ETR-2005 Dimensionnement temps réel d'un véhicule : étude de cas et perspectives

Jaime De Oliveira
Valeo Electronique & Système de Liaison
Division Electronique et Interconnexion
Europarc – 94042 Créteil Cedex - France
Jaime.de-oliveira@valeo.com

Abstract

Dans l'automobile, la problématique du dimensionnement temps réel de l'architecture fonctionnelle (électrique / électronique) d'un véhicule a évolué au même titre que les architectures électriques. La distribution des fonctions à travers des architectures multiplexées a permis de répondre à un besoin croissant de fonctionnalités de plus en plus nombreuses et complexes. Mais, ces innovations ne sont pas sans conséquence sur la conception de la dynamique et la validation des délais de « bout-en-bout ». L'objectif de cette publication est de présenter la vision de la Division Electronique et Interconnexion (DEI) de Valeo sur les fonctions de confort dans l'habitacle des véhicules. Cette vision est illustrée à l'aide d'une étude de cas utilisant une approche « pire cas » pour vérifier le respect des délais.

1. Introduction

Depuis sa création, l'automobile a toujours été proche des nouvelles technologies. Durant ces longues années, elle a su innover tout en intégrant des technologies venant d'autres secteurs. Aujourd'hui, l'électronique et le logiciel embarqué prennent une part croissante dans le monde de l'automobile et avec eux apparaissent de nouvelles problématiques qu'il faut résoudre.

Parmi elles, le dimensionnement temps réel de l'architecture électrique d'une voiture. Il s'agit d'identifier tous les éléments intervenant dans la réalisation des fonctions et d'en déterminer les impacts sur les temps de réactions.

Les fonctions dans l'automobile étant nombreuses et variées, nous aborderons uniquement la partie électronique et logicielle embarquée se trouvant dans l'habitacle des véhicules. Cette publication présente la vision de la Division Electronique et Interconnexion (DEI) de la branche Valeo Electronique et Système de Liaison (VESL) sur les différents aspects du dimensionnement Temps Réel dans cette partie de l'automobile.

2. Une architecture électrique évolutive

Un véhicule est traversé de part en part par des faisceaux de fils électrique afin de transmettre énergie et informations à l'ensemble des différents organes électriques composant le système.

L'augmentation croissante des fonctions dans l'automobile a accru considérablement le nombre de ces fils et par conséquent, augmenté le poids des véhicules ainsi que leur coût. Une mutation de ces liaisons vers des réseaux multiplexés permet de limiter ces contraintes tout en introduisant d'autres difficultés à surmonter, comme le dimensionnement temps réel et la réactivité.

2.1. Les Architectures Conventionnelles

Dans les architectures conventionnelles chaque fonction est prise en compte par un boîtier électronique.

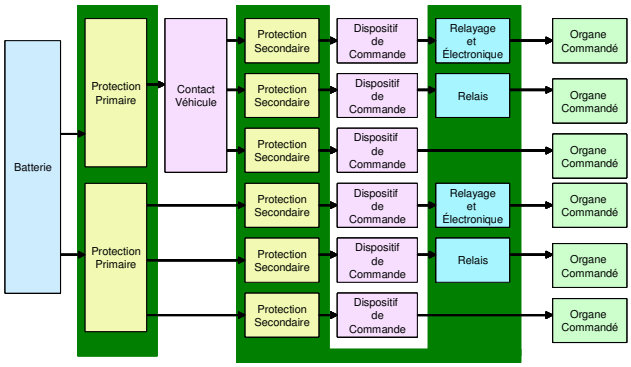

Figure 1. Architecture Conventionnelle.

Cette approche nécessite des composants de protection et d'alimentation de l'électronique, par boîtier, avec des broches de raccordement pour connecter les alimentations, les entrées et les sorties nécessaires à la prise en compte de la fonction.

Les contraintes de temps réel dans cette architecture sont internes à chaque calculateur : le logiciel embarqué dans les calculateurs est dédié à la fonction prise en charge.

Le choix d'une telle architecture tend à disparaître de nos voitures car très coûteuse et peu adaptée à l'augmentation du nombre de fonctions dans le véhicule.

2.2. Les Architectures Centralisées

Dans les architectures centralisées un seul calculateur électronique regroupe tout un ensemble de fonctions.

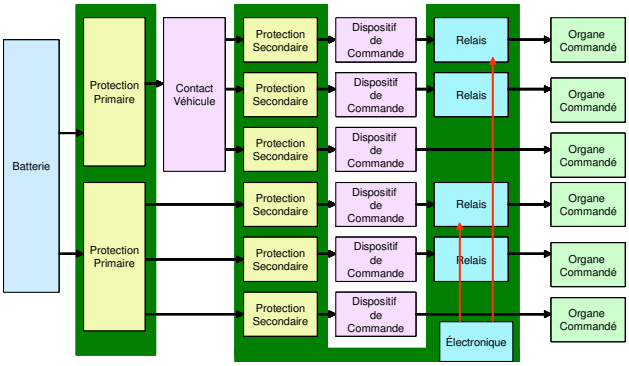

Figure 2. Architecture Centralisée.

Cette approche permet d'avoir un dispositif unique d'alimentation de l'électronique pour toutes ces fonctions, et, des composants de protection partagés entre elles. A cela, s'ajoute une optimisation du nombre de broches de raccordement pour connecter les alimentations, les entrées et les sorties nécessaires à la prise en compte de l'ensemble des fonctions.

Les contraintes de temps réel, dans cette architecture, apparaissent localement dans chaque calculateur au niveau du logiciel embarqué, mais ne sont toujours pas dimensionnantes. Des choix de microcontrôleurs plus puissants permettent de répondre en partie à ces contraintes.

La centralisation d'un ensemble de fonctions dans un boîtier électronique permet d'augmenter le nombre de fonctions dans les voitures.

2.3. Les Architectures Distribuées

Dans les architectures distribuées, une fonction peut être partagée entre plusieurs calculateurs électroniques reliés par des réseaux multiplexés.

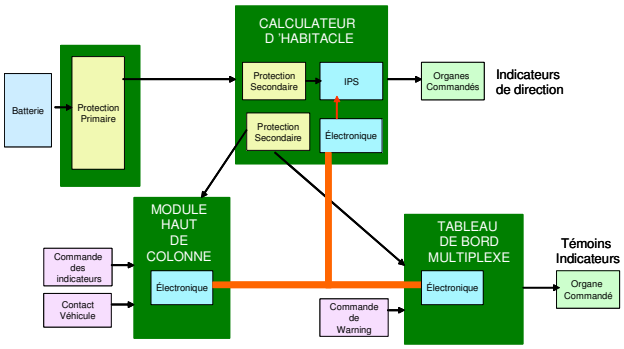

Figure 3. Architecture Distribuée.

L'introduction des réseaux permet de réduire la quantité de fils reliant les nombreux organes d'un véhicule. En contre partie, ceci impose de définir une messagerie pour ces réseaux et de revoir la conception des circuits de protection et des alimentations.

Les contraintes de temps réel dans ces architectures deviennent dimensionnantes dans la définition du système. Elles apparaissent avec la partie communication des réseaux et leurs modes de fonctionnement et se prolongent avec la partie distribuée entre plusieurs organes du logiciel embarqué pour la réalisation d'une fonction.

Aujourd'hui, les architectures sont un savant mélange de centralisation, pour l'optimisation des coûts, et de distribution, pour conserver une flexibilité optimale.

2.4. Les choix d'architectures réseaux dans l'automobile

L'architecture électrique dans l'automobile repose aujourd'hui sur les réseaux. La question est : un ou plusieurs réseaux ?

Un seul réseau, auquel tous les calculateurs seraient reliés et par lequel toutes les informations partagées passeraient, est une idée simple et séduisante. Mais, il n'existe pas de réseau parfait. Un réseau est fait de compromis entre performances et coût. Le nombre de nœuds, la longueur du médium et la bande passante sont limités.

Les besoins dans l'automobile sont hétérogènes : des informations évènementielles et avec réveil par le bus (fonctions de l'habitacle), des informations à cycle d'émission rapide (fonctions moteurs et châssis) et « l'Infotainment » à fort volume de données, ces deux dernières exigeant particulièrement une très grande bande passante.

Un système automobile est généralement ségrégué pour ne pas compromettre les fonctions vitales en cas de défaillance d'un calculateur de confort et/ou en cas de sectionnement d'un réseau dans une zone exposée.

Les réseaux sont ou ne sont pas de différents types (CAN, LIN,…) et communiquent entre eux au travers d'un calculateur communément appelé « Gateway »

(Passerelle). Cette Passerelle peut être centralisée, voir Figure 4a, ou bien décentralisée, voir Figure 4b.

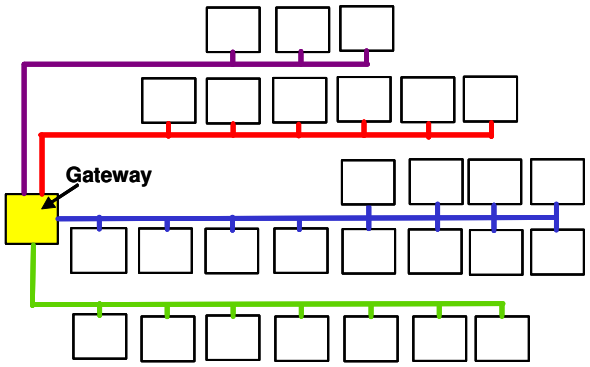

Figure 4a. Gateway Centralisée.

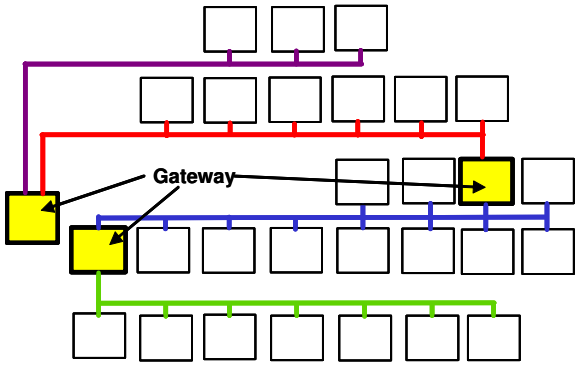

Figure 4b. Gateway Distribuée

2.5. L'architecture de l'étude de cas

Cette étude de cas va servir de fil conducteur tout au long de ce document afin d'illuster les sujets techniques abordés.

Il s'agit de mettre en œuvre une fonction clignotant simplifiée (mais nous verrons ultérieurement qu'elle n'est pas si simple : les fonctions feux de détresse, acquittement visuel de verrouillage/déverrouillage, etc… interagissant avec elle), dans une architecture distribuée, un peu académique pour les besoins de cette publication.

L'architecture se compose de trois calculateurs reliés par des réseaux multiplexés :

- HDC (Haut De Colonne) qui possède les contacteurs de demande de clignotement droite et gauche.
- BC (Body Contrôleur) qui analyse la demande et selon une stratégie définie, envoie la bonne commande de clignotement.
- BFD (Boîtier Feux Droit) qui réalise la commande de puissance des feux pour le coté droit sur demande du BC.

Les exigences temporelles sont : une latence de 300ms de la détection à la réalisation de la commande et une fréquence de 1,2 Hertz de clignotement.

L'étude de cas est orientée sur l'application de la démarche de dimensionnement plutôt que sur la validation formelle des calculs.

3. Le Multiplexage dans l'automobile

3.1. Les généralités d'un réseau

Un réseau élémentaire se constitue d'au moins deux noeuds (un émetteur et un récepteur) et d'un médium de transmission des informations.

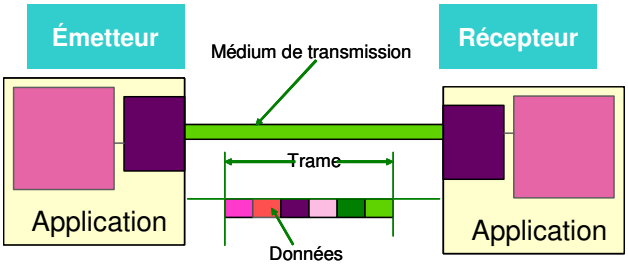

Figure 5. Généralités d'un Réseau.

Les données à transmettre sont encapsulées dans des trames répondant à un protocole de communication. Chaque trame se compose de plusieurs champs contenant différents niveaux d'informations (Début de Trame, Identifiant, Données, Contrôle,…). Un mécanisme d'arbitrage définit comment gèrer les conflits entre plusieurs nœuds voulant émettre simultanément.

Une caractéristique importante d'un réseau est sa vitesse de transmission. Cette vitesse correspond au nombre de bits par secondes (ou Bauds) que le réseau est capable de transmettre avec un médium de transmission donné. Les besoins des systèmes électroniques embarqués varient actuellement de 10Kb/s à 25Mb/s.

La vitesse de transmission est le premier critère pour sélectionner un réseau. Une classification primaire des besoins en multiplexage pour l'automobile répartit les applications en différentes classes suivant ce critère.

Classification	Vitesse	Exemple
Classe A	< 10 Kb/s	Ouverture coffre
Classe B	10-125 Kb/s	Tableau de bord
Classe C	125 Kb/s-1Mb/s	ABS, Motorisation
Classe D	> 1Mb/s	Multimédia, X-by-Wire

3.2. Les types de réseaux

Dans le cadre de cette publication dédiée à l'automobile, les types de réseaux abordés sont le Maitre-Esclaves et le Multi-Maitres avec des topologies physiques variables selon les réseaux (CAN, LIN, Flexray,…).

La liaison point à point mono ou bidirectionnelle étant un type de réseau peu utilisé dans l'automobile, elle ne sera pas traitée dans ce document.

3.2.1. Maitre-Esclaves

Dans une architecture Maître-Esclave(s) une seule station, appelée Maître, peut initier la communication sur le bus. Le maître (station émettrice) peut transmettre un message (trame) simultanément à plusieurs stations réceptrices. Dans ce cas, on parle de message en diffusion, toutes les stations réceptrices (esclaves) pouvant exploiter simultanément le message reçu.

Communément, le maître s'adresse à une des stations réceptrices. Celle-ci reconnaîtra le message transmis à l'aide d'un mécanisme de filtrage de son champ « Identifiant ».

Dans l'automobile, le réseau de type Maître-Esclaves le plus répandu est le LIN. Ses stations esclaves disposent également d'émetteur pour effectuer un acquittement dans la trame mais aussi pour pouvoir répondre dans la trame, en remplissant le champ des données utiles.

3.2.2. Multi-Maîtres

Dans une architecture Multi-Maîtres, toutes les stations ont accès au réseau et peuvent initier la communication sur le bus : elles disposent toutes d'un récepteur et d'un émetteur.

Dans une architecture de ce type, chaque calculateur possède une liste de trames correspondant aux différents messages qu'il est susceptible d'émettre. De la même manière, chaque calculateur dispose de filtres lui permettant de sélectionner uniquement les messages reçus qui le concernent.

Ce multiplexage permet le partage d'un support de communication entre plusieurs usagers et cela pour réaliser plusieurs fonctions. En contre partie, ces architectures nécessitent la mise en place d'un système de gestion de la priorité des accès au bus par type de trame : l'arbitrage.

Dans l'automobile, le réseau de type Multi-Maîtres le plus répandu est le CAN. Son système d'accès est de type CSMA/CR (Carrier Sense Multiple Access with Collision Resolution) où les collisions sont détectées et arbitrées selon un système de priorité basé sur un niveau électrique dominant. Un autre réseau fait actuellement une percée dans l'automobile avec un système d'accès de type TDMA (Time Division Multiple Access). Il s'agit du Flexray, où chacun dispose d'un créneau d'émission ddédié, alloué selon un timing (une séquence) préétabli et figé.

3.3. Le CAN (Controller Area Network)

Le thème de cette publication n'étant pas les réseaux CAN, des connaissances sont pré-requises. Le livre de D. Paret [1] est une bonne aide.

Le protocole CAN a été développé par la société Robert Bosch dans le courant des années 80. Ce protocole ainsi que les paramètres électriques du medium de transmission sont fixés par des normes ISO 11898 (High Speed) et ISO 11519-2 (Low Speed). Le protocole CAN, actuellement le plus répandu, est le 2.0. Il se divise en deux sous-catégories qui diffèrent uniquement au niveau de la longueur de l'identificateur des trames.

Les trames transmises par un nœud sur le bus contiennent un identificateur (ID) grâce auquel chaque nœud recevant les trames peut déterminer si celles-ci le concernent. Si c'est le cas, il les prend en compte, sinon, il les ignore. Cet ID est unique et indique aussi la priorité des trames.

Comme indiqué précédemment, le CAN est un réseau de type Multi-Maîtres utilisant le système d'accès au bus de type CSMA/CR. L'arbitrage du bus se fait au niveau des bits en utilisant l'ID des trames. Chaque nœud peut présenter sur le bus un bit appelé dominant (0 logique) et un bit appelé récessif (1 logique). Tout conflit de bus est résolu par un mécanisme de « AND », c'est-à-dire qu'un état dominant écrase un état récessif ; chaque station émettrice vérifie l'état de la ligne comparativement à ce qu'elle vient d'émettre et s'interrompant en cas d'émission plus prioritaire par une autre station.

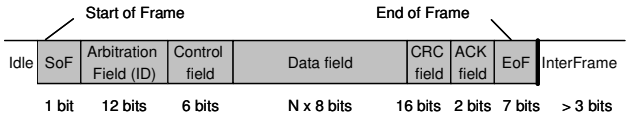

Figure 6. Data Frame.

Quatre types de messages peuvent circuler sur un bus CAN :

- Les Data Frames (Trames de Données) sont utilisés pour transporter les données sur le bus. Cette trame est la plus utilisée dans les réseaux automobile.
- Les Remote Frames (Trames de Requêtes) sont utilisés par un nœud pour demander la transmission d'une Data Frame de même ID par d'autres nœuds. Cette trame est peu ou pas utilisée (selon les constructeurs) dans les réseaux automobiles.
- Les Error Frames (Trames d'Erreurs) sont transmises par un nœud ayant détecté une erreur.
- Les Overload Frames (Trames de Surcharge) sont utilisées pour produire un délai entre deux Data ou Remote Frames successives. Cette trame est peu ou pas utilisée (selon les constructeurs) dans les réseaux automobiles.

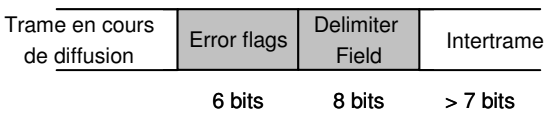

Figure 7. Error Frame.

Le flot de bits des trames du bus CAN est codé à l'aide de la méthode NRZ (Non Return to Zero) auquel vient s'ajouter un procédé particulier : le bit stuffing. Ce procédé consiste à introduire, au niveau de la transmission, après 5 bits de valeur identique (dominants ou récessifs), un bit supplémentaire de valeur opposée. Ceci allonge le temps de transmission d'un message mais participe activement à sécuriser le message lors de son transport sur le bus.

$$C_i = \left\lfloor \frac{34 + 8 \times N_i}{5} \right\rfloor + 47 + 8 \times N_i$$

Soit N_i le nombre de données utiles à transmettre dans une « Data Frame », C_i est la durée d'émission de la trame au format CAN 2.0A en bits.

$$\varepsilon(m, C_i) = m(23 + C_i)$$

Soit m le nombre de retransmission d'une trame i de durée C_i, ε est l'influence des « Error Frame » (23 étant la taille maximum de ces trames d'erreurs) sur la transmission de la trame.

La charge d'un bus CAN va dépendre de sa messagerie et des types de transmission autorisés ; c'est-à-dire, de la liste complète des messages pouvant circuler sur le bus et de leur fréquence de transmission. L'élaboration de cette messagerie et la planification des transmissions font parti du dimensionnement temps réel.

3.4. Le LIN (Local Interconnect Network)

Le thème de cette publication n'étant pas les réseaux LIN, des connaissances sont pré-requises (voir le site internet [2] et la publication [3]).

Le protocole LIN a été initié au travers d'un groupe de travail en 1998, et les premières spécifications ont été publiées en 1999. Ce Consortium se compose essentiellement de constructeurs (Volvo, Audi, BMW, VW et Daimler Chrysler) mais aussi du fabricant de semi-conducteurs Motorola (devenu Freescale) et du fournisseur d'outils Volcano Technologies.

Le LIN est un standard ouvert de communication multiplexé à bas coût pour l'automobile avec comme objectif de remplacer le protocole CAN lorsque les performances de celui-ci ne sont pas requises. Comme indiqué précédemment, le LIN est un réseau de type Maître-Esclaves. Le maître envoie un en-tête de message qui est reçu par tous les nœuds du réseau. Chaque nœud analyse l'identificateur contenu dans cet en-tête et passe alors soit en émission, soit en réception de données en fonction de l'identificateur, soit en attente.

Une particularité du LIN est son mécanisme de synchronisation permettant aux nœuds esclaves de se dispenser d'une base de temps stable intégrée pour des raisons de coût.

Une trame LIN est basé sur un format fixe avec un choix déterminé de longueur et se divise comme suit :

- Le champ « Header », envoyé par le maître se compose de trois sous champs : le champ « synchronisation break » qui indique le début de la trame, le champ de synchronisation qui permet aux esclaves de s'ajuster sur la base de temps exacte du maître, et l'identificateur qui définit la nature du message et le nombre de données du champ de réponse.
- Le champ « Response », envoyé soit par le maître, soit par l'un des nœuds esclaves, se compose de données pouvant contenir deux, quatre ou huit octets et d'un « Checksum ».

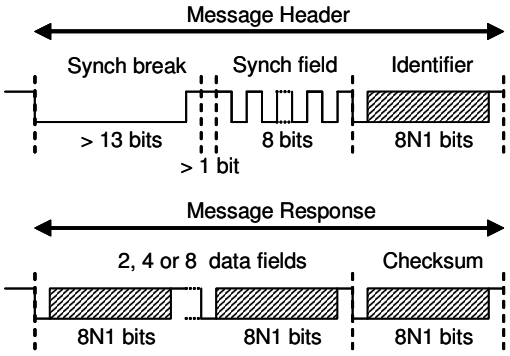

Figure 8. Structure d'une trame LIN.

Le flot de bits des champs d'octets des trames du bus LIN est codé à l'aide de la méthode 8N1 identique au format de communication série UART classique. Chaque champ octet a une longueur de 10 bits commençant par un « Start bit » de type dominant, suivi de 8 bits de données avec le LSB en premier et se terminant avec un « Stop bit » récessif.

Tous les champs peuvent être transmis sans délai entre eux. Pour des commodités de réalisation la spécification du LIN ne définit pas de délai maximum entre les différents champs mais une allocation de temps maximum pour la transmission du message.

Soit N_{field} le nombre de données du champ réponse hors Checksum et T_{bit} le temps de transmission d'un bit en secondes, $T_{msg,min}$ est le temps minimum de transmission de la trame et $T_{msg,max}$ est le temps

$$T_{msg,\min} = (44 + 10 \times N_{field}) \times T_{bit}$$

$$T_{msg,\max} = 140\% \times T_{msg,\min}$$

maximum autorisé.

La charge d'un bus LIN va dépendre de la table de séquencement des messages ; c'est-à-dire, de la liste complète des messages et du planning de transmission du nœud maître. L'élaboration de cette table de séquencement des transmissions fait parti du dimensionnement temps réel.

3.5. Les réseaux dans le dimensionnement temps réel

3.5.1. Une approche holistique pour le délai de « bout-en-bout »

Dans les architectures conventionnelles, le dimensionnement du temps de réponse maximum d'une fonction dans un calculateur était déjà existant. Mais ce temps n'était pas influencé par les fonctions se trouvant sur d'autres calculateurs. Il dépendait essentiellement de l'architecture temps réel du logiciel embarqué dans le calculateur.

Dans les architectures actuelles, le dimensionnement du temps de réponse maximum d'une fonction distribuée doit tenir compte des effets que produisent d'autres fonctions du système. Ces effets sont, d'une part, des délais dans les traitements logiciels (détaillé plus loin dans ce document) et, d'autre part, des délais de communication entre calculateurs. On parle alors de délai de « bout en bout » d'une fonction.

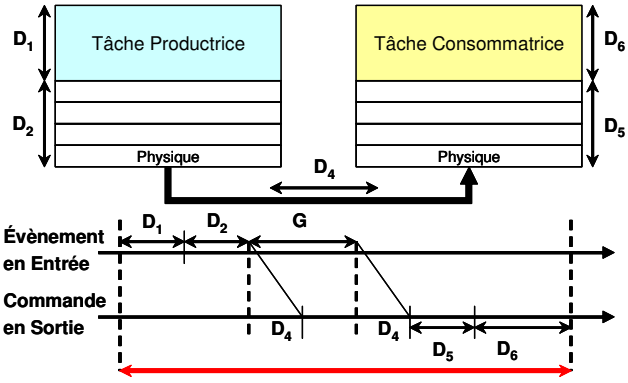

Figure 9. Le Délai de « bout-en-bout ».

L'architecture automobile est un système réactif. Elle est en interaction continue avec l'environnement. Les fonctions composant ce système sont toutes liées à l'environnement par des évènements asynchrones. Ces évènements sont nombreux et leur nombre de combinaisons d'apparition est exponentiel. Cette situation, dans un environnement de développement fortement contraint comme l'automobile, rend impossible de valider exhaustivement tous les cas de configurations possibles dans des délais compatibles. La difficulté réside dans les solutions à mettre en œuvre pour garantir un niveau maximal de sécurité, de fiabilité et de disponibilité des fonctions.

Dans ce cas, notre approche pour vérifier et garantir le dimensionnement des échéances temporelles est holistique. Cette approche consiste à considérer les pires conditions d'activation des évènements conduisant aux temps de réponse maximum. Cela inclus de prendre en compte la gigue possible introduite par tous les nœuds participant à l'obtention de ces échéances temporelles. La figure suivante illustre le principe de la gigue :

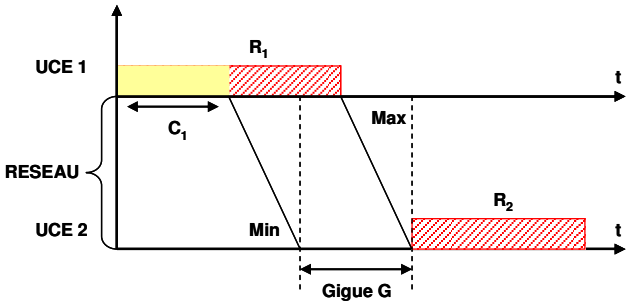

Figure 10. Le principe de la Gigue.

Soit C_1 le temps minimum d'exécution de la tâche 1 de l'UCE 1, R_x le pire temps de réponse de la tâche x, ***Min*** et ***Max*** les délais minimum et maximum sur le réseau, la gigue ***G*** se calcule comme suit :

$$G = R_1 - C_1 + Max - Min$$

Si ***R*** est le délai de « bout-en-bout », dans une démarche holistique, il se calcule comme suit :

$$R = R_1 + Max + R_2 - G$$

Ce calcul se généralise à N nœuds. Le calcul du pire temps de réponse sur chaque nœud dépend de la politique d'ordonnancement choisi au niveau de l'Operating System (OS) du nœud tandis que le calcul des délais de transmission maximum dépend du réseau et de la messagerie.

3.5.2. La messagerie et le délai de transmission pire cas

Le délai de transmission de chaque trame va dépendre du protocole de communication du réseau mais aussi de sa messagerie. Dans le contexte de l'automobile, il existe trois types de trames pour former une messagerie :

- Les trames périodiques émises sur le réseau tous les P_i périodes.
- Les trames évènementielles dont l'émission sur le réseau correspondent à l'arrivée d'un évènement.
- Les trames mixtes avec une émission à la fois périodique et évènementielle.

Dans le cas d'un bus LIN, la messagerie est composée exclusivement de trames périodiques qui forment la table de séquencement. Toutes les émissions sont initiées par un seul et même nœud : l'UCE maître. Le délai de transmission maximum pour une trame i de la messagerie est :

$$MAX_{i,LIN} = C_{i,\max} \cdot T_{bit}$$

$$C_{i,\max} = 140\% \times (44 + 10 \times N_{field})$$

La charge réseau induite par une messagerie composée de M trames est :

$$\rho_{LIN} = \sum_{i=0}^{M} \frac{C_{i,\max}}{P_i}$$

Ces calculs ne tiennent pas compte d'éventuelles erreurs de transmission. Les délais, qu'elles engendrent,

restent négligeables par rapport aux marges de temps qui sont prises.

Dans le cas d'un bus CAN, la messagerie se compose de tous les types de trames et chaque trame est assignée à l'un des nœuds du réseau. Chaque nœud peut initier l'émission d'une de ses trames à tout instant.

Si ρ_k est la charge réseau induite par la messagerie du noeud k du réseau, la charge réseau d'un bus CAN sur lequel est connecté N noeuds est :

$$\rho_{CAN} = \sum_{k=1}^{N} \rho_k$$

$$avec \quad \rho_k = \sum_{i=1}^{P} \frac{C_i}{P_i} + \sum_{j=1}^{E} n_j \cdot \frac{C_j}{W_j} + \varepsilon_k(FER)$$

Où, ***P*** est le nombre de trames périodiques émises par le nœud k, ***E*** le nombre de trames évènementielles émises par le nœud k et n_j le nombre d'évènements sur une fenêtre temporel W_j.

$\varepsilon_k(FER)$, Frame Error Rate, représente la surcharge réseau générée par des erreurs de transmission du noeud k. Par hypothèse, ce terme est négligé car majoré par les marges prises.

Le calcul du délai de transmission maximum d'une trame, dans le pire cas, tient compte des trames déjà dans la pile d'émission de l'UCE et de toutes les trames plus prioritaires envoyées par les autres UCE du réseau :

$$MAX_{i,CAN} = \max_{t \in S}(\overline{W}_{i,t} + C_i - t) \cdot T_{bit}$$

$$MIN_{i,CAN} = C_i \cdot T_{bit}$$

$$avec \quad S = \bigcup_{j \in CAN(i)} S_j, S_j = \{kT_j, k \in \mathbb{N}\} \cap [0; L[$$

CAN(i) est le contrôleur CAN (ordonancement local de type FIFO) auquel appartient la trame. ***L*** est déterminé en tenant compte de l'ensemble des tâches soumises sur le réseau CAN.

$$\overline{W}_{i,t} = \sum_{\substack{j \in CAN(i) \\ j \neq i}} \left(1 + \left\lfloor \frac{t}{P_j} \right\rfloor\right) \cdot C_j + \left\lfloor \frac{t}{P_i} \right\rfloor \cdot C_i +$$

$$\sum_{\substack{j \in hp(i) \\ j \notin CAN(i)}} \left(1 + \left\lfloor \frac{\overline{W}_{i,t}}{P_j} \right\rfloor\right) \cdot C_j + \max_{\substack{k \in hp(i) \\ k \notin CAN(i)}} (C_k - 1)$$

Pour plus de précisions sur ces résultats mathématiques, qui pour diverses raisons peuvent être incomplets, veuillez vous référer aux documents [4] et [5].

3.6. Le multiplexage de l'étude de cas

Application à l'étude de cas de la fonction clignotant pour illustrer les informations sur le multiplexage. L'architecture de l'étude de cas se compose de trois UCE avec des relations fonctionnelles différentes.

Le HDC est relié au BC afin de lui transmettre les informations en entrée de la fonction, mais aussi celles utiles à d'autres fonctions comme l'essuyage. Ces deux UCE sont reliés par réseau à d'autres UCE du véhicule.

Le BC est relié au BFD afin de lui transmettre les ordres de commandes des sorties de la fonction mais peut être aussi ceux de la fonction éclairage. Le BFD est un nœud esclave, il est uniquement relié au BC via un bus multiplexé.

Dans cette situation, l'architecture réseau choisie est la suivante : une liaison multi maîtres CAN à 125 Kbauds entre le HDC et le BC, et une liaison maître esclaves LIN à 20 Kbauds pour la liaison BC et BFD.

Les données à échanger entre les différents UCE sont contenues, d'une part, dans une seule trame LIN et, d'autre part, dans une seule trame CAN : d'identificateur « 0x525 ». L'approche holistique entraîne les hypothèses suivantes : des trames avec un maximum de données à transmettre et une trame CAN périodique avec un identifiant de faible priorité.

L'application numérique pour le LIN propose deux périodes, l'une maximale et l'autre plus probable :

$$T_{bit} = 50\mu s; N_{field} = 8; MAX_{i,LIN} = 8{,}680ms$$

$$P_{i,\max} = 10ms; \rho_{i,\max} = 0{,}868$$

$$P_{i,hyp} = 50ms; \rho_{i,hyp} = 0{,}1736$$

L'objectif de ce document n'étant pas de valider les différentes équations présentées, on considérera que le réseau CAN se compose uniquement de ces deux nœuds avec une messagerie réduite, définie ci-dessous :

ECU	Frame ID	Cycle (ms)	Longueur	Ci (ms)
HDC	0x200	500	6	0,888
HDC	0x525	100	8	1,04
HDC	0x7F2	250	8	1,04
BC	0x0C2	40	2	0,584
BC	0x1F8	50	1	0,504
BC	0x220	100	6	0,888

L'application numérique pour le CAN se base sur le pire cas, à savoir toutes les trames émises à l'instant t_0=0.

$$\rho_{CAN} = 0{,}0499 \quad et \; MAX_{0x525,CAN} = 3{,}904ms$$

$$MIN_{0x525,CAN} = 1{,}04ms$$

4. Les calculateurs électroniques dans l'automobile

Ces dernières années ont vu les voitures s'équiper de nouvelles fonctions de sécurité (ABS, Airbags,…) et de confort (allumage des feux automatiques, lèves vitres avec fonction anti-pincement,…). Cette augmentation impacte directement le système, d'une part, sur son

architecture électrique (réseaux multiplexés) et, d'autre part, sur le nombre des calculateurs électronique et le poids du logiciel embarqué.

L'automobile étant un secteur très concurrentiel, le coût est toujours un élément fort du choix des technologies. Cependant, il ne faut pas oublier le besoin de composants répondant aux exigences d'industrialisation et une conception système orienté Sûreté de Fonctionnement. Ces différents paramètres ne sont pas sans conséquences sur le dimensionnement temps réel des fonctions.

4.1. L'évolution des calculateurs vers la diversité

Dans le cadre d'architectures conventionnelles, les calculateurs regroupent les informations en provenance des capteurs, l'ensemble du logiciel lié aux fonctions et la commande des actionneurs. Le logiciel embarqué peut parfois être contenu dans un microcontrôleur 8 bits avec des tailles mémoires allant de quelques centaines à quelques kilos octets.

Dans le domaine de l'habitacle, l'architecture logicielle est plutôt monolithique : les couches hautes, à savoir applicatives, et les couches basses, dédiées au pilotage du matériel, sont confondues. La dynamique du logiciel se résume souvent à un séquencement des processus dans une boucle de temps périodique. Les mécanismes d'interruptions sont peu, voir pas, utilisé.

Ces solutions répondent aux besoins temps réel de ces architectures. Le dimensionnement de la boucle de temps fait en sorte que les échéances de temps les plus dures sont respectées. L'introduction d'un réseau dans ce type d'architecture n'a pas bouleversé le logiciel au début mais il a demandé plus de ressources et de puissance de calcul aux microcontrôleurs.

Avec les architectures centralisées et l'utilisation des réseaux, l'importance de la puissance de calcul (CPU) et de la taille mémoire a considérablement crû. Les calculateurs regroupent plusieurs fonctions pouvant utiliser des informations en provenance de et/ou en partance vers d'autres calculateurs du système via les réseaux. Deux tendances voient le jour chez les constructeurs automobiles :

- une augmentation du nombre de calculateurs équipés de microcontrôleurs légèrement plus puissants que la précédente génération et d'un coût raisonnable,
- un nombre de calculateurs sensiblement identique mais équipés de microcontrôleurs nettement plus puissants que les précédents avec un coût plus élevé.

Cette divergence crée une diversité des solutions et entraîne l'élargissement des offres de microcontrôleurs dédiés au monde de l'automobile. Les calculateurs plus gourmands en ressources utilisent des microcontrôleurs 16 ou 32 bits avec des tailles mémoires d'une à plusieurs centaines de kilos octets et munis de nombreux périphériques.

Dans le domaine de l'habitacle, l'architecture logicielle évolue vers une structure en couches d'abstraction : les couches hautes, à savoir applicatives, deviennent indépendantes du matériel et les couches basses dédiées au pilotage du matériel fournissent des interfaces d'utilisation. Cette mutation est aussi due aux contraintes de développement, d'intégration avec le matériel et de test du logiciel qui augmentent la taille du code et deviennent plus complexes à mettre en œuvre. La dynamique du logiciel s'adapte en utilisant les interruptions et en mettant en œuvre des mécanismes plus élaborés de séquencement des processus, voire en introduisant des noyaux temps réel.

La conception globale du logiciel doit évoluer pour répondre aux besoins temps réel de ces architectures. Le dimensionnement des séquences, des tâches et des interruptions est fait de manière à garantir les échéances de temps.

La distribution des fonctions entre plusieurs calculateurs n'a pas bouleversé l'architecture du logiciel au début mais a introduit plus de contrôle des flux de données et, par conséquent, a demandé plus de ressources et de puissance de calcul aux microcontrôleurs.

Actuellement, les architectures sont dites distribuées ; c'est-à-dire que les fonctions sont partagées entre plusieurs calculateurs pouvant se trouver au cœur de plusieurs réseaux interconnectés. La distribution d'une fonction induit au moins les deux conséquences suivantes sur le système :

- l'augmentation des contraintes de Sûreté de Fonctionnement dûe à la mixité des fonctions dites sécuritaire avec des fonctions dites de confort sur un même calculateur,
- le dimensionnement temps réel des fonctions qui ne se cantonne plus à un calculateur mais à l'ensemble des calculateurs participant à la réalisation des fonctions.

Comme dans les évolutions précédentes, les calculateurs deviennent plus gourmands en ressources et tendent à utiliser des microcontrôleurs de 16/32 bits avec des tailles mémoires allant de la centaine de kilos octets au méga octet.

Cette approche du système a en partie pour objectif d'augmenter la réutilisation du logiciel afin de le fiabiliser et, par là même, de réduire les coûts et délais de développement. Pour cela, l'ensemble des acteurs de l'automobile mondial est regroupé au sein du Consortium AUTOSAR pour élaborer un standard dans le domaine du logiciel embarqué pour l'automobile.

4.2. Les architectures statiques du logiciel

Le choix de l'architecture statique du logiciel embarqué va avoir sa part d'influence dans le dimensionnement temps réel. L'organisation statique du logiciel, si l'on reste sur la même cible, va souvent à l'encontre des performances temps réel (par l'ajout de

couches standardisées, mises au format ou autres). Mais, cette structuration est primordiale pour tenir les délais de développement, augmenter la portabilité et la testabilité du logiciel et, surtout, améliorer sa qualité et sa fiabilité.

La première approche du logiciel, abordé ici succinctement, est de type monolithique. Elle se caractérise par l'absence de structure permettant l'abstraction des différentes parties du logiciel. En d'autres termes, elle permet l'accès direct au matériel (par exemple les registres des périphériques du microcontrôleur) à partir de toutes les fonctions, même de très haut niveau. Ce type de structure est très difficile à maintenir et à faire évoluer. La portabilité d'une fonctionnalité d'un calculateur à un autre est compromise de même que l'utilisation d'un processus d'intégration itératif. En contre partie, cette approche permet d'avoir un code réactif et concis.

Bien que cette approche ait ses avantages, ses inconvénients deviennent rédhibitoires dans un environnement de développement logiciel de plus en plus contraint comme peut l'être celui de l'automobile. Les délais de développement entre deux générations de calculateurs se réduisent de plus en plus, ce qui implique d'améliorer le niveau de réutilisation d'une ou de plusieurs parties du logiciel. Les spécifications peuvent évoluer tout au long d'un projet ce qui demande d'améliorer la maintenabilité et, par la même, l'évolutivité du logiciel.

La structuration d'un logiciel va répondre à des besoins et des objectifs pouvant diverger d'un domaine à un autre. L'architecture structurée, présentée dans ce document, est la réponse spécifique de Valeo (DEI) pour les calculateurs d'habitacle résultant d'une approche produit schématisée ci-dessous.

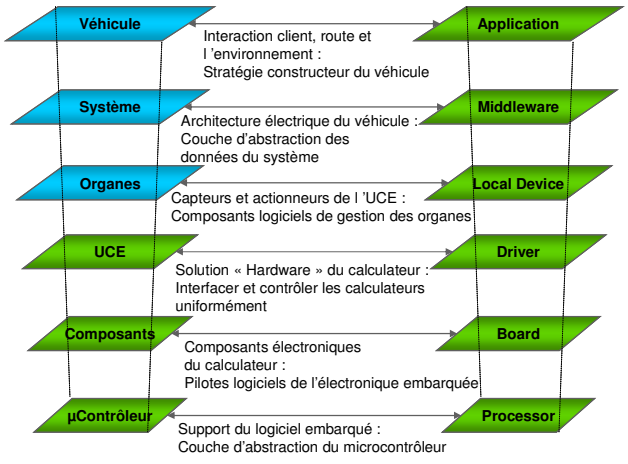

Figure 11. Une approche produit de l'architecture statique du logiciel.

Cette démarche résulte d'une découpe du logiciel en plusieurs couches. Chaque couche fait l'abstraction des éléments du produit qu'elle englobe. L'ensemble des fonctionnalités de ces couches est réparti en modules logiciels avec pour chacun une API (Application Program Interface) : une liste d'interfaces avec des comportements et un flux de données. L'impact d'une telle orientation se fait sentir de suite, d'une part, sur la charge CPU nécessaire à l'exécution des fonctions et, d'autre part, sur la taille du code, et donc de la mémoire nécessaire par rapport à une solution à architecture monolithique.

Chaque module logiciel d'une couche est conçu pour abstraire un ensemble de fonctionnalités et non pas pour répondre à une conception spécifique du logiciel. Le module pourra donc être utilisé dans plusieurs calculateurs avec des solutions électroniques différentes. Cela renforce la nécessité pour chaque module d'intégrer des aspects nouveaux comme : la vérification de la cohérence des paramètres en entrée de ses fonctions, mais aussi, le contrôle de sa bonne utilisation (exemple : vérifier que le module est correctement initialisé lorsqu'on lui demande d'émettre un message sur le bus). La figure 12 montre que, du point de vue utilisateur, le module « DIO » (Digital Input Output) est unique et réalise l'abstraction de deux conceptions. Pour cela, tous les modules sous « DIO » doivent vérifier qu'ils sont utilisés correctement selon leur spécification.

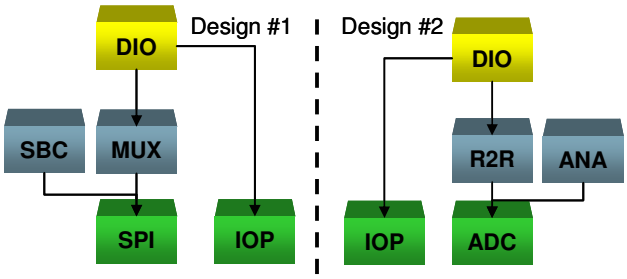

Figure 12. Exemples d'abstractions.

Bien que cette approche ait des impacts sur le choix des microcontrôleurs et sur le dimensionnement temps réel des fonctionnalités du calculateur, elle a l'avantage d'améliorer le niveau de réutilisation d'une ou de plusieurs parties du logiciel et ainsi de réduire les délais de développement entre deux générations de calculateurs, augmentant aussi le niveau global de qualité du logiciel grâce à la standardisation. Elle permet de répondre aux besoins de maintenabilité et d'évolutivité du logiciel.

Basé sur cette approche d'abstraction du matériel, et d'autres similaires, l'architecture évolue pour répondre à un besoin plus général d'abstraction du système. Il s'agit de permettre au logiciel d'être distribué à travers plusieurs calculateurs de l'ensemble du véhicule afin de répondre mieux à la complexité croissante des architectures électrique et électronique de l'automobile.

C'est pour définir une telle architecture que les constructeurs d'automobiles et les fournisseurs se sont regroupés au sein du Consortium AUTOSAR

(AUTomotive Open System Architecture). Les objectifs techniques principaux d'AUTOSAR sont résumés sur la partie gauche de la figure 13.

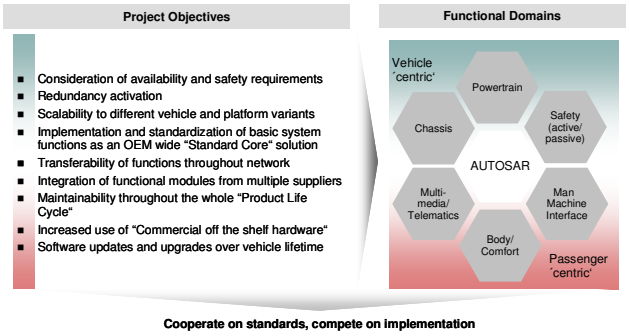

Figure 13. Objectifs du projet AUTOSAR

Cette architecture ouverte repose sur les concepts de SW-C (SoftWare Component) qui encapsule une fonction logicielle, et du VFB (Virtual Functional Bus) qui réalise une abstraction complète des communications entre ces différents SW-C. La standardisation de toutes les interfaces permet de distribuer ces SW-C à travers les calculateurs d'un système spécifique qui est documenté à l'aide des spécifications AUTOSAR. Les fonctionnalités du VFB sont instanciées sur chaque calculateur pour former le RTE (RunTime Environement) et garantir ainsi une compatibilité totale de tous les SW-C où qu'ils soient.

La figure 14 montre une vue d'ensemble de l'architecture logicielle pour un calculateur.

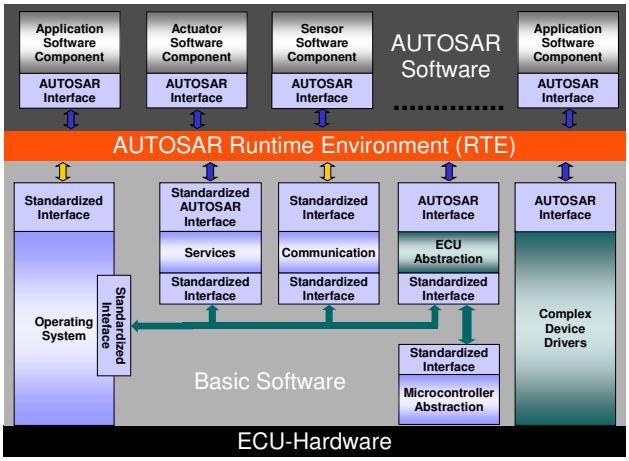

Figure 14. Architecture AUTOSAR d'une UCE

Cette architecture logicielle n'a pas encore fait ses preuves sur des véhicules mais elle est l'orientation la plus forte pour les développements futurs et son impact sur le dimensionnement temps réel est en cours d'évaluation.

4.3. Les architectures dynamiques du logiciel

L'architecture dynamique du logiciel est l'organisation du déroulement de l'exécution du code. Ce document n'aborde que les calculateurs monoprocesseur. Dès à présent, il est important d'identifier trois « étapes dynamiques » existant dans les UCE :

- Etape de « démarrage/réveil » (Startup) : le microcontrôleur sort d'un état de non alimentation, de « reset » ou de veille, un code de lancement s'exécute.
- Etape « d'exécution » (Runtime) : suite au code de lancement, l'application fonctionnelle s'exécute.
- Etape « d'arrêt/mise en veille » (Shutdown) : sur évènements ou à la demande de l'application, le code d'arrêt ou de mise en veille du microcontrôleur (et de l'UCE) est exécuté.

Ces trois étapes ont une dynamique différente, et ceci quelque soit l'architecture choisie. Chaque étape devra répondre à des exigences qui lui sont spécifiques comme par exemple le temps de démarrage ou de réveil, la consommation en courant du calculateur,…

L'étape de « lancement/réveil » dépend de l'architecture système de chaque constructeur. Le rôle d'une UCE, à savoir les fonctions qu'elle héberge, va déterminer les différents aspects que peut prendre cette étape. Typiquement, l'UCE de contrôle de l'habitacle va avoir un nombre de causes de « lancement/réveil » variable selon qu'il intègre ou non la détection des ouvrants, la fonctionnalité d'antivol et bien d'autres. Les traitements exécutés, lors de cette étape, ont comme principale contrainte de devoir être rapide (réveil entre 10 et 100ms par exemple).

L'étape « d'arrêt/mise en veille » est souvent le pendant de l'étape de « lancement/réveil », elle va dépendre du rôle de l'UCE. L'objectif d'une telle étape dans un contrôleur d'habitacle est de réduire au maximum la consommation d'énergie de l'électronique tout en maintenant un petit nombre de traitements spécifiques en activité. L'un de ces traitements étant la détection de causes valides de réveil du système.

Dans le cadre de ce document l'étape « d'exécution » (Runtime) est abordée plus en détail par la suite.

4.3.1. Une dynamique séquencée

Dans un calculateur centralisé avec une architecture logicielle monolithique, la dynamique est souvent basée sur l'exécution des traitements sous forme de séquences au sein d'une boucle périodique. Chaque traitement est caractérisé par sa périodicité, sa durée maximum et sa localisation au sein d'une séquence. La figure qui suit est un exemple montrant l'application du concept de base d'un tel séquenceur. Les conditions de sortie de cette boucle dépendent des fonctions embarquées dans le calculateur.

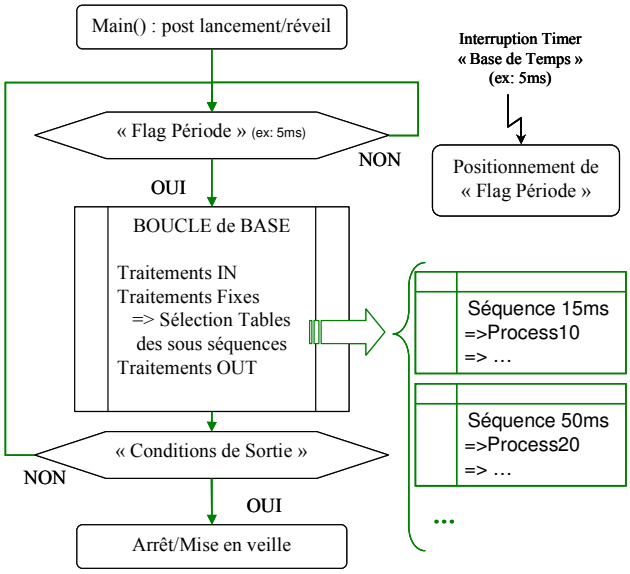

Figure 15. Exécution séquentielle

Généralement, une séquence de base (période très rapide de l'ordre de 5ms) intègre les traitements évaluant les données en entrée (avec ou sans mécanisme de validation de celles-ci), des traitements fixes pouvant être liés à un réseau, la sélection de la table de sous-séquence à exécuter et, enfin, les traitements de pilotage des sorties.

Une telle dynamique cohabite difficilement avec des interruptions pouvant survenir à n'importe quel moment durant l'exécution des séquences. L'arrivée d'évènements est surveillée cycliquement (« polling »). La période de ce cycle fait partie du dimensionnement temps réel des fonctions du système afin qu'il n'y ait pas trop de latence entre l'évènement et sa prise en compte par l'UCE mais aussi pour que le processeur ne soit pas uniquement dédié à cette surveillance.

Sur un petit calculateur avec des fonctions centralisées peu nombreuses et un niveau de priorités homogène, une telle approche a des avantages indéniables tels que sa simplicité de mise en œuvre, sa robustesse et son déterminisme. Malheureusement, lorsque les fonctions se décentralisent (plus de réseaux et/ou une messagerie plus grande), lorsque leur niveau de priorité (voir de sécurité) diverge plus et lorsque leur nombre augmente sur un même calculateur, cette approche atteint rapidement ses limites dans l'automobile. Elle permet de garantir une échéance locale d'exécution pour chaque fonction mais ne tient pas compte des aspects prioritaires de certains traitements sur d'autres à un instant donné. Le cycle servant de base ne peut être très long, cela limite le nombre des traitements dans le calculateur. Les autres inconvénients sont : le manque de réactivité aux évènements et le besoin de déterminer précisément les durées unitaires de tous les traitements. Dans ces conditions l'évolution et/ou la réutilisation du logiciel est extrêmement difficile.

Ces constatations, et d'autres plus liées aux processus de développement, d'intégration et de test, ont amenés les acteurs de l'automobile à s'orienter vers d'autres architectures dynamiques.

4.3.2. Une dynamique ordonnancée et multi-tâches

Le thème de cette publication n'étant pas l'étude des politiques d'ordonnancement, la détermination des temps de réponse des tâches n'est pas développée.

Les changements de la dynamique s'opèrent en parallèle avec l'apparition des architectures statiques structurées et la décentralisation des calculateurs au travers des réseaux.

Les premières solutions sont hybrides. Elles utilisent des séquenceurs combinés avec des interruptions qui intègrent une partie des traitements. Ces séquenceurs regroupent des traitements avec des caractéristiques homogènes (la période et la priorité) et ils sont cadencés par des évènements temporels et/ou matériels. Au fur et à mesure, ils deviennent des tâches et l'allocation du processeur se fait à travers un OS (Operating System).

Ces solutions sont souvent propriétaires. Elles ont des coûts de développement élevés et récurrents, ne tirent pas tous les avantages d'une solution standard en terme de qualité, et, de plus, elles ne permettent pas d'interchangeabilité entre les calculateurs des différents fournisseurs. Elles vont peu à peu céder la place à des RTOS (Real Time Operating System) standard du commerce.

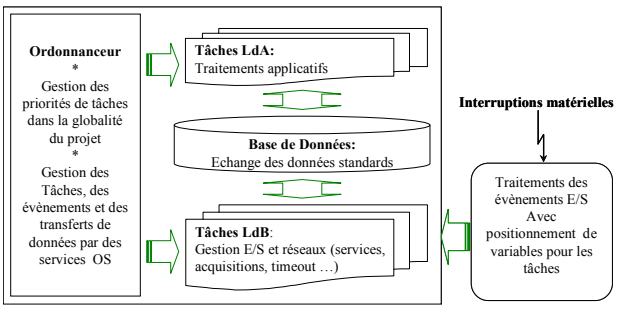

Figure 16. Ordonnancement multi-tâches

OSEK[1] (Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug / Open Systems and the Corresponding Interfaces for Automotive Electronics) se positionne comme la solution standard de RTOS pour les calculateurs automobiles.

Les principaux modules d'OSEK utilisés dans les UCE sont : OSEK OS, OSEK COM (module de Communication) et OSEK NM (module de Network Management). OSEK est un standard publié, ses

[1] Le thème de cette publication n'étant pas la norme OSEK, elle n'est pas détaillée. De nombreux ouvrages et articles sont disponibles dans le commerce [6] ou sur internet [7].

modules sont peu gourmands en mémoire et en puissance, ils sont ajustables et disponibles sur plusieurs plateformes (8, 16 et 32 bits) avec une large offre. Il s'agit d'un RTOS « statique » : tous les objets sont définis avant le lancement de l'applicatif et ne peuvent être détruits.

Les services offerts par OSEK et ses politiques d'ordonnancement (non préemptif, préemptif et mixte) permettent de mieux répondre aux besoins temps réel mais complexifient la conception de l'architecture dynamique et la validation de l'applicatif. La préemption des tâches (phénomène d'interruption de l'exécution d'une tâche en faveur d'une autre tâche plus prioritaire) et la gestion concurrentielle des ressources (mémoires et périphériques) rendent le calcul d'échéance au plus tard d'une fonction (en local) difficile sans des outils mathématiques.

Dans OSEK, les tâches sont à priorités fixes ou encore dites statiques, c'est à dire indépendantes du temps et fixées à priori. La gestion des ressources partagées est faite par Priority Ceiling Protocol (PCP) : une tâche donnée ne peut-être bloquée qu'une seule fois par une tâche moins prioritaire.

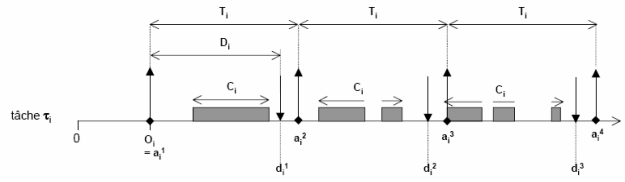

Figure 17. Modèle canonique des tâches

Il existe plusieurs types de tâches parmi lesquels :

- les tâches périodiques (activées à intervalles de temps réguliers),
- les tâches apériodiques (activées à des instants aléatoires),
- les tâches sporadiques (un temps minimal séparant 2 activations successives est connu),
- les modèles fenêtrés (dont le nombre maximal de tâche dans un intervalle de temps donné est connu).

Ces tâches peuvent être décrites à l'aide du modèle canonique de la figure 17. Avec $\boldsymbol{O_i}$ la date d'activation initiale, phase initiale, offset. $\boldsymbol{C_i}$: la durée d'exécution sans préemption. $\boldsymbol{T_i}$: la période d'activation (aussi notée P_i). $\boldsymbol{D_i}$: le délai critique, l'échéance relative. $\boldsymbol{A_i^k}$: $k^{ième}$ requête d'activation de la tâche i.

Pour vérifier et garantir le dimensionnement des échéances temporelles relatives, l'approche est similaire à celle décrite pour les réseaux. Cette approche consiste à considérer un ordonnancement non concret ; les pires conditions d'activation des tâches conduisant aux temps de réponse maximum.

L'amélioration de la réactivité et du comportement temps réel des fonctions pose tout de même un problème de robustesse temporelle : comment garantir un comportement du véhicule fiable en cas de déviation des spécifications temporelles ? Le logiciel embarqué doit se doter de mécanismes de surveillance de bon fonctionnement mais aussi de mécanismes passifs de signalement des dérives et/ou actifs de correction des erreurs afin d'améliorer sa fiabilité. Ces techniques, issues de la Sureté de Fonctionnement, peuvent être coûteuses, d'une part en taille mémoire, d'autre part en temps d'exécution, ce qui augmente les échéances locales des fonctions.

Typiquement, tout développeur doit avoir une obsession lors du développement d'une fonction logicielle : Il doit régulièrement se demander si le code écrit réalisera correctement la fonctionnalité pour laquelle il l'a été, et ce quelque soit les conditions de son exécution. Pour ce faire, le développeur va mettre en place des mécanismes de vérification de cohérence des données en entrée de la fonction. Il utilisera, également si besoin est, les services de l'OS pour la protéger d'autres traitements (tâches et/ou interruptions) plus prioritaires pouvant la perturber. L'utilisation croissante d'interruptions matérielles (et logicielles) demande une attention particulière quant à leur impact sur le reste du logiciel, en particulier sur la dérive temporelle qu'elles peuvent engendrer, sur l'accès aux variables et aux ressources partagées avec les tâches interruptibles.

Ces aspects du développement logiciel font apparaître une surcharge du temps d'exécution des traitements, et ce « time overhead » doit être pris en compte localement (par exemple dans le $\boldsymbol{C_i}$ de chaque tâche) lors du dimensionnement temps réel d'une fonction système.

4.4. Une conception pour l'étude de cas

Application à l'étude de cas de la fonction clignotant pour illustrer les informations sur l'architecture dynamique locale aux différents UCE. L'étude de cas fait intervenir trois UCE dans lesquelles sont distribuées les traitements participant à la réalisation de la fonction clignotant.

Le HDC est directement connecté aux organes de selection des fonctions. Il réalise l'acquisition des évènements liés à ces organes et valide les demandes associées. Le HDC transmet ensuite l'état des informations à sa charge au BC au travers du réseau CAN. L'architecture dynamique est réalisée autour d'un ordonnanceur hybride propriétaire.

Le BC peut directement être connecté à des organes mais reçoit aussi des informations en provenance d'autres UCE, comme le HDC. Le BC contient la stratégie décisionnelle (machine d'état) de la fonction clignotant. En particulier, la gestion de priorité entre fonctions (clignotants versus feux de détresse) et le contexte de fonctionnement général du véhicule (moteur tournant, clé en position accessoire,…). Le BC transmet ensuite sa décision au BFD au travers du réseau LIN. L'architecture dynamique est réalisée autour d'un noyau OSEK du commerce.

Le BFD est directement connecté à des organes de puissance. Il traduit les décisions fonctionnelles du BC en commandes de sortie et pilote les organes de puissance. L'architecture dynamique est réalisée autour d'un simple séquenceur.

Les données chiffrées utilisées pour illustrer cette étude de cas sont issues de la capitalisation du retour d'expérience du département logiciel de la DEI. Les résultats annoncés sont basés sur des calculs de charge CPU et de vérification du respect des échéances temporelles de chaque tâche.

HDC					
Nom des Interruptions	**Ci (µs)**	**Pi (µs)**	**Di (µs)**	**Prio**	**Ni**
CANRxISR	40	40000	500	255	3
CANTxISR	40	100000	800	255	3
TimerISR	70	5000	5000	255	1
Nom des Tâches Non-Préemptives	**Ci (µs)**	**Pi (µs)**	**Di (µs)**	**Prio**	
IOPDriver	170	5000	5000	9	
LdAHigh	1000	10000	10000	3	
LdALow	500	50000	50000	1	
LdBProcess	300	5000	5000	7	
CANManager	500	10000	10000	5	

La charge estimée du processeur du HDC est de l'ordre de 27% et toutes les échéances des différents éléments composant cette architecture dynamique sont respéctées.

Les éléments intervenant dans la fonction clignotant au sein de l'HDC sont IOPDriver (acquisition d'entrée, 3 échantillons), LdBProcess (validation de la demande), CANManager (gestion de la messagerie). Le délai localement estimé, dans le pire cas, est inférieur à 40ms.

BC					
Nom des Interruptions	**Ci (µs)**	**Pi (µs)**	**Di (µs)**	**Prio**	**Ni**
CANRxISR	45	100000	800	255	3
CANTxISR	45	40000	500	255	3
TimerISR	75	5000	5000	230	1
LINTxISR	25	10000	10000	200	9
LINRxISR	25	10000	10000	200	1
Nom des Tâches Non-Préemptives	**Ci (µs)**	**Pi (µs)**	**Di (µs)**	**Prio**	
IOPDriver	200	5000	5000	13	
LINSchedule	100	50000	50000	7	
CANNM	50	50000	50000	5	
LdBProcess	400	5000	5000	11	
CANManager	500	10000	10000	9	
Nom des Tâches Préemptives	**Ci (µs)**	**Pi (µs)**	**Di (µs)**	**Prio**	
BlinkProcess	50	50000	30000	3	
LdAHigh	500	10000	10000	1	
LdALow	1000	50000	50000	0	

La charge estimée du processeur du BC est de l'ordre de 30% et toutes les échéances des différents éléments composant cette architecture dynamique sont respéctées.

Les éléments intervenant dans la fonction clignotant au sein du BC sont CANRxISR (réception de la trame), CANManager (gestion de la messagerie), BlinkProcess (stratégie foncionnelle), LINSchedule (gestion de la table des messages). Le délai localement estimé, dans le pire cas, est inférieur à 120ms.

BFD					
Nom des Interruptions	**Ci (µs)**	**Pi (µs)**	**Di (µs)**	**Prio**	**Ni**
TimerISR	10	1000	1000	255	1
Nom des Tâches Non-Préemptives	**Ci (µs)**	**Pi (µs)**	**Di (µs)**	**Prio**	
BackgroundSchedule	800	1000	1000	0	

La charge estimée du processeur du BFD est de l'ordre de 80% et les échéances sont respéctées. Le délai localement estimé, dans le pire cas, est inférieur à 2ms.

Il est maintenant possible de finaliser la démarche globale de dimensionnement du délai de « bout-en-bout » de la fonction. Pour cela, à l'aide de ces résutats et de ceux obtenus précédemment sur les délais de transmission sur les réseaux, on vérifie que l'exigence de 300ms de délai d'activation du clignotant est respectée. Cela à l'aide de l'équation du §3.5.1 adapté à cette étude de cas :

$R_{MAX} = R_{HDC} + MAX_{CAN} + R_{BC} + MAX_{LIN} + R_{BFD}$

$R_{MAX} < 175ms$

Le délai maximum de « bout-en-bout » de la prise en compte d'une commande à sa réalisation pour la fonction clignotant respect l'échéance exigée de 300ms.

5. Quelques perspectives d'évolution

Le dimensionnement temps réel dans les véhicules abordé dans ce document au travers de l'expérience de Valeo DEI dans la partie habitacle, montre avant tout à quel point cette problèmatique a évolué suite aux besoins croissants de compétitivité et de fiabilité de l'automobile. Des changements fondamentaux dans les architectures électriques des véhicules mais aussi dans les architectures logicielles des UCE ont été induits par le nombre grandissant de fonctions.

La réalisation de ces changements engendre l'apparition de nouvelles exigences nécessitant d'autres évolutions qui, elles-même impliquent des changements…Ainsi, l'automobile est pris dans une boucle sans fin d'évolutions.

Du point de vue système, si les architectures électriques multiplexées seront toujours d'actualité, les réseaux qui les composent vont évoluer. La nécessité de répondre aux besoins de déterminisme, de tolérance aux fautes et de plus de débit va introduire, sans aucun doute, un nouveau type de protocle : Time-Triggered Protocol (TTP), et cela, au travers de FlexRay Automotive Communication Bus et/ou de Time-Triggered communication on CAN (TTCAN).

Une approche pourrait être d'utiliser, un réseau FlexRay comme « colonne vertébrale » de l'architecture multiplexée des véhicules, reliant ainsi les différents domaines fonctionnels. Cette proposition à l'avantage de fiabiliser les informations inter domaines et de réduire

l'influence des domaines sur les échéances temporelles des messages.

Du point de vue calculateur, la tendance semble être à leur homogénéisation, et cela à travers leur puissance de calcul pour mieux répondre aux besoins de distribution des fonctions. A ce stade de notre reflexion, trois possibilités sont envisageables :

- une tendance aux calculateurs monoprocesseur plus puissants, voir sans doute d'une nouvelle génération (32 bits avec plus de périphériques intelligents et des espaces mémoires protégés,…),
- une tendance aux calculateurs biprocesseurs de puissance équivalente issue de la génération actuelle (16 ou 32 bits reliés ensemble),
- une tendance aux calculateurs biprocesseurs de puissance différente. Le moins puissant servant de superviseur à l'autre.

L'objectif de cette approche est, d'une part, de réduire les coûts d'achat composant en jouant sur les volumes, d'autre part, de réduire le coût de développement lié au logiciel en réutilisant un maximum d'éléments et en facilitant la portabilité des autres.

L'approche du Consortium AUTOSAR semble être la perspective la plus plausible pour l'évolution de l'architecture statique du logiciel. Cette orientation a pour objectif d'autoriser la cohabitation des fonctions de confort avec des fonctions critiques sur la même UCE. Cela n'est pas sans conséquence sur la conception globale du logiciel et sur l'Operating System (OS) qui doivent répondre aux exigences de Sûreté de Fonctionnement. De ce fait, l'OS en cours de spécification par AUTOSAR intègrera des mécanismes de protection spatialle (espaces mémoires) et temporelle (contrôle des échéances).

La dynamique temps réel des calculateurs tend à s'ancrer encore plus autour de conceptions multi-tâches préemptives. Sur ce sujet, l'approche du Consortium AUTOSAR, en cours de définition, devra être validée sur des prototypes avant son déploiement sur véhicule. Quoiqu'il en soit, dès à présent, les besoins d'outils mathématiques et de langages formels, pour formaliser la conception dynamique d'une UCE, se font sentir. Indéniablement, cela permettra, en amont du développement, de valider que le processeur n'est pas surchargé et qu'il existe au moins un ordonnancement garantissant que toutes les échéances seront respectées, et cela même dans le pire cas.

La tendance dans l'automobile est à l'augmentation de l'électronique, et par la même occasion, du logiciel au sein des véhicules. Les préoccupations actuelles de fiabilité des systèmes et de sécurité des Hommes seront au coeur des évolutions.

Références

[1] D. Paret, « Le Bus CAN – Description / De la théorie à la pratique », Série Electronique appliquée, DUNOD, 1996

[2] http://www.lin-subbus.org/

[3] J.W. Specks, A. Rajnák, « LIN – Protocol, Development Tools, and Software Interfaces for Local Interconnect Networks in Vehicules », 9th International Conference on Electronic Systems for Vehicles, Baden-Baden, October 5/6, 2000

[4] S. Martin, P. Minet, L. George, « End-to-end response time with Fixed Priority scheduling : trajectory approach vs. Holistic approach », IJCS (International Journal of Communication Systems), Wiley. Volume 18, Issue 1, Pages 1-95 (February 2005)

[5] K. Tindell, H. Hansson, A.J. Wellings, « Analyzing real-time communication : Controller Area Networks (CAN) », Real-Time Systems, p259-263, 1994.

[6] J. Lemieux, « Programming in the OSEK/VDX Environment », CMP Books, ISBN 1-57820-081-4

[7] http://www.osek-vdx.org/