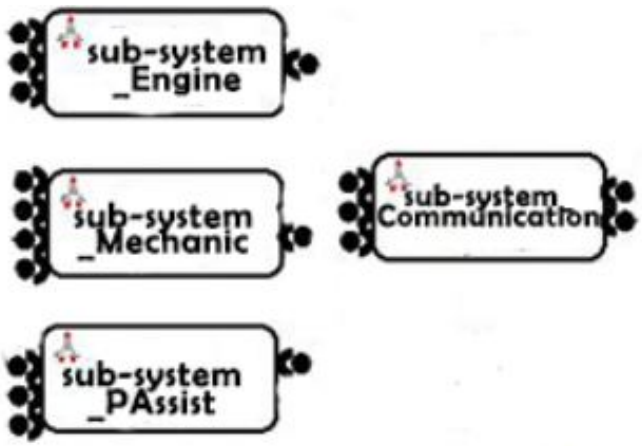


Fig. 4.2 Représentation graphique des sous-systèmes

- La température du moteur (*engine_temp_sensor*).

La représentation textuelle et celle graphique (figure 4.4) d'un composant de type *device* sont respectivement données comme suit:

```
device fuel_level_sensor
features
    fuel_level : out data port VBD::icd::Float_Type;
flows
    f0 : flow source fuel_level;
end fuel_level_sensor;
```

Fig. 4.3 Représentation textuelle du *device* ***fuel_level_sensor***

La figure 4.3 montre que ce *device*, a une interface de sortie qui retourne la donnée *fuel_level* de type *réel*; renseignant sur le niveau du carburant contenu dans le réservoir

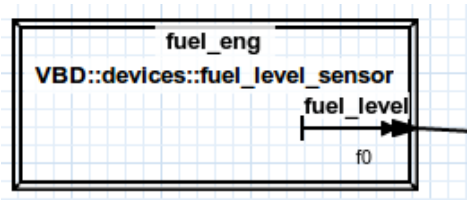


Fig. 4.4 Représentation graphique du *device* ***fuel_level_sensor***

Les composants *oil_level_sensor* et *engine_temp_sensor* présentent une structure similaire à ce cas. Ces différents capteurs (*device*) sont directement gérés par des tâches qui sont des composant de type *thread*. Ces dernières récupèrent les données retournées par les capteurs; pour ce sous-système nous avons les tâches suivantes:

- Le *thread* ***fuel_level_estimate_thr***, gère le capteur *fuel_level_sensor*;
- Le *thread* ***oil_level_estimate_thr***, gère le capteur *oil_level_sensor*;
- Le *thread* ***engine_temp_thr***, *gère le capteur engine_temp_sensor*.

La présentation textuelle du *thread* ***fuel_level_estimate_thr*** est présentée comme suit:

```
thread fuel_level_estimate_thr
features
    fuel_level  : in data port VBD::icd::Float_Type;
    fuel_level_out  : in out data port VBD::icd::msg_data;
flows
    f0 : flow path fuel_level -> fuel_level_out;
properties
    Dispatch_Protocol => Periodic;
    Priority => 3;
    Period        => 600000ms;
    compute_execution_time => 150ms .. 200ms;
end fuel_level_estimate_thr;
```

Fig. 4.5 Représentation textuelle du *thread* ***fuel_level_estimate_thr***

La figure 4.5 montre que ce *thread*, a deux interfaces dont une de sortie qui retourne la donnée *fuel_level_out* de type *message* vers le sous-système *subsystemCommunication* qui sera défini plus en bas, tandis que celle d'entrée reçoit la donnée *fuel_level* du capteur *fuel_level_sensor*. Cette tâche a les propriétés suivantes:

- Elle est périodique (*periodic*) de période *600.000ms* soit 10min;
- De priorité *3*, donc de faible priorité;
- Un temps d'exécution (*compute_execution_time*) allant de 150ms à 200ms.

Les tâches *oil_level_estimate_thr* et *engine_temp_thr* ont une structure semblable à celle-ci.

Pour une organisation fluide de ce flot de tâches le langage AADL suggère de la regrouper en processus du type composant « ***process*** ». Les tâches de ce sous-système ont été regroupé en deux processus:

- Le *process* ***fluid_level_estimate_process***, pour la prise en charge des fluides du moteur il gère les tâches *fuel_level_estimate_thr* et *oil_level_estimate_thr*;
- Le *process* ***engine_parameter_process***, gère le paramètre (température) du moteur à partir du *thread engine_temp_thr*.

La présentation textuelle (figure 4.6) et celle graphique (figure 4.7) du *process* ***fluid_level_estimate_process*** sont les suivants:

```
process fluid_level_estimate_process
features
    oil_level   : in data port VBD::icd::Float_Type;
    fuel_level  : in data port VBD::icd::Float_Type;
    transit_dataB   : out data port VBD::icd::msg_data;
flows
    f0 : flow path oil_level -> transit_dataB;
    f1 : flow path fuel_level -> transit_dataB;

end fluid_level_estimate_process;

process implementation fluid_level_estimate_process.i
subcomponents
    thr_oil     : thread oil_level_estimate_thr.i;
    thr_fuel    : thread fuel_level_estimate_thr.i;
connections
    c0 : port oil_level -> thr_oil.oil_level;
    c1 : port fuel_level    -> thr_fuel.fuel_level;
    c2 : port thr_oil.oil_level_out -> transit_dataB;
    c3 : port thr_fuel.fuel_level_out   -> transit_dataB;
flows
    f0 : flow path oil_level -> c0 -> thr_oil.f0 -> c2 -> transit_dataB;
    f1 : flow path fuel_level-> c1 -> thr_fuel.f0 -> c3-> transit_dataB;
end fluid_level_estimate_process.i;
```

Fig. 4.6 Représentation textuelle du *process* ***fluid_level_estimate_process***

Sa représentation textuelle montre que ce processus, trois interfaces dont deux d'entrées *fuel_level* et *oil_level* et une de sortie *transit_dataB*.
Le *process* ***fluid_level_estimate_process*** a sous sa supervision deux instances des tâches *fluel_level_estimate_thr* et *oil_level_estimate_thr* dont les connexions sont décrits de ***c0*** à ***c3***.

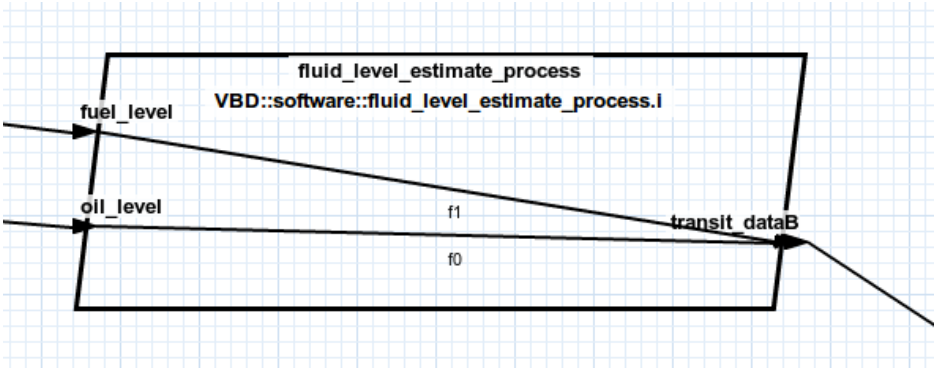


Fig. 4.7 Représentation graphique du *process* ***fluid_level_estimate_process***

Le processus *engine_parameter_process* a structure similaire à cet représentation.
La représentation structurale du sous-système « ***sub-system_Engine*** » est présentée par la figure 4.8:

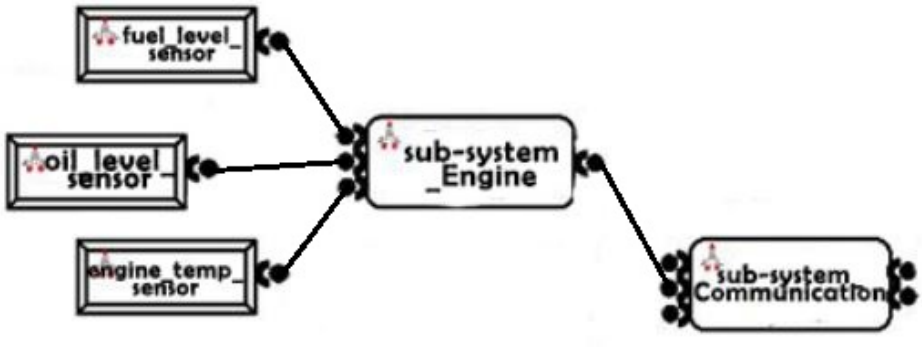


Fig. 4.8 Représentation graphique du sous-système «***sub-system_Engine***»

Le sous-système « *sub-system_Mechanic* »

Ce composant est lié en entrée à des capteurs, ceux-ci permettront de surveiller :

- La pression d'air dans les pneus (*tire_preassure_sensor*);
- Le niveau de charge de la batterie (*battery_level_sensor*);
- L'usure des plaquettes de frein (*brake_wear_right|left_sensor*);
- La vitesse de l'automobile (*effet_hall_sensor*).

La présentation textuelle et celle graphique (figure 4.10) du capteur *tire_preassure_sensor* sont respectivement données comme suit:

```
device tire_preassure_sensor
features
    preassure : out data port VBD::icd::Float_Type;
flows
    f0 : flow source preassure;
properties
    compute_execution_time => 250 ms .. 300 ms;
end tire_preassure_sensor;
```

Fig. 4.9 Représentation textuelle de ***tire_preassure_sensor***

La figure 4.9 montre qu'il a une interface de sortie qui retourne la donnée *preassure* de type *réel* et dispose d'un temps d'exécution allant de 250ms à 300ms

Ces capteurs sont directement gèrés par les tâches (*thread*) suivantes:

- Le *thread* ***preassure_analyse_thr***, gère le capteur *tire_preassure_sensor*;
- Le *thread* ***enegy_acquisition_thr***, gère le capteur *battery_level_sensor*;
- Le *thread* ***state_brake_analyse_thr***, gère le capteur *brake_wear_right|left_sensor*;

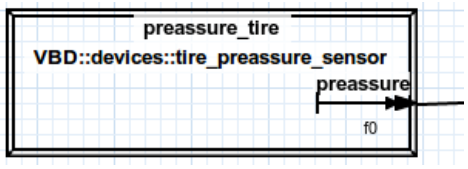


Fig. 4.10 Représentation graphique de ***tire_preassure_sensor***

- La *thread* ***pulse_analyse_thr****, gère le capteur effet_hall_sensor*.

La représentation textuelle du *thread preassure_analyse_thr* est présentée comme suit:

```
thread preassure_analyse_thr
features
    preassure       : in data port VBD::icd::Float_Type;
    preassure_out   : out data port VBD::icd::msg_data;
flows
    f0 : flow path preassure -> preassure_out;
properties
    Dispatch_Protocol => Periodic;
    Period        => 600000ms;
    compute_execution_time => 260 ms .. 310 ms;
    reference_processor => classifier (VBD::plateform::processors_atmega::ATMEGA328p);
    sei::instructionsperdispatch => 1.0 kipd .. 1.05 kipd;
end preassure_analyse_thr;
```

Fig. 4.11 Représentation textuelle de ***preassure_analyse_thr***

La figure 4.11 montre qu'il a deux interfaces dont une de sortie qui retourne la donnée *preassure_out* de type *message* vers le sous-système *subsystemCommunication* qui sera défini plus en bas, tandis que celle d'entrée reçoit la donnée *preassure* du capteur *tire_preassure_sensor*. Cette tâche a les propriétés suivantes:

- Elle est périodique (*periodic*) de période *600.000ms* soit 10min;
- De priorité *3*, donc de faible priorité;
- Un temps d'exécution (*compute_execution_time*) allant de 260ms à 310ms.
- Elle s'éxecutera sur un processeur *ATMEGA328p*, avec une vitesse de transit de données d'environ 1kbit

Ce sous-système est subdivisé en deux processus:

- Le *process* ***enegy_acquisition_process***; pour la prise en charge du paramètre energetique du véhicule, il gère la tâche *enegy_acquisition_thr*;
- Le *process* ***motricity_analyse_process***, gère les paramètres relatifs à la motricité du véhicule à partir de *state_brake_analyse_thr*, *preassure_analyse_thr* et *pulse_analyse_thr*.

La présentation textuelle (figure 4.12) et celle graphique (figure 4.13) du *process* ***motricity_analyse_process*** sont les suivants:

```
process motricity_analyse_process
features
    state_brake : in data port VBD::icd::enum_type;
    pulse       : in data port VBD::icd::Float_Type;
    preassure   : in data port VBD::icd::Float_Type;
    transit_dataB   : out data port VBD::icd::msg_data;
    transit_dataG   : out data port VBD::icd::msg_data;

flows
    f0 : flow path state_brake -> transit_dataB;
    f1 : flow path pulse       -> transit_dataB;
    f2 : flow path preassure   -> transit_dataB;
    f3 : flow path preassure   -> transit_dataG;
end motricity_analyse_process;

process implementation motricity_analyse_process.i
subcomponents
    thr_ste : thread state_brake_analyse_thr.i;
    thr_pls : thread pulse_analyse_thr.i;
    thr_prs : thread preassure_analyse_thr.i;
connections
    c0 : port state_brake -> thr_ste.state_brake;
    c1 : port pulse -> thr_pls.pulse;
    c2 : port preassure -> thr_prs.preassure;
    c3 : port thr_pls.speed -> transit_dataB;
    c4 : port thr_ste.state_brake_out -> transit_dataB;
    c5 : port thr_prs.preassure_out -> transit_dataB;
    c6 : port thr_prs.preassure_out -> transit_dataG;
flows
    f0 : flow path state_brake -> c0 -> thr_ste.f0 -> c4 -> transit_dataB;
    f1 : flow path pulse -> c1 -> thr_pls.f0 -> c3 -> transit_dataB;
    f2 : flow path preassure -> c2 -> thr_prs.f0 -> c5 -> transit_dataB;
    f3 : flow path preassure -> c2 -> thr_prs.f0 -> c6 -> transit_dataG;

end motricity_analyse_process.i;
```

Fig. 4.12 Représentation textuelle de ***motricity_analyse_process***

Sa représentation textuelle montre que ce processus, cinq interfaces dont:

- Trois en entrée: *state_brake*, *pulse* et *preassure*;
- Deux en sortie: *transit_dataB* pour la communication par Bluetooth et *transit_dataG* pour celle par GPRS.

Le *process* ***motricity_analyse_process*** a sous sa supervision les instances des tâches:

- *state_brake_analyse_thr*;
- *pulse_analyse_thr*;
- *preassure_analyse_thr*.

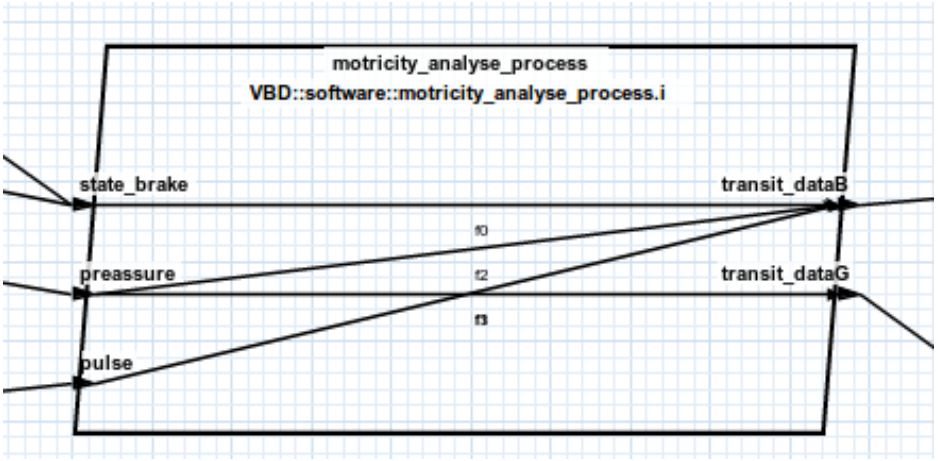


Fig. 4.13 Représentation graphique du *process* ***motricity_analyse_process***

Leurs inter-connexions sont décrites de ***c0*** à ***c6***.

La représentation structurale du sous-système « ***sub-system_Mechanic*** » est présentée par la figure 4.14:

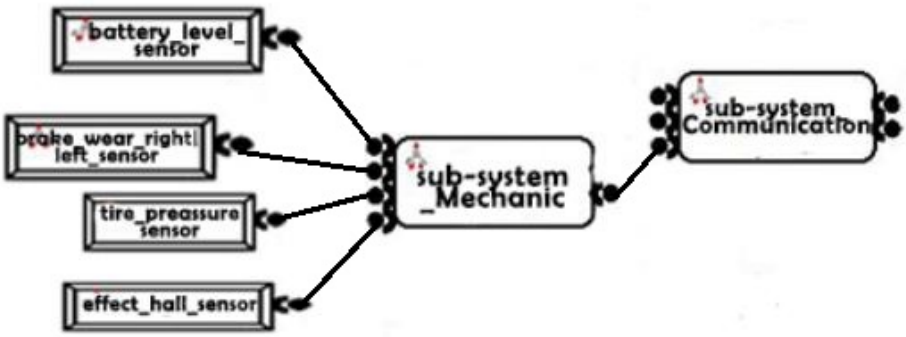


Fig. 4.14 Représentation graphique du sous-système «***sub-system_Mechanic***»

Le sous-système « *sub-system_PAssist* »

Il est lié à des capteurs: deux capteurs à ultrasons à l'arrière du véhicule (*ultrason_BLeft|BRight_sensor*), un à droite (*ultrason_right_sensor*) et d'un capteur à effet Hall pour la vitesse de l'automobile (*effet_hall_sensor*).

Il est lié en entrée à des capteurs, ceux-ci permettront d'assister le conducteur lors de son stationnement dans un parking :

- Un capteur de télémétrie à ultrasons à l'arrière droit du véhicule (*ultrason_BRight_sensor*);
- Un capteur de télémétrie à ultrasons à l'arrière gauche du véhicule (*ultrason_BLeft_sensor*);

- Un capteur de télémétrie à ultrasons à la droite du véhicule (*ultrason_right_sensor*);
- Un capteur à effet Hall pour déterminer la vitesse de l'automobile (*effet_hall_sensor*).

La présentation textuelle et celle graphique (figure 4.16) du capteur *effet_hall_sensor* sont respectivement données comme suit:

```
thread eval_dispo_thr
features
    pulse       : in data port VBD::icd::Float_Type;
    eval_largeur    : in out data port VBD::icd::msg_data;
flows
    f0 : flow path pulse -> eval_largeur;
properties
    Dispatch_Protocol => Periodic;
    Priority => 2;
    Period        => 10ms;
    compute_execution_time => 1ms .. 2ms;
end eval_dispo_thr;
```

Fig. 4.15 Représentation textuelle de ***effet_hall_sensor***

La figure 4.15 montre qu'il a une interface de sortie qui retourne la donnée *pulse* de type *réel* qui permettra de déterminer le nombre de tour effectué par la roue, et dispose d'un temps d'exécution allant de 1ms à 2ms

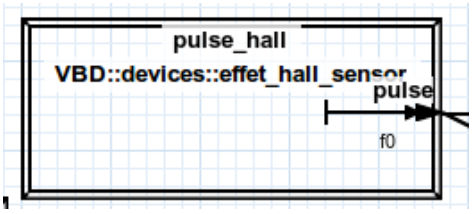


Fig. 4.16 Représentation graphique de ***effet_hall_sensor***

Les composants *ultrason_BRight_sensor*, *ultrason_BLeft_sensor* et *ultrason_right_sensor* ont une représentation assez similaire.
Ces capteurs sont directement gèrés par les tâches (*thread*) suivantes:

- Le *thread* ***eval_dispo_thr***, s'assure que l'espace de parking soit suffisament large pour le véhicule, ceci à partir du capteur *ultrason_right_sensor*;
- Le *thread* ***eval_longueur_thr***, gs'assure que l'espace de parking est suffisament long pour le véhicule, ceci à partir du capteur *effet_hall_sensor*;
- Le *thread* ***behind_control_thr***, controle le distance entre de pontentiel obstacle à l'arrière du véhicule à partir des capteurs *ultrason_BRight_sensor* et *ultrason_BLeft_sensor*.

```
thread eval_dispo_thr
features
    pulse       : in data port VBD::icd::Float_Type;
    eval_largeur    : in out data port VBD::icd::msg_data
flows
    f0 : flow path pulse -> eval_largeur;
properties
    Dispatch_Protocol => Periodic;
    Period        => 10ms;
    compute_execution_time => 1ms .. 2ms;
end eval_dispo_thr;
```

Fig. 4.17 Représentation textuelle de ***eval_dispo_thr***

La représentation textuelle du *thread effet_hall_sensor* est présentée comme suit:
La figure 4.17 montre qu'il a deux interfaces dont une de sortie qui retourne la donnée *eval_largeur* de type *message* celle-ci apprécie la largeur de l'espace de parking, tandis que celle d'entrée reçoit la donnée *pulse* du capteur *effet_hall_sensor*. Cette tâche a les propriétés suivantes:

- Elle est périodique (*periodic*) de période 10ms;
- De priorité *2*;
- Un temps d'exécution (*compute_execution_time*) allant de 1ms à 2ms.

Ce sous-système est subdivisé en deux processus:

- Le *process* ***test_free_parking_process***; pour la supervision de la disponibilité en largeur et en longueur de l'espace de parking, il gère les tâches *eval_dispo_thr* et *eval_longueur_thr*;
- Le *process* ***behind_control_process***, veille sur l'absence de potentiels obstacles à l'arrière du véhicule à partir de la tâche *behind_control_thr*.

La présentation textuelle (figure 4.18) et celle graphique (figure 4.19) du *process* ***test_free_parking_process*** sont les suivants:
La représentation ci-dessus montre que ce processus, trois interfaces dont:

- Deux en entrée: *pulse*, *distance*;
- Une en sortie: *transit_dataB* vers la communication par Bluetooth.

Le *process* ***test_free_parking_process*** a sous sa supervision les instances des tâches:

```
process  test_free_parking_process
features
    pulse       : in data port VBD::icd::Float_Type;
    distance    : in data port VBD::icd::Float_Type;
    transit_dataB   : out data port VBD::icd::msg_data;

flows
    f0 : flow path pulse -> transit_dataB;
    f1 : flow path distance -> transit_dataB;
    f2 : flow path pulse -> transit_dataB;
end  test_free_parking_process;

process implementation test_free_parking_process.i
subcomponents
    thr_dsp : thread eval_dispo_thr;
    thr_lng : thread eval_longueur_thr;
connections
    c0 : port pulse -> thr_dsp.pulse;
    c1 : port thr_dsp.eval_largeur -> transit_dataB;
    c2 : port distance -> thr_lng.distance;
    c3 : port thr_lng.eval_longueur -> transit_dataB;
    c4 : port pulse -> thr_lng.pulse;
flows
    f0 : flow path pulse -> c0 -> thr_dsp.f0 -> c1 -> transit_dataB;
    f1 : flow path distance -> c2 -> thr_lng.f1 -> c3 -> transit_dataB;
    f2 : flow path pulse -> c4-> thr_lng.f0 -> c3 -> transit_dataB;
end test_free_parking_process.i;
```

Fig. 4.18 Représentation textuelle de ***test_free_parking_process***

- *eval_dispo_thr*;
- *eval_longueur_thr*.

Leurs inter-connexions sont décrites de ***c0*** à ***c4***.

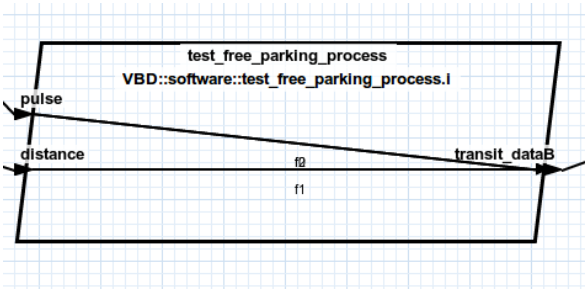


Fig. 4.19 Représentation graphique du *process* ***test_free_parking_process***

Le processus *behind_control_process* ayant une structure similaire à celle-ci.
La représentation structurale du sous-système « ***sub-system_Mechanic*** » est présentée par la figure 4.14:

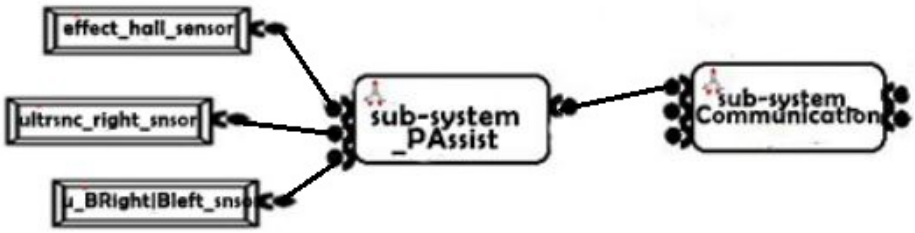


Fig. 4.20 Représentation graphique du sous-système «***sub-system_PAssist***»

Le sous-système « *sub-system_Communication* »

Il est lié à deux "gateways" (*device*), pour le transport de l'information à l'extérieur de notre système, on a:

- Un module "Bluetooth" (*bluetooth_Module*), pour une communication Bluetooth;
- Une passerelle "GPRS" (*GPRS_Module*), pour une communication gsm;

La présentation textuelle et celle graphique (figure 4.22) du module *bluetooth_Module* sont respectivement données comme suit:

```
device bluetooth_Module
features
    transit_data : in out data port VBD::icd::msg_data;
    transit_dataB : in out data port VBD::icd::msg_data;
flows
    f0 : flow source transit_data;
    f1 : flow sink transit_dataB;
properties
    compute_execution_time => 150ms .. 200ms;
end bluetooth_Module;
```

Fig. 4.21 Représentation textuelle du ***bluetooth_Module***

La figure 4.21 montre qu'il a deux interfaces d'entrée-sortie: *transit_data* et *transit_dataB* pour l'envoi et la réception des messages du système avec extérieur.

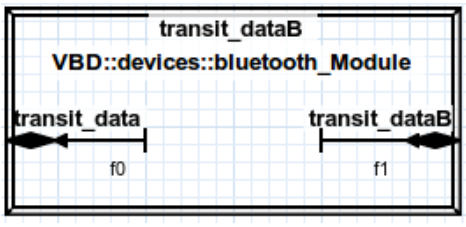


Fig. 4.22 Représentation graphique du ***bluetooth_Module***

Ces capteurs sont gèrés par les tâches (*thread*) suivantes:

- Le *thread* ***bluetooth_acquisition_thr***, pour les transmissions Bluetooth;
- Le *thread* ***gprs_acquisition_thr***, pour les transmissions gsm.

La représentation textuelle du *thread bluetooth_acquisition_thr* est présentée comme suit:

```
thread bluetooth_acquisition_thr
features
    transit_dataB   : in out data port VBD::icd::msg_data;
    transit_data    : in out data port VBD::icd::msg_data;
properties
    Dispatch_Protocol => Aperiodic;
    Priority => 1;
    compute_execution_time => 250ms .. 300ms;
end bluetooth_acquisition_thr;
```

Fig. 4.23 Représentation textuelle de ***bluetooth_acquisition_thr***

La figure 4.23 montre que c'est une tâche de priorité maximale, car égale à 1. Elle est "apériodique" parce qu'elle se réveille dès réception d'un message à transférer.
La représentation structurale du sous-système « ***sub-system_Mechanic*** » est présentée par la figure 4.14:

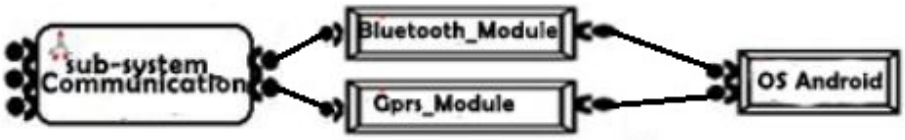


Fig. 4.24 Représentation graphique du sous-système «***sub-system_Communication***»

La représentation graphique de notre système ***systeme_VBD*** est donnée par la figure 4.25:

4.2 Architecture matérielle et technologique

L'architecture matérielle et technologique de notre système est formalisée par la figure 4.26.

4.2.1 Connexion 1 :: AADL - Cheddar

Après une modélisation AADL, nous obtenons un ensemble de tâches interconnectées entre elles chacune ayant une priorité et une période qui lui est propre. Cette section a pour objectif de simuler l'ordonnancement des tâches définies dans le modèle AADL.
La première étape est celle du choix du bon algorithme d'ordonnacement ayant les contraintes suivantes:

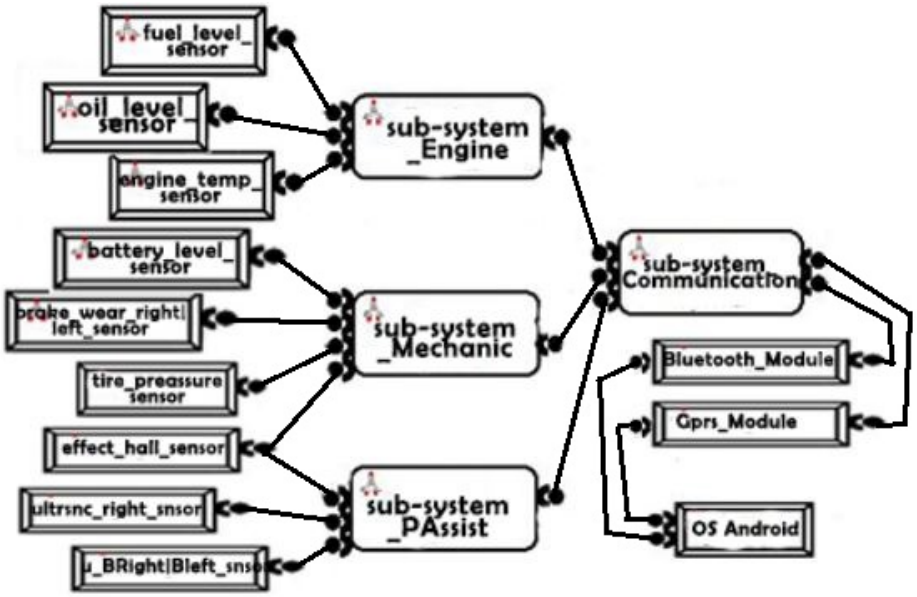


Fig. 4.25 Représentation graphique du sous-système «***systeme_VBD***»

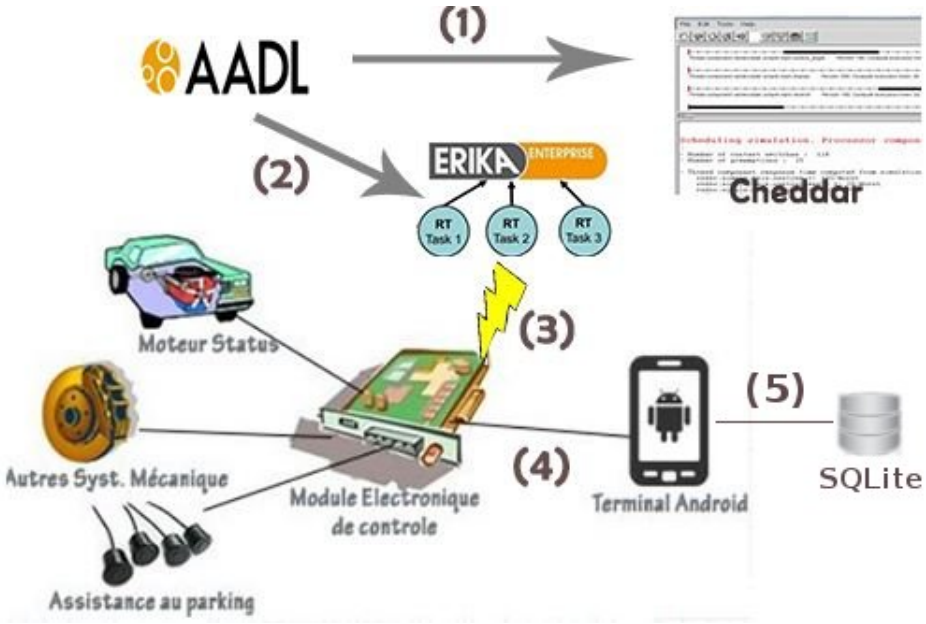


Fig. 4.26 Architecture matérielle/echnologique

- Préemptif;
- Gestion des priorités.

Compte tenu de ces contraintes nous avons choisi, l'algorithme d'ordonnancement : ***«Rate Monotonic avec priorité fixe»***.
La simulation d'ordonnancement avec cet algorithme par Cheddar, est décrit comme suit; la présentation de Cheddar au chapitre *Outils technologiques utilisés* de *l'état de l'art*, nous présente les deux modes d'exécution avec Cheddar:

- Tests de faisabilité;
- Moteur de simulation.

Tests de faisabilité

Le test de faisabilité a retourné le résultat présenté à la figure 4.27: À partir, de cette figure

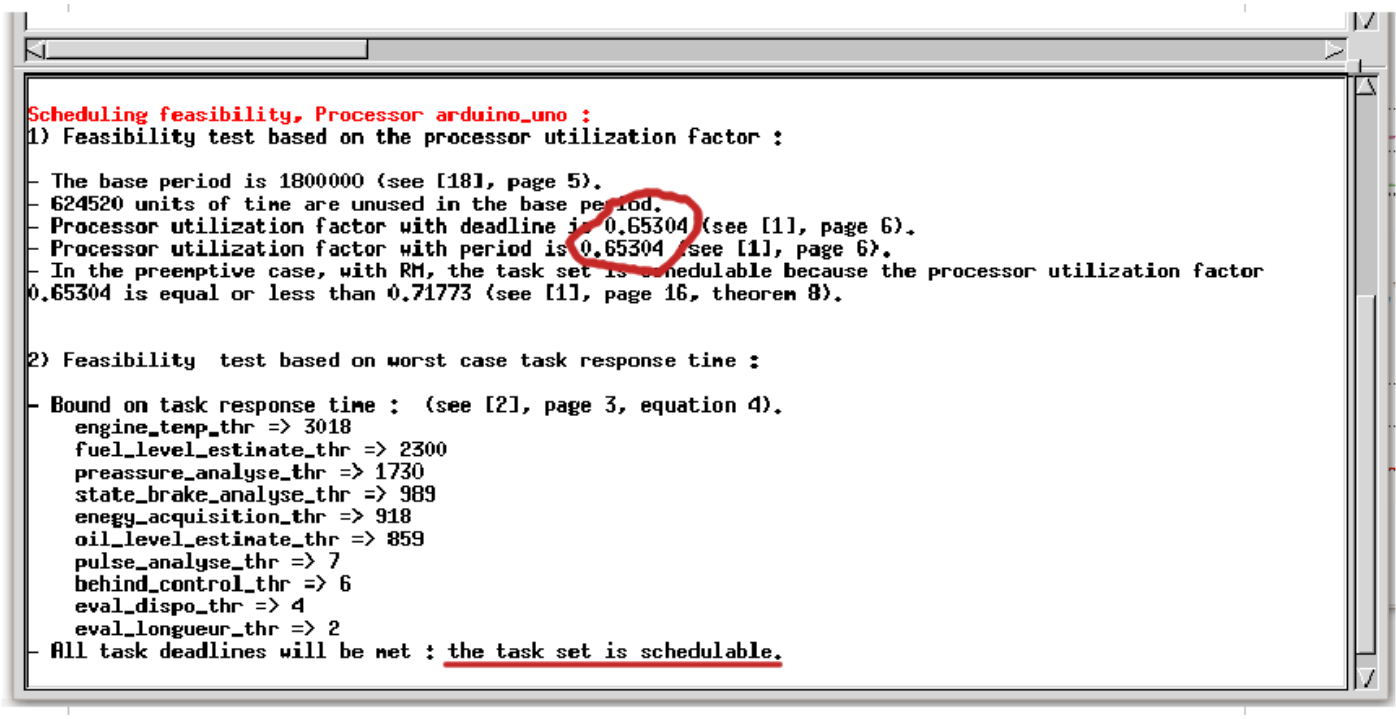


Fig. 4.27 Test de faisabilité avec *Cheddar*

on peut lire que pour l'ordonnancement de ces tâches un processeur *ATmega328* a un taux d'utilisation d'environ **0,65** avec l'algorithme d'ordonnancement *«Rate Monotonic avec priorité fixe»*, ce taux étant intérieur à 1, donc cet ensemble de tâches est *ordonnable*.

Moteur de simulation

La simulation de l'ordonnancement des tâches du modèles par Cheddar, présentée à la figure 4.28 À partir, de cette figure on peut lire qu'on a 67 changements de contexte (passage

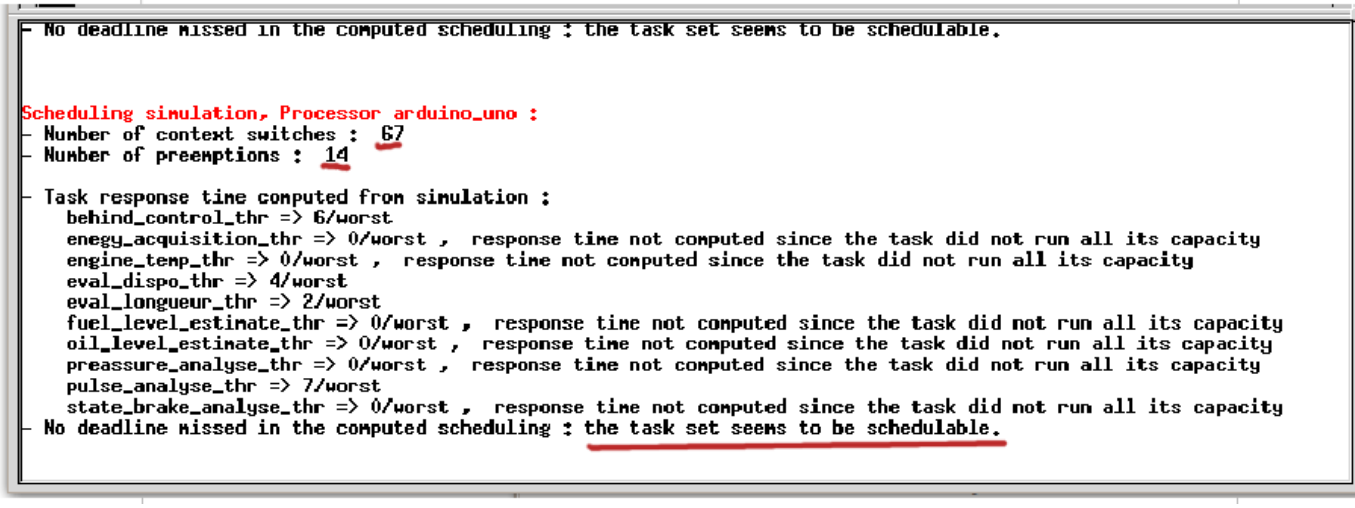


Fig. 4.28 Simulation avec *Cheddar*

d'une tâche à une autre) et 14 préemptions (mise en attente d'une tâche en cours d'exécution). Cette simulation, nous permet de conclut que ces tâches sont ordonnançable.

4.2.2 Connexion 2 :: AADL - ERIKA Enterprise

L'implémentation algorithmique du modèle obtenu avec le langage AADL, se fera à partir d'un environnement qui prend en charge la propriété multitâche de notre modèle.
L'exemple du clignotement des 3 leds, présenté en annexe montre que le framework de développement standard d'Arduino; ne prend pas charge la contrainte multitâche: il est *monotâche*.
Ce qui justifie l'utilisation d'ERIKA Enterprise qui est un environnement qui prend en charge, les cartes *Arduino* avec la propriété *multitâche* (figure 4.29)

4.2.3 Connexion 3 :: ERIKA Enterprise - Arduino

Le code géneré par l'environnement ERIKA Enterprise, sera flashé dans la carte à microprocesseur *Arduino Uno*. Cette carte est configurée pour ce travail comme suit:

- Elle supervisera la collecte des données de 8 capteurs et de 2 extensions de communication;

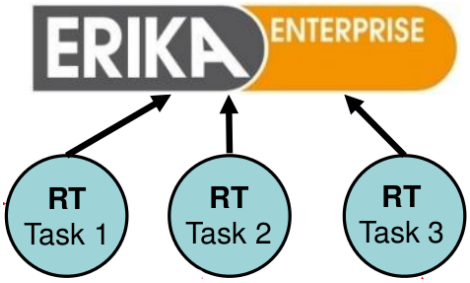


Fig. 4.29 Multitâche avec *ERIKA*

- Pour une utilisation totale de 10 ports numériques et 6 analogiques.

Tous les capteurs seront installés de façon permanente sur le microcontrôleur; le schéma du brochage définitif est décrit par la figure ci dessous:

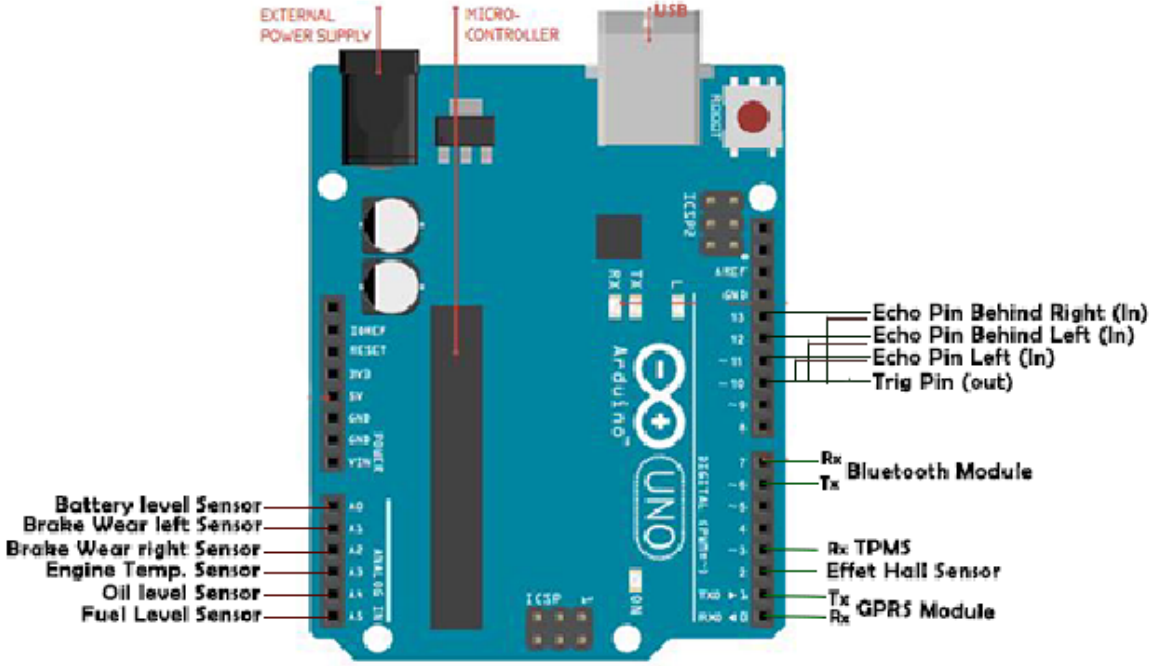


Fig. 4.30 Brochage de la carte Arduino

4.2.4 Connexion 4 :: Arduino - Android

Les données lues par *Arduino* de notre ensemble de capteurs, doivent être communiquées vers l'extérieur pour affichage et suite de traitement. Le nœud retenu pour affichage des données est Android.
Après avoir passé en revus les protocolos de communication possible entre une carte Arduino

et le système Android; au chapitre *Outils technologiques utilisés* de *l'état de l'art* notre choix ce porte sur les modes de communication:

- ***Bluetooth*** : de part sa portée de 10m et son débit de 2Mbits/s;
- ***GPRS*** : car il offre au conducteur la possibilité d'obtenir un diagnostic de son automobile quelque en soit le lieu où il se trouve du moment qu'il a accès au réseau GPRS compatible.

4.2.5 Connexion 5 :: Android - SQLite

Pour l'historisation et l'archivage des informations relatives au suivie des paramètres sensibles du véhicule, afin d'optimiser la fonction prévention des pannes par notre système. Le stockage des données se fera sur le nœud *Android*, à partir du SGBD conçu pour système embarqué: **SQLite**.

4.3 Conclusion

Dans ce chapitre, nous avons présenté une méthodologie d'ingénierie dirigée par les modèles. Où nous avons commencé par modéliser les résolutions du chapitre analyse. En utilisant le langage AADL dont le modèle obtenu est constitué de 4 sous-systèmes (*sub-system_Engine*, *sub-system_Mechanic*, *sub-system_PAssist* et *sub-system_Communication*). Chacun de ces sous-systèmes a en entré plusieurs capteurs, qui sont managés par diverses tâches (*thread*); ces tâches sont à leur tour regroupées en processus (*process*) dont les sorties sont envoyées vers l'extérieur par le sous-système *sub-system_Communication* qui est le plus prioritaire du système.
Le modèle ainsi obtenu a été stimulé et validé par le simulateur d'ordonnancement Cheddar. En s'appuyant sur les propriétés et contraintes soulignées par AADL, nous avons implémenté puis flashé du code généré à partir de l'environnement ERIKA Enterprise; dans un microprocesseur Arduino Uno.
Le chapitre suivant présentera les résultats obtenus lors de l'implémentation et de la simulation de notre système.

Chapitre 5

Expérimentations et premières évaluations

Sommaire

Introduction

Après conception et modélisation de notre système, à partir du langage AADL et du simulateur Cheddar. Nous aborderons dans ce chapitre l'implémentation logicielle et matérielle de notre solution.
Dans ce chapitre, nous présenteront les concepts de base de fonctionnement des capteurs afin de les simuler, les concepts de développement d'une application avec d'une part ERIKA et l'OS Android d'autre part.

5.1 Simulation de nos capteurs

Cette section sera consacrée à la simulation des capteurs étudiés à la section 2 du chapitre *analyse*. Ces simulations ne se ferons non pas avec les capteurs et dispositifs réels et disponibles sur le marché, mais avec les capteurs de laboratoire et dispositifs expérimentaux ; ceci dans l'optique de s'assurer de la réactivité, la sûreté et le respect des différentes contraintes temps réel (les données arrivent elles au moment attendu?) du système.

5.1.1 Niveau de carburant, Niveau d'huile moteur et Température moteur

Comme décrit précédemment:
- C'est un flotteur relié à une résistance variable, qui permet d'estimer le niveau du carburant;
- Le niveau d'huile est mesuré à partir d'une différence de potentielle au borne du fil thermique, dû à la variation de la résistivité du fil en fonction du niveau d'huile dans le réservoir;
- La température du moteur est déterminée à l'aide de la variation de la valeur d'une résistance thermique *«NTC»*.

Nous constatons que ces trois fonctions sont axées autour d'une résistance variable, à cet effet elles peuvent être simulées à partir de la plus simple des résistances variables : le potentiomètre (figure 5.1).

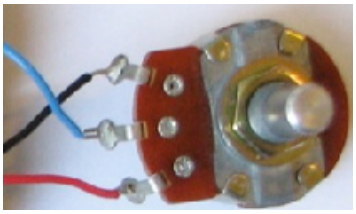


Fig. 5.1 Potentiomètre

5.1.2 Niveau de charge batterie

Le but de cette fonction est d'évaluer le niveau de charge de batterie dont le voltage normalisé est de 12Volts, or la plupart des microcontrôleurs fonctionnent avec de faibles de tensions généralement autour de 5 Volts. Dans le souci de ne pas détériorer notre microcontrôleur nous allons utiliser un « diviseur de tensions ».
La simulation de cette fonction se fera même suit, pour augmenter la sûreté du dispositif nous allons supposer que la tension d'entrée maximale soit de 24 Volts, car certaines voitures

robustes dispose de deux batterie de 12 volts montées en série. Pour mettre cette tension en une contrôlable par la carte à microcontrôleur nous allons utiliser un « diviseur de tensions »(figure 5.2) avec une résistance R1 de 1KOhm et une R2 de 4KOhm.

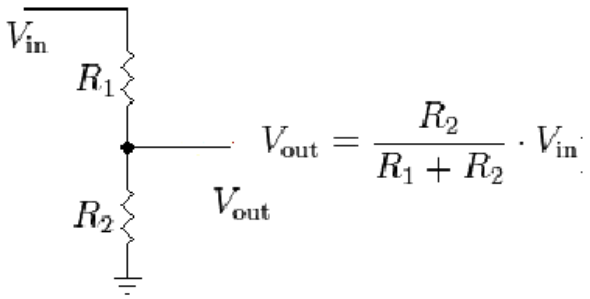


Fig. 5.2 Diviseur de tensions

5.1.3 État d'usure des plaquettes de freins

Son principe repose sur la fermeture et l'ouverture d'un circuit, cette fonction de fermeture d'ouverture est assuré par le capteur incorporé dans la plaquette de frein. Sa simulation sera faite avec un composant électronique «bouton poussoir»(figure 5.3) , il a pour fonction d'ouvrir et fermer un circuit.

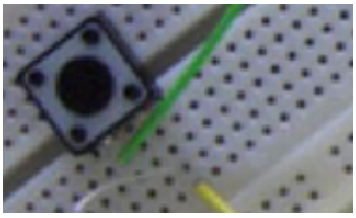


Fig. 5.3 Bouton poussoir

5.1.4 État de la pression d'air dans les pneus

Celles-ci est implémentée par un dispositif installé dans la roue, ce dernier relève la pression pneumatique et la renvoie au microcontrôleur par radio fréquence. Ce capteur étant rare et coûteux, nous ne simulerons que la transmission RF de ce dernier. Cette simulation sera faite par un émetteur – récepteur de radio fréquence(figure 5.4).

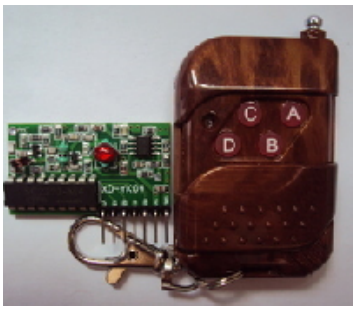


Fig. 5.4 Émetteur-Recepteur RF *Arduino*

5.1.5 Aide au stationnement

Cet implémentation, ce présente comme étant la plus avancée; car elle fera appelle simultanément à deux concepts: télémétrie par ultrasons et calcul de vitesse suivant le principe d'effet hall.
La simulation suivra la processus suivant :

- ***Simulation de la télémétrie par ultrason***: nous utiliserons le capteur à ultrason *HC SR-04* (figure 5.5);

Fig. 5.5 Capteur à ultrason *HC SR-04*

- ***Simulation par capteur à effet hall pour avoir la vitesse du véhicule***: cette simulation se fera à partir du capteur à effet Hall "*KY-00344E*" (figure 5.6);

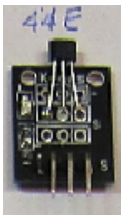


Fig. 5.6 Capteur à effet hall *KY-003 44E*

- Enfin une ***simulation croisée du capteur à ultrason et celui à effet hall pour évaluer l'espace de parking***: pour évaluer la disponibilité de l'espace de parking. Le calcul des distances sera fait selon la figure suivante (Figure 5.7):

.

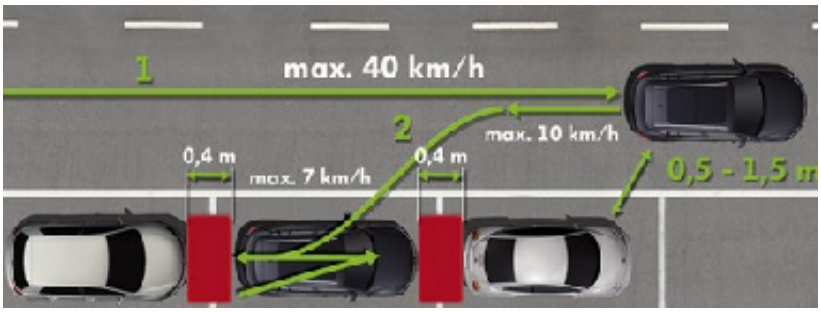


Fig. 5.7 Espacement requis pour un positionnement dans un espace de parking

5.2 ERIKA Enterprise

Le code source conçu par Erika Enterprise et flashé dans la carte Arduino Uno, et subdiviser en trois parties :

5.2.1 Partie logicielle

Le fichier *.oil* pour la description de la structure ERIKA

Elle présente la structure d'organisation des configurations OIL généré sous ERIKA pour notre système. Chaque boucle spécifie une tâche OSEK. Les priorités des tâche sont explicitement définies selon l'ordonnancement « Fixe-Priority »[22].

La partie traitement proprement dit du système

Cas de traitement d'un capteur analogique

TASK(energy_acquisition_thr)
voltage = analogRead(A0);
vout = (voltage*5.0)/1024.0;
voltageRead = vout/(R2/(R1+R2));
};

Ce bout de code li la valeur retournée par un capteur ou dispositif analogique (diviseur de tensions). Il obtient la valeur du capteur avec la fonction **analogRead**(port d'écoute du capteur) tel en arduino standard, cette valeur appartient à intervalle 0-1023. Le voltage de la batterie sera évalué à partir de la règle de trois suivi de l'équation du diviseur de tensions.

Cas de traitement d'un capteur numérique

TASK(pulse_analyse_thr) {
if (pulse >= 5) {
//Update RPM every 20 counts, increase this for better RPM resolution
mps = perimetreRoue*pulse*1000/(millis()-timeold);
pulse = 0;
timeold = millis();
}
if(((millis()-timeold)/1000) >=4)
mps=0; //On réinitialise la vitesse si l'on a fait + de 4sec sans détecter au + 5 fois la rotation de la roue
}
kph=mps/3.6;
distanceKM=distanceKM+(perimetreRoue*pulse*1000);
speed=kph;
distance=distanceKM;
};
if(((millis()-timeold)/1000) >=4) //mps=0; //On réinitialise la vitesse si l'on a fait + de 4sec sans détecter au + 5 fois la rotation de la roue }
kph=mps/3.6;
distanceKM=distanceKM+(perimetreRoue*wheelRevolution*1000);
valueRead[0]=kph;
valueRead[1]=distanceKM;
};
Le code ci dessus permet d'évaluer la vitesse et la distance parcourue par le véhicule. À partir d'un capteurs numérique (*capteur à effet hall*) qui génère 0 ou 1 selon qu'il soit excité ou pas, l'état du capteur digital est lu à partir de la fonction **digitalRead**(port d'écoute du capteur). Le principe de ce code est le suivant :

- On mulltiplie le nombre d'excitation (nombre de fois que le capteur a détecté un champs magnétique et a retourné 1), par la circonférence de la roue ceci afin d'obtenir la distance parcourue par l'automobile.

- Une fois après avoir obtenu la distance du véhicule, on la divise par le temps écoulé durant le calcul de la distance parcourue.

5.2.2 Partie matérielle

Test individuel de chaque capteur

Pour la réalisation de cette batterie de test, nous avons téléversé le code EIKA Enterprise dans la carte «Arduino Uno» à partir de la commande `arduinounoflash` saisie dans l'invite de commande à partir dossier d'exécution du projet [24].

Niveau de batterie *(figure 5.8)*

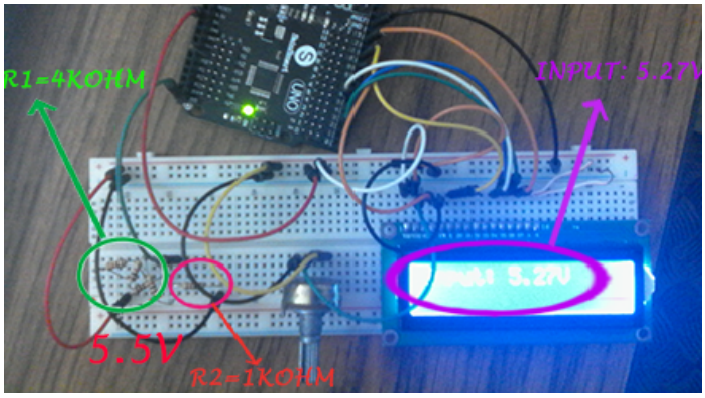


Fig. 5.8 Diviseur de tensions pour estimation du niveau de la batterie

Niveau réservoir à essence *(figure 5.9)*

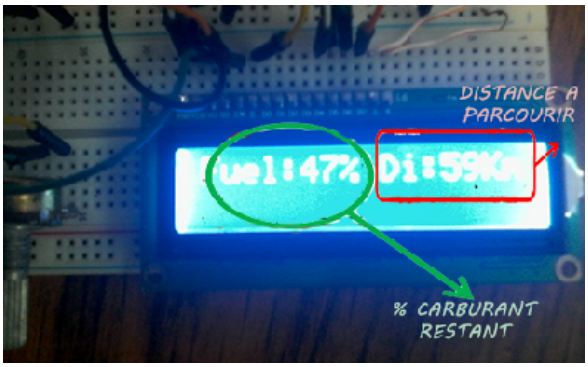


Fig. 5.9 Recueil d'information sur l'état du niveau de carburant dans le réservoir

Niveau d'huile dans le réservoir *(figure 5.10)*

Appréciation de la distance entre un obstacle et l'automobile *(figure 5.11)*

État des garnitures des plaquettes des freins *(figure 5.12)*

Estimation de la vitesse du véhicule *(figure 5.13)*

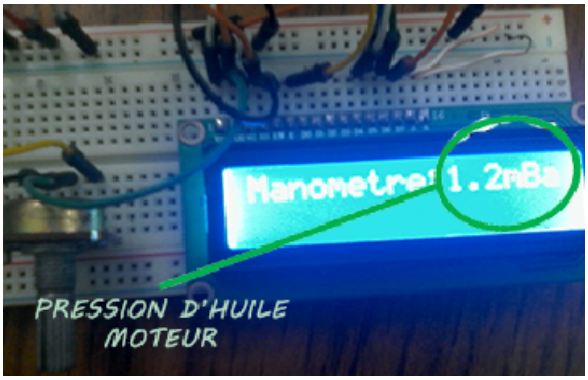


Fig. 5.10 Recueil d'information sur l'état du niveau d'huile moteur

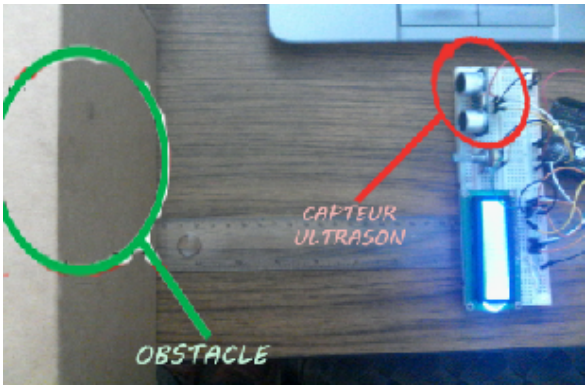


Fig. 5.11 Appréciation de la distance entre un obstacle et l'automobile

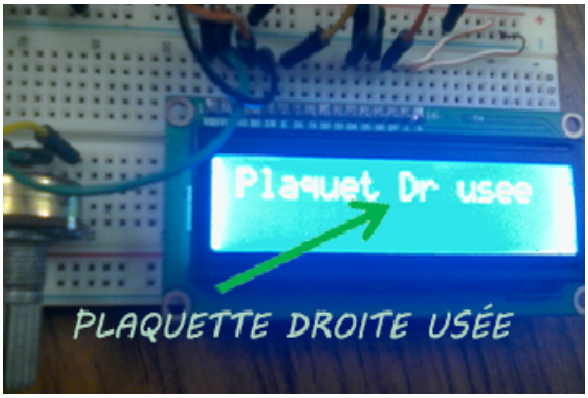


Fig. 5.12 Simulation avec bouton poussoir

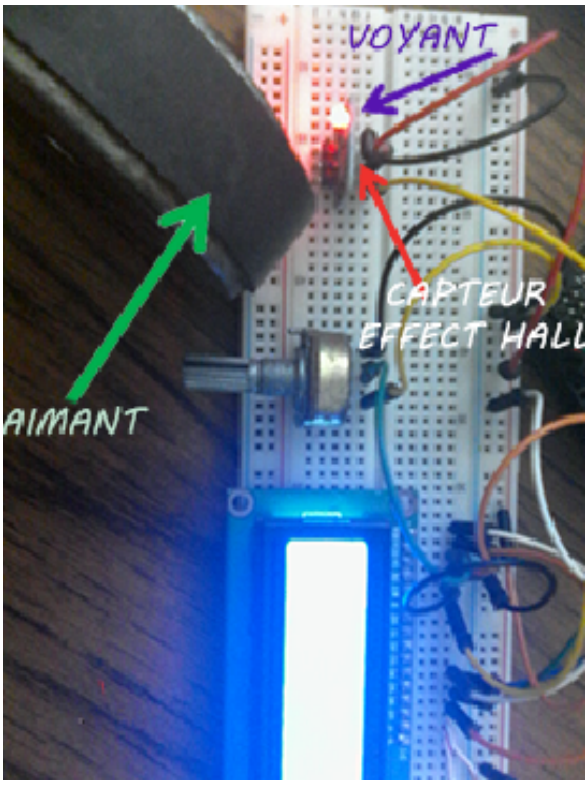


Fig. 5.13 Estimation de la vitesse du véhicule avec capteur à effet hall

Test d'intégration

La figure 5.14, présente la structure finale de notre montage de simulation matérielle, avec l'intégration de tout les capteurs:

5.3 Application Android

Le développement de notre application Android, dont la page principale est présentée à la figure 5.15, a été déployée sur un smartphone Alcatel OneTouch Pixi 4.5 équipé du système d'exploitation Android s'est avérée fonctionnelle et a ainsi pu être testée. Elle sera fonctionnelle sur les smartphones disposant des mêmes caractéristiques techniques (Android 4.0 ou supérieur).
Les tests effectués avec la maquette ont confirmé la difficulté de prendre visuellement compte des données renvoyées par les capteurs et ont montré l'importance d'une bonne ergonomie pour garantir un emploi facile du produit développé.
Cette première version de l'application se décompose en trois grandes fenêtres d'affichage :

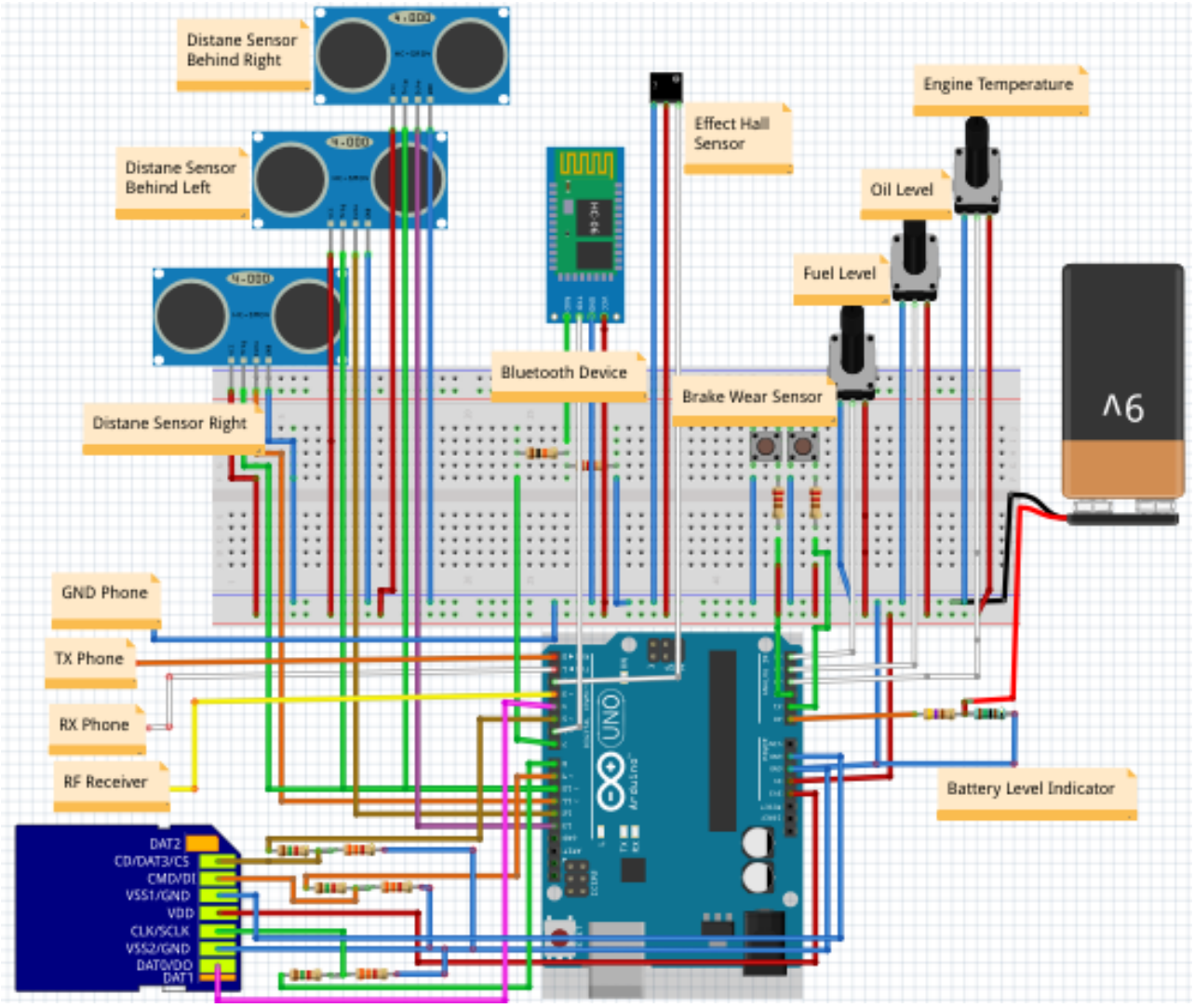


Fig. 5.14 Montage matériel du système

La première, celle principale nous présente les données récupérées par nos capteurs (niveau d'essence et huile, état des capteurs à ultrason, voltage de la batterie,...) (figure 5.15).

Fig. 5.15 Écran principal de l'application Android

La seconde interface utilise les données qui transitent dans le système pour réaliser une étude statistique des éléments étudiés par l'ensemble des capteurs (figure 5.16).

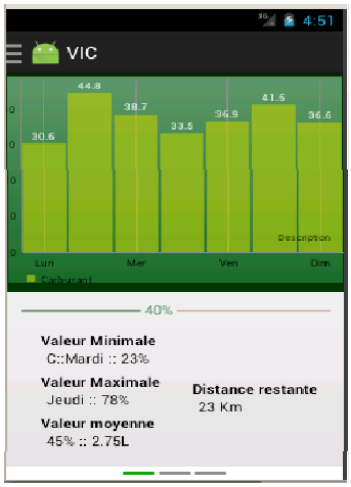


Fig. 5.16 Interface de statistique de l'application

La troisième fournit une interface pour la configuration des paramètres du système. La navigation se fait à l'aide des vues et d'onglets latéraux (cf figure 5.16) [21].

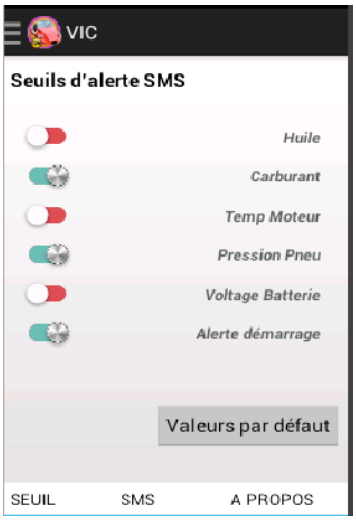


Fig. 5.17 Interface de paramètrage de l'application

La communication entre l'application Android et la carte Arduino, se fera par bluetooth ; afin de garantir une interactivité faible entre ces deux entités (nœuds), notre programme Android implémentera deux tâches principales :

- La première pour écouter tout les événements déclenchés sur l'IHM par l'utilisateur;
- La seconde garantira une communication bluetooth transparente avec la carte Arduino.

Conclusion générale

Nous recapitullons ici les résultats obtenus au cours de nos travaux durant le stage, ensuite nous présentons les perspectives relatives ces travaux.

Conclusion

Rappel du problème

Notre objectif était de fournir à un utilisateur un kit d'auto-diagnostique permettant de prendre en compte, un ensemble de pannes assez fréquentes en automobile. Cela suppose:

- Comprendre le fonctionnement des systèmes embarqués;
- Programmer des applications à forte contraintes temporelles;
- Déployer un ensemble de capteurs permettant de monitorer plusieurs paramètres vitaux d'un véhicule (moteur, carburant, freinage,...);
- . . .

Solutions proposées

Nous avons conçus et implémentés un kit d'auto-diagnostique. La réalisation de ce kit a été menée comme suit:

- Une analyse des pannes que le kit prendra en charge, à partir d'un ensemble de capteurs dont les données sont archiver à l'aide du SGBD SQLite;
- La modélisation a été réalisée avec AADL, "Cheddar" a stimulé le modèle obtenu et l'implémentation du modèle en "langage" arduino avec la contrainte temps-réel a été fait avec l'environnement ERIKA Enterprise;

- Le déploiement du système s'est fait sous les environnements Arduino et Android, mis en communication à partir d'une communication bluetooth;

Perspectives

Perspectives à court terme

A cours terme, les perspectives seront de :

- Réorganiser notre ensemble de capteurs en un réseau de capteurs normalisé, afin d'augmenter la vitesse du flux d'information transitant dans cet ensemble de capteurs;
- Optimiser le modèle de la base de données;
- Concevoir le kit matériel, en tenant compte des contraintes de son environnement (vibration, température,...).

Perspectives à long terme

A long terme, plusieurs perspectives s'ouvrent à partir des résultats de nos travaux:

- Obtenir un brevet du kit dévellopé;
- Déploiement la solution logicielle avec licence Open Source soigneusement étudiée.

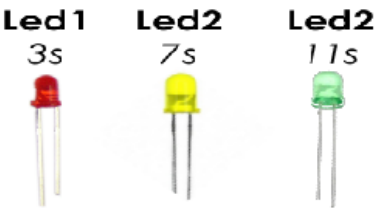
Annexe

Exemple:: Clignotement de 3 leds

L'exemple ci dessous illustre un problèmes de priorité entre différentes tâches.
Soit trois leds, nous désirons les faire clignoter à des fréquences différentes (figure 18) [22]:

- LED1 clignote chaque 3 secondes;
- LED2 clignote chaque 7 secondes;
- LED3 clignote chaque 11 secondes.

.




```
int led1 = 13;
int led2 = 14;
int led3 = 15;
int count = 0;

void loop() {
        if (count%3 == 0)
                digitalToggle(led1);

        if (count%7 == 0)
                digitalToggle(led2);

        if (count%11 == 0)
                digitalToggle(led3);

        if (count == 3*7*11)
                count = 0;
        count++;
        delay(1000);
}
```

Fig. 18 Exemple de clignotement de 3 leds

Avec le framework de programmation classique d'arduino, nous aurons les problèmes suivants ;

- Quelle led entre la une et la deux allumera le programme à la 21 secondes (multiple de 3 et de 7)?

- Comment le processeur réagira t'il si deux ou plusieurs tâches venaient à être prêtent simultanément?

Références

[1] Technique de diagnostic dans le domaine automobile.

[2] Malika BENAMMAR. *Une Approche Basée Architecture pour la Spécification Formelle des Systèmes Embarqués Thèse Doctorat en Sciences*. PhD thesis, 2011.

[3] Bosch. Capteurs pour véhicules automobile. Technical report, Bosch, 2009.

[4] Direction des systèmes de transport Environnement Canada. Systèmes de diagnostic de bord 2 (obd-2) et programmes d'inspection et d'entretien pour le contrôle des émissions des véhicules légers. April 2004.

[5] Christian Fotsing. Systèmes temps-réel embarqués. 2014.

[6] http://beru.univ-brest.fr/ singhoff/cheddar/. The cheddar project : a free real time scheduling analyzer. July 2015.

[7] http://fr.wikipedia.org/wiki/Affaire_Volkswagen. Affaire volkswagen. 2015.

[8] http://fr.wikipedia.org/wiki/On$_{B}oard_{D}iagnostics$. $Onboarddiagnostics - wikipdia.May2015$.

[9] http://fr.wikipedia.org/wiki/SQLite. Sqlite-wikipédia. June 2015.

[10] http://kitcar.bb-fr.com/t4975-comment-fonctionne-une-jauge-a essence. comment fonctionne une jauge a essence? October 2015.

[11] http://obd2outil.ivoire-blog.com/archive/2014/11/05/est-il-necessaire-d-avoir-une-voiture-hud-head-up-display 452431.html. Est-il nécessaire d'avoir une voiture hud (head-up display)? : obd2 outil. June 2015.

[12] http://www.aadl.info/aadl/currentsite/. Architecture analysis and design language. July 2015.

[13] http://www.autocarpro.in/features/bosch-introduces-unique-parking-aid 3498. Bosch introduces unique parking aid. October 2015.

[14] http://www.fiches-auto.fr/articles-auto/fonctionnement-d-une-auto/s-849-fonctionnement-du-radar-de recul.php. Fonctionnement du radar de recul - le radar de recul est un systèm. October 2015.

[15] http://www.outilsobdfacile.com/obd presentation.php. Présentation de l'obd et de la prise diag obd2 - outils obd facile. May 2015.

[16] ANDERS NORDIN JOANNA ERIKSSON, MIKAEL JOHANSSON. *Development of Android Software for Logging of Engine Data for Shell Eco Marathon*. PhD thesis, 2013.

[17] JUMO. Capteur de niveau et capteur de température jumo pour le diagnostic de véhicules. 2009.

[18] Frank Singhoff Jérôme Hugues. Développement de systèmes à l'aide d'aadl - ocarina/cheddar.

[19] Claude Lahache. Électronique auto. 2009.

[20] Philippe Pucheral. Nicolas Anciaux, Luc Bouganim. Sgbd embarqué dans une puce : retour d'expérience. techniques et sciences informatiques. 2008.

[21] Alban Wibaux-... Nicolas Brocheton, Kevin Bruget. *Système d'assistance à la navigation handivoile*. PhD thesis, August 2013.

[22] Pietro Lorefice Pasquale Buonocunto, Alessandro Biondi. Real-time multitasking in arduino.

[23] ... Pau Mart, Javier Tejedor. Proposal for an embedded control systems laboratory activity.

[24] Evidence Srl. Erika enterprise pre-built virtual machine. 2014.

[25] S.V. Syed Anwaarullah. Rtos based home automation system using android. 2013.

[26] Pierre Dissaux... Vincent Gaudel, Frank Singhoff. Composition of design patterns : from the modeling of rtos synchronization tools to schedulability analysis.

[27] www.obdii.com. Obd-ii on-board diagnostic system. July 2015.