Methodology for Model-based Timing Analysis Process for Automotive Systems

Méthodologie pour un Processus d'Analyse Temporelle Dirigé par les Modèles pour les Systèmes Automobiles

THESE

Présentée et soutenue publiquement le

09 Novembre 2011

Par

SAOUSSEN ANSSI

Pour obtenir le grade de

DOCTEUR EN INFORMATIQUE DE

l'Université Paris-Sud

Devant le jury composé de :

Prof. Guy Vidal-Naquet, Université Paris-Sud	Président
Prof. Lionel Briand, Université d'Oslo	Rapporteur
Prof. Jean-Philippe BABAU, Université de Brest	Rapporteur
Prof. François TERRIER, CEA LIST	Directeur de thèse
Dr. Ing. Sébastien GERARD, CEA LIST	Encadrant de thèse
Ing. Stefan Kuntz, Continental Automotive	Encadrant de thèse

To my father,

For all the love he gave to us…

Acknowledgements

This work has been financially supported by the ANRT (Association Nationale de la Recherche Technique) in France.

I am grateful to my thesis committee for devoting their time to read the manuscript and to participate in my dissertation. I have particularly appreciated the thoroughness and insightful comments from Lionel Briand and Jean-Philippe Babau on earlier drafts.

Special thanks go to my thesis supervisor Prof. François Terrier for giving me the opportunity to develop this thesis at the CEA LIST labs.

I want to thank my advisor Sébastien Gérard for his crucial support in the development of the ideas presented here.

I am grateful to my technical advisor in Continental Stefan Kuntz for his support during this work and the very interesting technical discussions that we had together. I would like to salute his technical brilliance and his strong cultural sensitivity. This work allowed me to learn much from him.

I would like to thank Frédéric-le-Hung, our team manager in Continental, for his support during these three years and his open-mindedness.

I had the pleasure to work with Denis Claraz and Philippe Cuenot in Continental; I would like to thank them for their interesting feedbacks and recommendations for the deployment of this thesis work in Continental.

Special thanks go to my earlier technical advisor in CEA, Huascar Espinoza, for his help and recommendations during the state-of-the-art studies.

During these three years I had the pleasure to participate to two research projects, EDONA and Memvatex. I would like to thank all the members of these projects. In particular, I would like to warmly thank Sara Tucci from CEA for her interesting ideas for the scientific publications that we wrote together.

I am thankful to all the people of the LIST lab in CEA and the ADE team in Continental. I always found high availability of technical support, a very comfortable discussion environment, a lot of resources to develop the work efficiently, wide openness for work diffusion, and team participation.

I am immeasurably thankful to my parents, Sayeh and Khadija, my brothers Imed and Ridha, my sisters Naziha, Salwa, Henda, and my cousins Sonia and Randa in Tunisia for all their love, sacrifices and endless encouragement. They always stand by me and never let me down in spite of physical distance. I owe all my success to their love, care and sacrifices. They taught me the real sense of sacrifice and altruism.

Finally, there are no words to express my gratitude to my dearly beloved husband Ahmed-Amine for his sacrifices and everlasting encouragement. He gives me energy, peace of mind and happiness. This thesis is dedicated to him for everything he means to me…

RESUME

Ce travail de thèse a été effectué dans le cadre d'une collaboration technique entre le CEA-LIST à Paris et le service « développement avancé électronique » de Continental Automotive à Toulouse.

1. Objectifs de la thèse

Dans ce travail de thèse on se propose de définir une méthodologie décrivant un processus d'analyse temporelle dirigée par les modèles pour les systèmes automobiles. Cette méthodologie vise à donner un guide aux ingénieurs de développement logiciel automobile pour l'intégration de la vérification temporelle dans un processus de développement dirigé par les modèles. Ceci permettrait alors la détection au plus tôt des erreurs de conception liées au comportement temps réel des systèmes.

En plus de la définition de la méthodologie elle-même, sa validation doit être aussi étudiée en montrant à quelle mesure elle contribue à résoudre les problèmes rencontrés actuellement dans le domaine du développement logiciel automobile. L'acceptabilité de la méthodologie est également à étudier pour évaluer son potentiel d'adoption pour le développement des systèmes de contrôle moteur (Engine management System EMS) à Continental.

2. Contexte de la thèse

2.1. Contexte Industriel

Aujourd'hui, l'architecture des systèmes automobile est devenue de plus en plus complexe avec une utilisation massive du logiciel embarqué pour assurer les diverses fonctionnalités d'une voiture.

Pour répondre correctement aux besoins de ces clients ainsi qu'aux contraintes de concurrences, un équipementier (tel est le cas de Continental Automotive) doit considérer deux facteurs essentiels: la maitrise du temps et du coût du développement logiciel ainsi que la garantie de la fiabilité du système conçu. Vue la complexité croissante du logiciel embarqué automobile, la garantie de sa fiabilité dépend énormément de la capacité de maîtriser cette complexité lors du développement. En plus de la maîtrise de la complexité, la fiabilité des systèmes automobiles doit être également assurée à travers les techniques de vérification et de validations. La vérification et la validation des contraintes de temps est d'une importance énorme pour garantir cette fiabilité. Aujourd'hui la vérification temporelle des systèmes automobiles est effectuée très tard au cours du développement (après la phase

d'intégration). Elle se base essentiellement sur des tests et des mesures plutôt que sur une approche formelle et systématique. Ainsi, Pour développer un logiciel fiable tout en respectant les contraintes de concurrences, il y a un besoin fort pour des approches de développement qui permettent de : **1) Maîtriser la complexité du logiciel lors du développement.2) Réduire le temps et le coût de développement. 3) Définir une activité de développement ainsi qu'une chaine d'outils homogène et continue.4) Permettre l'intégration de la vérification temporelle au cours du processus de développement**.

2.2. Approches existantes

Pour apporter des solutions aux besoins du développement du logiciel automobiles, plusieurs approches, méthodes et techniques ont été définit aux cours de la dernière décennie. Ces approches visent soit à donner des méthodes de développement permettant l'amélioration des processus de développement des systèmes automobiles (tel est le cas des approches définit dans le cadre de l'ingénierie dirigée par les modèles), soit à permettre de vérifier le comportement temps réel des systèmes (comme les technique d'analyse d'ordonnancement et de performance).

Dans le domaine automobiles, les approches et langages de modélisations qui ont été définit sont:

- EAST-ADL : Ce langage permet la modélisation de l'architecture électrique/électronique des systèmes automobiles suivant plusieurs niveaux d'abstraction. Il donne plusieurs concepts permettant la modélisation de la structure fonctionnelle (sur les niveaux Analyse et Design) et matérielle (à partir du niveau design) des systèmes automobiles. La Figure 1 montre les niveaux d'abstraction d'EAST-ADL (le niveau implémentation s'appuie sur les concepts d'AUTOSAR)

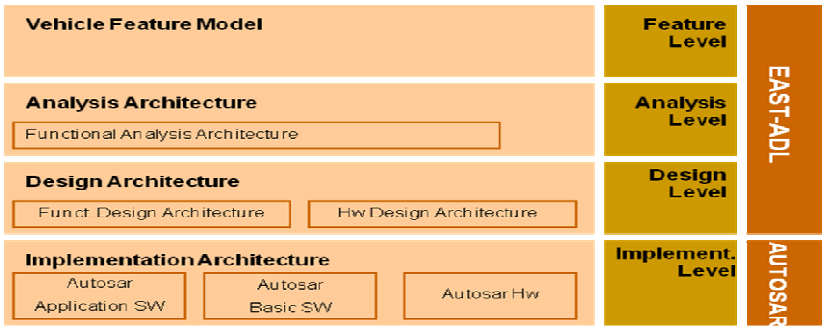

Figure 1 Niveaux d'abstraction d'EAST-ADL

- TADL : Ce langage permet la modélisation des propriétés et des contraintes temporelles des éléments structurels décrits dans une architecture EAST-ADL ou AUTOSAR.
- AUTOSAR : Il donne une approche pour décrire une architecture logicielle standard pour l'automobile. Il offre un modèle d'architecture logicielle organisé suivant trois niveaux : le logiciel applicatif, le RTE (RunTime Environment) et les couches logicielle de bas nivaux (basic software). Le RTE fait le lien entre le logiciel applicatif et les couches logicielle basses. Le logiciel est organisé sous forme de composants logiciels (software components). Pour chaque composant, il est possible de décrire les unités exécutables qu'il contient (runnable entities) ainsi que ses interfaces de communication (port)
- MARTE : Ce langage permet de modéliser l'architecture des systèmes temps réel. Il offre un set de concepts de modélisation pour permettre d'effectuer de l'analyse d'ordonnancement basée sur les modèles.

Parmi les techniques de vérification temporelle on cite essentiellement l'analyse d'ordonnancement. Dans le contexte de cette technique, plusieurs tests d'ordonnançabilité ainsi que des outils d'analyse d'ordonnancement ont été développés. Parmi ces outils il ya des outils académique tel que Cheddar et MAST et d'autres commerciaux tel que SymTA/S et Chronval. L'évaluation de ces outils d'analyse montre que SymTA/S est le plus adapté pour faire de l'analyse d'ordonnancement pour les applications automobiles.

L'évaluation des ces approches de développement et de vérification temporelle (effectué aux cours de ces travaux de thèse) a montré qu'il y a un **manque pour un guide méthodologique pour l'intégration de la vérification temporelle notamment l'analyse d'ordonnancement au cours du cycle de développement dirigé par les modèles. Ce travail de thèse propose une approche qui permettrait de résoudre ce problème.**

3. Méthodologie

3.1. Objectifs de la méthodologie :

- Définition d'un processus de développement dirigé par les modèles qui assure une activité de développement continue et homogène tout au long du processus. Ce processus doit être facilement utilisable par un ingénieur Continental pour le développement des systèmes de control moteur (Engine Management Systems

EMS). La méthodologie définit doit décrire les différentes phases du processus ainsi que l'approche suivit pour le développement et l'affinement des modèles d'une phase à autre.

- Donner un guide pour l'intégration de la vérification temporelle dans ce processus de développement. Ceci requiert la définition du type de vérification temporelle à effectuer durant chaque phase, les techniques et les outils de vérification temporelle à utiliser ainsi que la description de la manière d'utiliser les résultats d'analyse de chaque phase pour affiner les modèles de la phase suivante.
- Décrire la manière de développer des modèles analysables. Plus particulièrement comment extraire à chaque phase les modèles comportementaux nécessaires pour l'analyse temporelles des modèles architecturaux utilisés pour la description de l'architecture.

Figure 2 donne une vue générale du processus d'analyse temporelle dirigé par les modèles que la méthodologie vise à définir ; aux cours de chaque phase, le concepteur a en entrée un nombre d'exigences temporelle, il conçoit donc l'architecture qui doit respecter ces exigence et puis il effectue une analyse temporelle pour vérifier que l'architecture conçu respecte bien ces exigences.

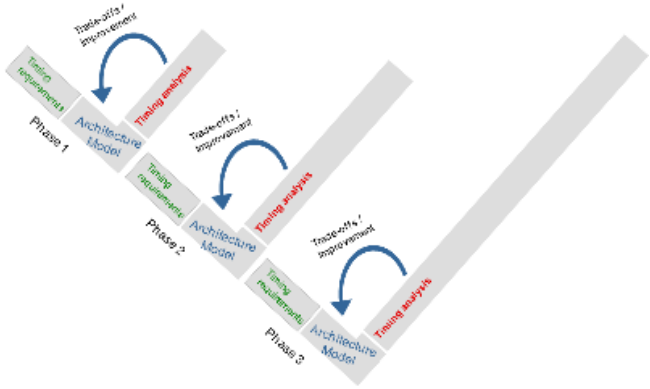

Figure 2: Processus d'analyse temporelle dirigé par les modèles

3.2. Description de la méthodologie :

Notre méthodologie propose de réutiliser les niveaux d'abstraction du processus de modélisation d'EAST-ADL/AUTOSAR (figure 2) pour définir notre processus d'analyse temporelle dirigé par les modèles. Cependant, le processus EAST-ADL/AUTOSAR présente seulement les niveaux d'abstraction et les concepts à utiliser à chaque niveau. Il ne

donne aucune approche décrivant la manière d'affiner les modèles d'un niveau à autre. En plus, il ne propose aucune chaine outil pour supporter le processus de développement. Notre méthodologie doit donc adapter et améliorer ce processus pour apporter des solutions pour ces problèmes.

On propose de commencer l'analyse temporelle à partir du niveau Analyse car le niveau Véhicule ne donne pas assez de moyen permettant d'effectuer une analyse temporelle. Notre processus se compose donc de trois phases : Analyse, Design et Implémentation. Sur chaque phase, on décrit les activités de modélisation ainsi que d'analyse temporelles qui doivent se faire.

Le figure 3 donne une vue générale du processus définit. Les paragraphes suivants expliquent les activités effectués durant chaque phase.

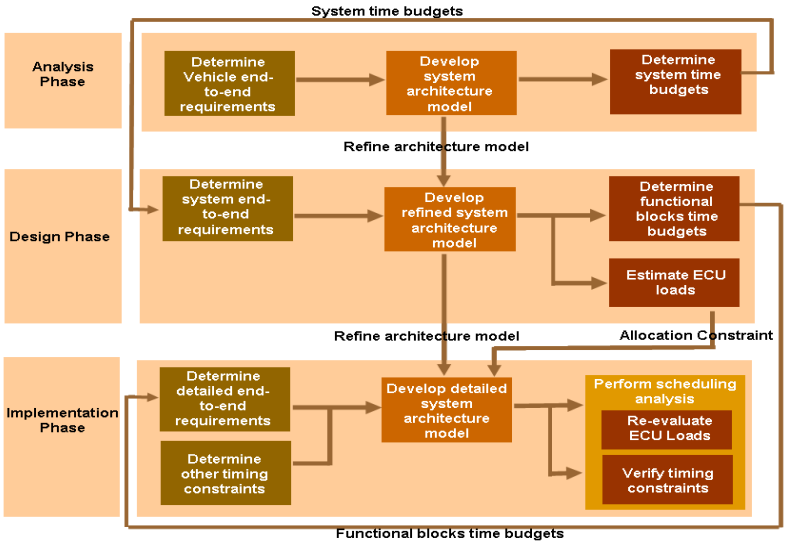

Figure 3 Les phases de la méthodologie proposée

Phase Analyse

Au cours de cette phase, une vue décrivant l'architecture fonctionnelle du système (« sub-system analysis funtional view ») est développée en utilisant les concepts d'EAST-ADL pour la modélisation fonctionnelle et les « composite structure diagram » d'UML2. Cette vue décrit le système dans son environnement véhicule. Une deuxième vue (« sub-system analysis timing view ») décrivant le comportement temporel du système est développé à partir de la première vue fonctionnelle en utilisant les concepts de TADL et les diagrammes de séquence pour représenter les informations temporelles du système (notamment les contraintes temporelles). L'analyse temporelle effectuée durant cette phase se base sur cette

dernière vue comportementale. L'analyse temporelle de cette phase vise à vérifier la bonne intégration du système dans le véhicule en termes de compatibilité temporelle. Le concepteur a en entrée une liste d'exigences de bout-en-bout (end-to-end requirements). Ces exigences impliquent le système en cours de développement et les autres fonctions du véhicule qui communiquent avec lui. Pour chaque exigence de bout-en-bout, le concepteur doit déterminer un budget temps (« time budget ») qu'il faut allouer au system en cours de développement pour respecter cette exigence. Chaque budget déterminé durant la phase analyse représente une contrainte à respecter durant la phase design. La figure 4 montre les activités de modélisation et d'analyse effectuées durant cette phase.

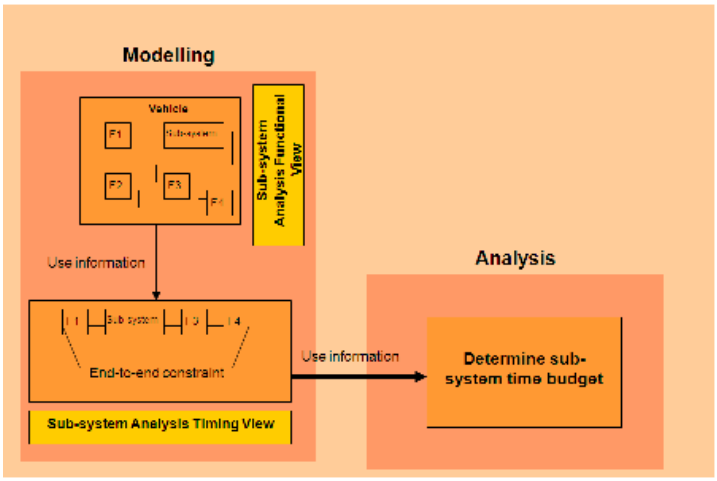

Figure 4 Activités de modélisation et d'analyse temporelle de la phase Analyse

Phase Design

Durant cette phase, la décomposition fonctionnelle du système est représentée à travers une vue qui décrit les blocks fonctionnels qui le composent (« sub-system design functional view »). Les ressources matérielles sont aussi décrites durant cette phase (« hardware platform view »). Le concepteur effectue donc deux types d'analyse temporelle : la première consiste à affiner les budgets temps alloués au system durant la phase précédente (Analyse) en déterminant les budgets temps qu'il faut allouer à chaque block fonctionnel qui le compose. La deuxième analyse temporelle consiste à explorer l'architecture matérielle pour déterminer la meilleure plateforme matérielle à utiliser (en termes de performance) ainsi que le meilleur scénario d'allocation des blocks fonctionnels aux ressources matérielles. Ceci est fait par le biais d'une exploration empirique d'un nombre de scénarios d'allocation candidats qui se base sur le calcul de l'utilisation des processeurs pour chaque scénario.

Remarque : Dans cette dernière analyse temporelle (exploration de l'architecture matérielle), on ne considère pas de modèle de ressources logicielle tel que les tâches OS ni l'allocation des blocks fonctionnels à ces ressources. On considère seulement le modèle fonctionnel, le modèle de ressources matérielles et l'allocation directe des blocks fonctionnels aux ressources matérielles.

La figure 5 montre les activités de modélisation et d'analyse effectuées durant cette phase.

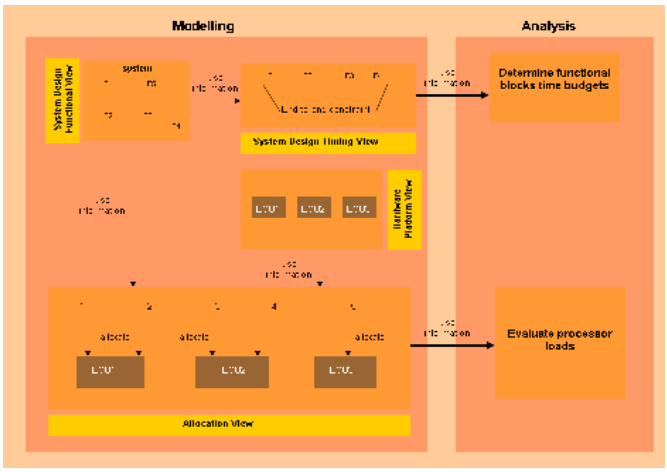

Figure 5 Activités de modélisation et d'analyse temporelle de la phase Design

Phase Implementation

Durant cette phase, un modèle complet décrivant les différents aspects nécessaires pour effectuer une analyse d'ordonnancement (architecture logiciel, ressources logicielle et matérielles, allocation, etc.) est développé en utilisant les concepts d'AUTOSAR. Ce modèle est l'affinement du modèle fonctionnel et matériel développé au cours de la phase design en se basant sur les résultats d'analyse temporelle effectué durant cette phase (design). La figure 6 montre les activités de modélisation et d'analyse effectuées durant cette phase.

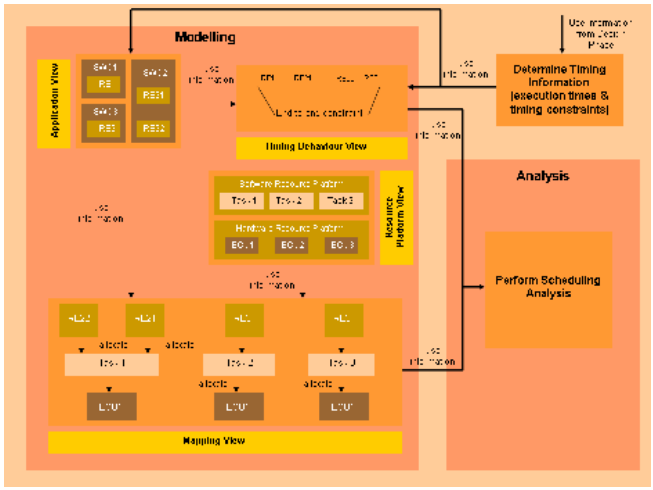

Figure 6 Activités de modélisation et d'analyse temporelle de la phase Implémentation

4. Déploiement et Validation de la méthodologie

4.1. Déploiement de la méthodologie

Dans cette partie, on propose une approche de déploiement de la méthodologie en décrivant la manière de l'appliquer dans le contexte de développement des fonctions de contrôle moteur à Continental (EMS). Figure 7 décrit le processus de développement actuel des EMS chez Continental.

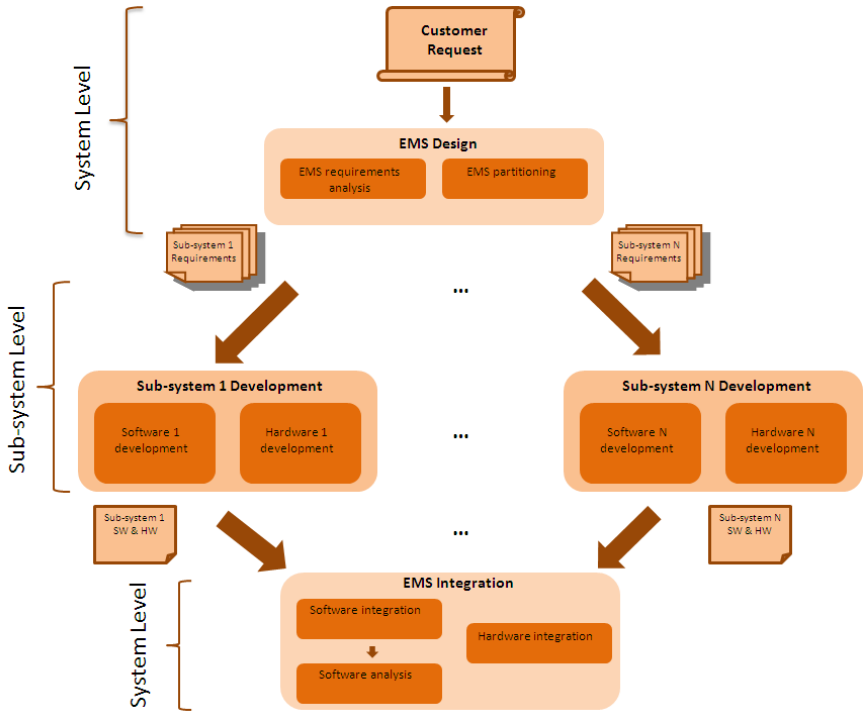

Figure 7 Processus actuel de développement des EMS

La figure 8 montre ce même processus dans le cas de l'application de notre méthodologie

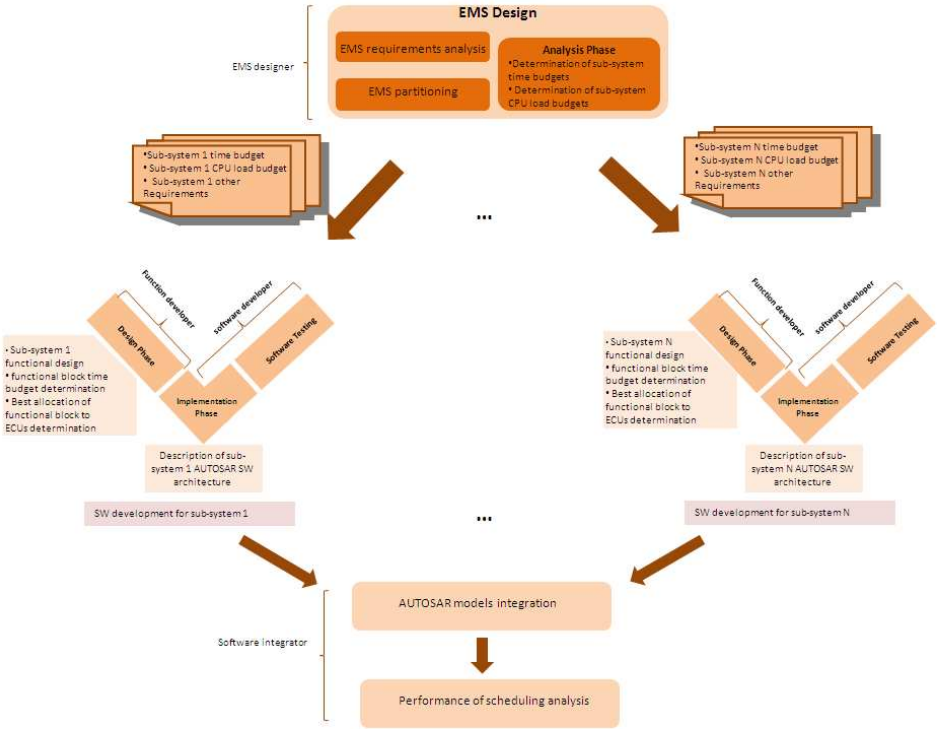

Figure 8 Application de la méthodologie pour le développement des EMS

4.2. Acceptabilité de la méthodologie

En se basant sur l'approche de déploiement décrite précédemment, on a étudié l'acceptabilité de notre méthodologie en termes de compétences demandés, les tâches à effectuer aux cours de chaque phase ainsi que la chaine d'outil proposée. Tous ces éléments ont été comparé avec les compétences, tâches et chaine d'outil utilisé actuellement chez Continental. Cette étude montre un bon potentiel d'adoption de notre méthodologie chez Continental. Ceci est particulièrement valide pour la phase Implémentation de la méthodologie surtout que Continental est en train de migrer vers une nouvelle plateforme basé sur les concepts d'AUTOSAR.

4.3. Validation générale de la méthodologie

La contribution de la méthodologie à satisfaire les besoins du développement logiciel automobile (présentés dans la section 1) a été aussi étudiée. La valeur ajoutée de notre méthodologie consiste à donner la possibilité de commencer l'analyse temporelle tôt au cours de processus de développement (beaucoup plus tôt que l'analyse temporelle effectuée actuellement chez Continental). Ceci permet de réduire le temps et le coût nécessaires pour l'amélioration de l'architecture en cas de détection tardive d'erreur. En outre, notre

méthodologie permet une bonne maitrise de la complexité de l'architecture conçue tout au long du processus de développement.

Contents

Introduction

This chapter introduces the thesis work that has been performed in the context of a technical collaboration between the CEA LIST near Paris and the Advanced Development Electronics (ADE) service of Continental Automotive in Toulouse.

CEA LIST is a key software system and technology research center whose mission consists among others in providing methodologies and tools for real time embedded system development (systems architecture and design, methods and facilities for software and system dependability, etc). This laboratory works on several research projects in partnership with industrial partners from nuclear, automotive, aeronautical, defense and medical domains. Thus, the laboratory investigates and develops innovative solutions corresponding to the requirements of these industrial partners.

The ADE service is a part of the Engine System (ES) business unit within the Powertrain division at Continental. This service provides innovative techniques and methodologies for the development of automotive electronic systems. These innovative approaches aim at providing solutions for the challenges met to develop Engine Management Systems (EMS) within the ES business unit. An Engine Management System (EMS) is a system used to control the engine functionalities (e.g., Combustion, injection, ignition, etc). At Continental, Engine Management Systems are developed to control many types of gasoline and diesel engines for many customers all over the world. To develop these systems, many requirements should be satisfied, including customer requirements but also environmental norms (gas emission). In addition, due to the competition factor, development time and cost for engine management systems should be mastered. The ADE service investigates innovative approaches to meet all these challenges in future engine management system generations.

This introduction is divided in three sections. The first section presents a brief overview of the general technical context in which this work has been done. The second section presents the thesis objectives and the third section describes the outline of the manuscript.

1. Thesis Technical Context

Automotive real time systems are characterized by increasing complexity and tight requirements for safety and timing. Today, highly competitive automotive industries developing real-time systems must face industry requirements both quickly and dependably. "Quickly" refers to the "time-to-market" issue, where delays in design or implementation incur penalties and reduce market profit. "Dependably" refers to the trustworthiness of the services provided by developed systems. One of the key dependability factors in real time systems is system failure. Unlike fabrication faults and faults during usage, design faults are supposed to be found and eliminated by system verification. Hence, whenever fault tolerance cannot be guaranteed, fault prevention is the only way to avoid system failure [13].

Quantitative analysis [1] (such as performance and scheduling analysis) is a sound approach to study non-functional properties at an early stage. It allows designers to detect unfeasible real-time architectures, prevent costly design mistakes, and provide an analytical basis to assess design tradeoffs associated to resource optimization. Quantitative analysis uses mathematical-based techniques which purpose is to prove that a system meets its requirements at any time. While the maturity of quantitative techniques has led to a set of well established mathematical formalisms such as rate monotonic analysis (RMA) [2], Petri nets [3], queuing theory [4] and timed automata [5], their widespread use with complex industrial systems and into integrated tool environments still remains largely open. Quantitative analysis is a difficult and time-consuming task, and to save time, many industries either forgo it until absolutely necessary or train their designers to perform preliminary analysis. However, most designers are under-trained in analysis and too busy to perform useful analysis.

Model-Based Engineering (MBE) is gaining momentum in automotive system and software development domains, as a means for mastering system complexity and assessing system-level tradeoffs geared to achieving higher quality and dependability [6]. MBE and modeling languages lead a major approach to enrich real-time systems engineering practices, by moving the development process from lines-of-code to coarser-grained architectural elements. One of the advantages expected from this approach is the ability to employ correct-by-construction, but also incremental design processes (which rely extensively on automated transformations and synthesis) and to formalize computer-based correctness analysis.

The model-based development community has invested special efforts in incorporating the abilities to specify analytical constructs and non-functional properties with enough expressive power, while still preserving the modeling abstraction level used by MBE practitioners. Important research work has been carried out in order to provide modeling languages (e.g., UML [7], SDL [8], AADL [9], MARTE [10], and TADL [11]) with clear and well-formed semantics to support quantitative analysis.

However, most of the current works are characterized by providing only means and concepts for the modeling of non-functional and especially timing information of the system. Unfortunately, none of these approaches provide sufficient guidance on how to integrate timing verification and validation into the model-based development process.

2. Thesis Objectives

The underlying work investigates the definition of a methodology describing a model-based timing verification process for automotive systems. It aims at giving guidance to software development engineers about how to integrate timing verification within a model-based development process enabling hence early detection of time-related errors.

In particular, this thesis work focus on the following specific objectives:

1. One fundamental objective that drove our research work is the definition of a model-based development process ensuring a seamless development activity that can be easily adopted in the context of engine management system development. The methodology defined should describe the different phases of the model-based process and how models should be refined from one phase to another.
2. The second objective is to give guidance on how to integrate timing verification in each phase of this development process. This means defining the kind of timing verification that should be performed during each phase, the verification techniques and tools that can be used and how analysis results of each phase can be used to refine the architecture during the next phase.
3. From a modeling and analysis point of view, the methodology defined should give a way on how to develop analyzable models in each phase and especially how to derive behavioral views needed for timing analysis from modeling views intended e.g. for structure description.

4. Besides the definition of the methodology itself, in this work we aim to validate the methodology suggested by evaluating its degree of acceptability and showing to which extent it allows resolving the problems faced currently in the context of automotive software development.

3. Thesis Outline

This manuscript is composed of three major parts. The first part contains three chapters. The first chapter describes precisely the particular context of this study related to the development of automotive software in the particular case of Continental as a supplier. This chapter ends by listing the needs of automotive domain in term of software development. The second chapter gives an investigation and a state-of the art of the available model-based approaches that attempted to bring answers and solutions to some of these needs. The third chapter draws the general features of our approach to define a methodology for a model-based timing analysis process. This is done after studying the feasibility of the approach based on the chosen directions that will be presented in the same chapter.

The second part of the manuscript presents the methodology itself. This part is composed of four chapters. The first chapter gives a general overview of the defined process. The remaining three chapters tackle respectively the different process phases, the analysis phase, the design phase and the implementation phase. Each chapter describes both the modeling and timing analysis activities carried out during each phase.

The third part is dedicated to the deployment and validation of the proposed methodology. This part is composed of four chapters. The first chapter presents an approach describing how we intend to apply our methodology for the development of Engine Management Systems (EMS) at Continental. The second chapter illustrates the approach by presenting the application of the methodology to the development of two use cases. The third chapter studies the acceptability of the methodology by showing the extent to which this methodology can be adopted by Continental engineers. In this chapter we identify the gap between the proposed methodology and the current development process at Continental in terms of required vs. available skills, tasks, tool chain, etc.

The methodology tooling is also studied by presenting the tools that were implemented to guide Continental engineers and ease their use of the methodology.

The last chapter of this part presents the final validation of the methodology by showing to which extent it provides solutions for the automotive software development needs determined during the first part of this work.

The conclusion summarizes the study and discusses the possible perspectives for this work.

Part I: Industrial Context and Related Work

This first part aims at describing in detail the industrial context in which this thesis work was done. The technical directions chosen for this work to meet the automotive domain needs are presented and justified based on this context itself but also based on some already available approaches. The first chapter presents the automotive context and highlights the automotive needs in term of software development and timing verification. The second chapter presents the available approaches that attempted to bring solutions for these needs. We highlight the limitations of these approaches and we conclude on the need for a new approach to satisfy better the automotive needs. The third chapter gives a general overview of the directions chosen for our work based on the available approaches and a feasibility study for our approach.

1. Automotive State of Practice and Challenges

In the automotive domain, the first time that software was deployed into cars was to control the engine and, in particular, the ignition, 30 years ago [6]. At the beginning of software deployment in cars, software-based solutions were very local, isolated and unrelated. Hence, there were dedicated controllers (Electronic Control Units or ECUs) for the different functions as well as dedicated sensors and actuators. With the intention to optimize wiring, bus systems were deployed into cars allowing ECUs to be connected to each other and exchange information.

Today, premium cars feature not less than 70 ECUs connected by more than five different bus systems [6]. Within only 30 years, the amount of software in cars went from zero to more than 10.000.000 lines of code. More than 2000 individual functions are realized or controlled by software in premium cars. Software as well as hardware became enabling technologies in cars. They enable new features and functionalities. Hardware is becoming more and more a commodity while software determines the functionality and therefore becomes the dominant factor for system complexity.

To understand better the automotive needs in term of software development, it is important to clarify the state of practice in this domain. The development of a car involves mainly two partners, the manufacturer (OEM) and the first-tier suppliers. The aim of the manufacturer is to market cars that satisfy the needs and the desires of the customers, on one hand by respecting the manufacturing standards and norms and on the other hand by ensuring the prosperity of his group [12]. For these reasons, manufacturers have usually a strong and global trade expertise. A car can be seen as an assembly of many systems integrated together to ensure the various functionalities of the vehicle. The OEM intervenes during two particular phases, the specification of systems and their integration into the vehicle. The development of these systems is then carried out by the different suppliers that are involved; such as the case of Continental Automotive. Once the request is specified by the manufacturer, the supplier should develop the system that respects the requirements specified. In this context of multi-partner development, the systems developed by a supplier are more and more sophisticated and require usually highly specialized technical skills. Due to concurrency pressure, manufacturers choose then to delegate the development of such systems to several suppliers and focus only on vehicle integration and validation. In the case of Continental Automotive, a system requested by a manufacturer may vary from a simple

software component to a whole system consisting of software, hardware (ECU) and mechanics (actuators, etc).

Being able to satisfy efficiently customer request is the key factor for a supplier to save his place in market. Efficiently means quickly, dependably and in a cost-efficient way. Quickly refers to the time-to-delivery where the supplier is continuously submitted to the customer pressure to deliver systems as early as possible. Development cost is also among the decisive factors that guarantee the competitiveness of a supplier. Up to 40% of the production costs of a car are due to electronics and software. Today, the costs of cars get more and more influenced by development costs of software; 50-70% of the development costs of the software/hardware systems are software costs [6].

Dependability means the trustworthiness of the service delivered by the developed system. To develop dependable systems, suppliers should take up many challenges. In fact, the size and structure of the embedded software and hardware in cars are enormous. Most of the software is hard real time critical or at least soft real time critical. Several functions are safety critical ones. In addition, car functions are quite heterogeneous (from embedded real time control to infotainment, from comfort functions to driver assistance, etc). As a result, the complexity and spectrum of requirements for on board software is enormous. In front of this complexity and time/cost pressure, suppliers have usually recourse to reuse existing solutions from one car to the next. However this remains insufficient with regard to development time and cost[1]. In addition, the amount of automation in software production is today quite low. Tools are many times used in an isolated manner. There is neither a properly formalized design flow nor seamless tool chain for distributed functions.

It is hence obvious that there is a need for a suitable development process that reduces complexity, enables innovation and saves time and costs.

Guaranteeing dependability is not ensured only by mastering system complexity. In fact verification and validation is also of paramount importance in software development. This allows verifying the proper functioning of the system and validating it against the requirements specified by the customer. As mentioned previously, developing time critical systems is among the challenges that suppliers should take up. Mastering the development

[1] This statement is based on the study of the state of practice of software development at Continental

of such systems requires being able to understand, analyze and validate their real time behavior. Automotive software development costs are significantly impacted by wrong design choices made in the early stages of development, but often detected after implementation. Most timing-related failures are detected very late in the development process, during implementation or system integration phases. Timing verification is usually addressed by means of measuring and testing rather than through formal and systematic analysis. For this reason, innovative and complex functionalities are not implemented in a cost-efficient way.

The need for defining an approach that permits timing verification throughout the development process, starting from the early phases of design, is thus obvious

Such an approach would enable early prediction of system timing behavior and allow potential weak points in design to be corrected as early as possible.
To conclude, in automotive software development, there is an obvious need today for development approaches that allow:

- Mastering system complexity
- Reducing software development time and cost
- Defining seamless development activity supported by a seamless tool chain
- Ensuring system dependability, especially timing correctness through verification and validation.

During the last decade, many approaches, methods and techniques have been developed to bring solutions for the abovementioned automotive needs. For example, model based engineering is gaining momentum in the automotive domain, as a means intended for mastering system complexity and assessing system-level tradeoffs geared to achieve higher quality and dependability.

Continental supports the development and the use of several model-based development approaches such as AUTOSAR [18], EAST-ADL [46] and TADL [48]. The directions of our work are chosen with respect to this context.

In the domain of timing verifications, we can talk especially about quantitative techniques (scheduling and performance analysis) where a variety of schedulability tests and tools have been developed as a means to predict early real time system behavior.

In this thesis work, we focus on a key problem of automotive industry which is software timing verification. After studying available approaches that attempted to bring answers to automotive needs in the next sections, we present, in the second part of this manuscript, a methodology enabling the integration of timing verification in a model-based development process.

2. Related Work: Model-based Approaches & Timing Verification

2.1. Model-based Approaches

2.1.1. Basics of Modeling Languages

As engineers work with many different kinds of models, it is important to understand which models are dealt with in this thesis. Therefore, few definitions are given to provide a basis to understand the rest of the thesis.[13]

Models and Metamodels

Models, as conceived in engineering, are representations of reality. The aim of the engineering modeling process is to make our world measurable, calculable, predictable, and thus more manageable. Computer models are computerized abstractions, data structures, or simulations of, not only real systems or phenomena, but also of fictional objects, set-theoretic structures and mathematical representations.

To know the nature of different models used in computer systems, we may identify two relationships. The first relationship, called “represented by”, identifies a representation role of a given modeled object over a model. For instance, a computer program can be “represented by” a set of data flow diagrams. A given model could also represent another model. For example, a mathematical function can be represented by a numerical approximation. The second relationship, called “conforms to”, identifies a dependency of a given model on a modeling language. Thus, we could say that a given data flow diagram representing a piece of programming code conforms to the rules and modeling elements defined, for example, for Gane-Sarson diagrams.[13]

In MBE, the latter relationships receive special attention since domain specific modeling languages are described and prescribed by models. These models are called *metamodels*. A metamodel is yet another abstraction highlighting properties of the model itself. This model is said to conform to its metamodel like a program conforms to the grammar of the programming language in which it is written. This means that a metamodel describes the various kinds of contained model elements and the way they are arranged, related and constrained.

UML Profile Basics

In this thesis work, some notions related to the definition and use of UML profiles are used. We describe here some basic notions related to this issue.

Profiles [14] are the built-in lightweight mechanism that serves to extend Meta Object Facility (MOF)-based languages. More specifically, profiles are used to customize UML for a specific domain or purpose via extension mechanisms that enrich the semantics and syntax of the language. A *stereotype* is the basic feature for UML extension. It can be viewed as the specialization of an existing UML concept, which provides capability for modeling domain-specific concepts or patterns. Stereotypes may have attributes (also called *tags*) and be associated with other stereotypes or existing UML concepts. From a notational viewpoint, stereotypes can give a different graphical symbol for UML model elements. For instance, a class stereotyped as «clock» might use a picture of a clock symbol instead of the ordinary class symbol. Additionally, stereotypes can also be influenced by restrictions expressed in constraints. The standard machine-readable textual language for defining constraints in MOF-based languages is Object Constraint Language (OCL) [15].

2.1.2. Model-Based Development in Automotive Domain

Model-based and component-based approaches are gaining more and more success and popularity in today's automotive software domain. This success is due to the state of practice and the way of proceeding in this domain [6]. In fact, in order to integrate one software unit into the car, a supplier must design, integrate and test against the units of other suppliers. Since the code inside the units (e.g. ECUs) is the intellectual property of the suppliers, the other supplier (or the OEM) often will not get the code of the other units. As a consequence, both have to build up some kind of "black box model" that they code/integrate/test against. The high degree of interaction between OEM and suppliers makes the need for clear interfaces and specifications evident. Models that take into account the static and dynamic

aspects of sub-systems are attractive ways to specify the sub-systems architecture, syntactic interfaces and behavior. Models could help very much in the communication between OEMs and first and second tier suppliers. But, the major advantage expected from model based development is the ability to employ correct-by-construction, but also incremental design processes (which rely extensively on automated transformations and synthesis) and to formalize computer-based correctness analysis. In addition, there are many claims that model-based and component-based approaches using architecture description languages can help improve the overall system quality, foster reuse and evolution, and increase the potential for automatic validation and verification.

The root of model based development is the advent of UML (Unified Modeling Language) [7] as a standard modeling language. However, the general-purpose aspect of UML made its use complicated for specific domains as it requires mastering in detail UML concepts. UML use becomes hence difficult for engineers who are expected to have domain skills and knowledge rather than UML knowledge. As a consequence, this led to the advent of domain specific languages, DSL [16]. Domain-specific languages allowed modeling concepts to map directly domain concepts rather than computer technology concepts.

In automotive domain, several modeling approaches and languages have been developed during the last decade to cope with automotive software development challenges. These approaches give means and concepts to capture the electric/electronic automotive architecture such is the case of the modeling languages EAST-ADL [17] and AUTOSAR [18]. For real time modeling, we cite TADL [11] and also MARTE [10], the OMG language for modeling and analysis of real time systems.

The next section gives a detailed overview of these approaches.

2.1.3. Model Based Approaches Presentation

- **EAST-ADL**

EAST-ADL [46] (Electronic Architecture and Software Technology-Architecture Description language) is intended to capture the electric/electronic architecture of automotive systems at different level of abstraction ranging from feature to implementation level. EAST-ADL has been developed and improved in the context of several research projects. The last available version of EAST-ADL has been developed in the context of the ATESST2 project [17].

EAST-ADL provides a rich set of concepts to model system structure through several levels of abstraction. From one level to another, the structural model of the system is refined by including more precise implementation oriented details. Figure 1 shows an overview of the EAST-ADL abstraction levels. Note that, as shown in this figure, the Implementation level of EAST-ADL is based on AUTOSAR.

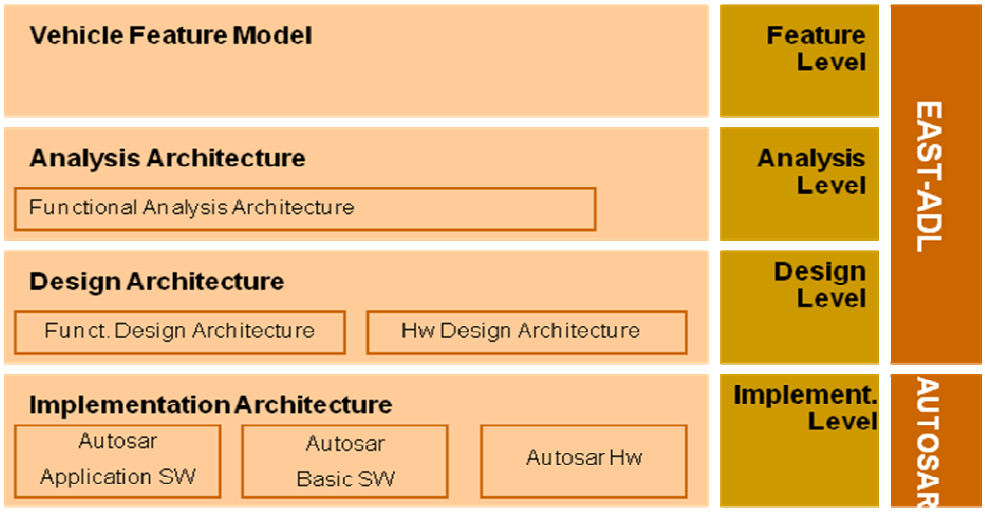

Figure 1 EAST-ADL/AUTOSAR modeling process

Modelling of vehicle electronic systems with EAST-ADL starts with the capture of features at the Feature level, thus providing product line organization and description. These features are then realized at Analysis level by abstract entities, which model the functions and functional devices that interact with the vehicle environment. At the Design level, models are refined by including more realization-oriented details that allow subsequent decomposition/refinement of the functional architecture. The Hardware Design Architecture, which is denoted in parallel, captures the primary hardware entities as abstract elements (e.g. sensor, actuator, power, ECU or electrical wiring including the communication bus) to describe the topology of the system's electronic architecture.

EAST-ADL gives means and concepts to model system functional architecture [47]. Figure 2 gives an overview of the EAST-ADL metamodel for functional modelling. Modelling of functional architecture with EAST-ADL is based on the core concept of "*FunctionType*". A "*FunctionType*" is used to model system functions at both Analysis level ("*AnalysisFunctionType*") and Design level ("*DesignFunctionType*"). An ("*AnalysisFunctionType*" (respectively ("*DesignFunctionType*") can be composed of "*AnalysisFunctionPrototypes*" (respectively "*DesignFunctionPrototypes*") that represent its sub functions. Interaction

between EAST-ADL FunctionTypes is captured through "*FunctionPort*" and "*FunctionConnector*" concepts.

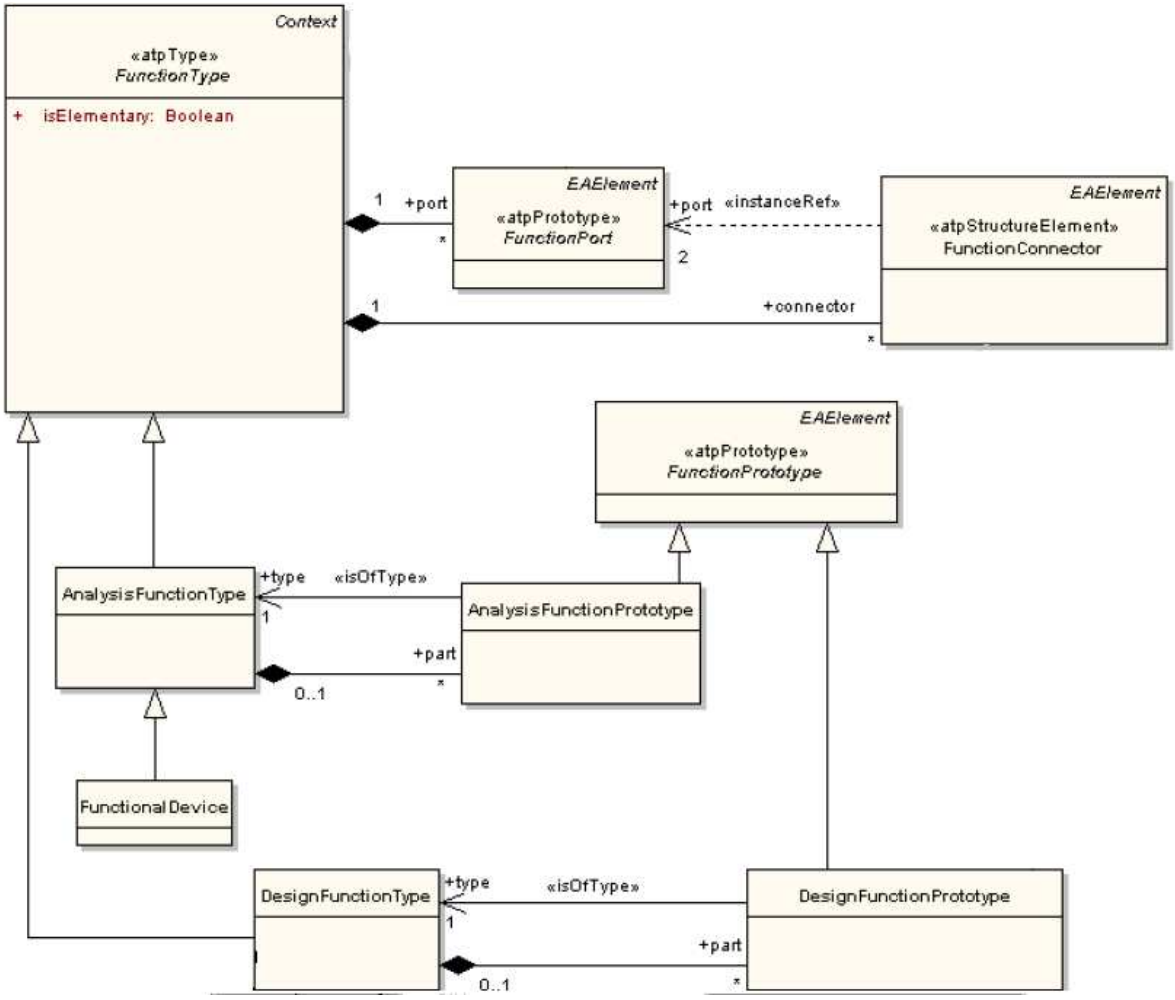

Figure 2 EAST-ADL metamodel for functional modelling [47]

EAST-ADL also provides concepts for abstract hardware modelling. For example sensors and actuators can be captured respectively through the concepts *"sensor"* and *"actuator"* from EAST-ADL. Communication buses can be modelled as *"LogicalBus"*. The concept *"Node"* allows modelling ECUs involved in the system.

For timing modelling, EAST-ADL adopted TADL concepts to annotate architecture models with timing properties and constraints.

- **TADL**

TADL (Timing Augmented Description Language) [48] has been developed in the context of the European research project TIMMO (TIMing MOdeling) [11]. The definition of TADL is based on modelling concepts from EAST-ADL and AUTOSAR by which the

structural definition of the considered system is modelled. The augmentation is done by adding information related to timing and events referring to structural elements [48].

TADL proposes a set of concepts to annotate structural models (function and software) with timing properties and constraints such as maximum delays, repetitions and sampling rates and synchronization constraints [49]. Figure 3 gives an overview of the TADL metamodel. TADL concepts are centred on the concepts of *"Event"* and *"EventChain"*. An *"EventChain"* describes the causal relationship of a set of functionality-dependant events. Every event chain describes a causal relationship between two events. The first is called *"Stimulus"* (e.g. event representing the activation of a function) and the second is called *"Response"* (e.g. event representing the termination of a function). Furthermore, event chains can be hierarchically decomposed into an arbitrary number of sub-chains called *"EventChainSegment"*. TADL timing constraints can be attached to events and event chains to specify e.g. the repetition rate of an event or the maximum latency of an event chain.

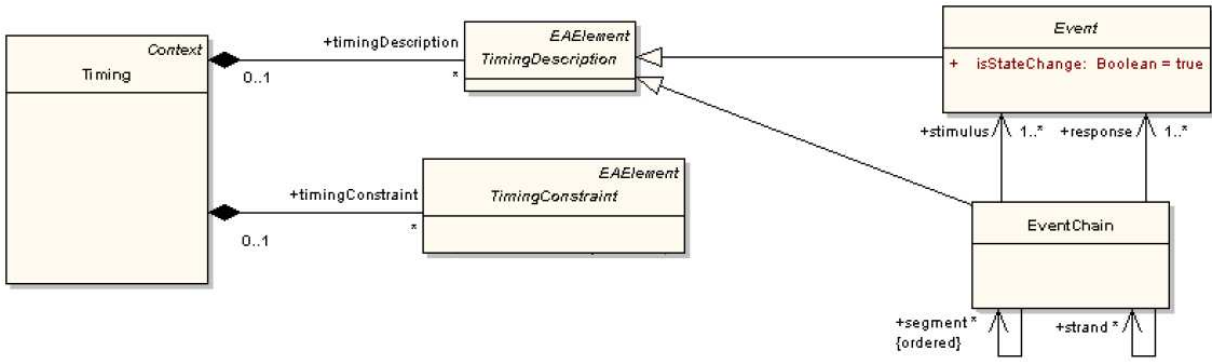

Figure 3 TADL metamodel [49]

- **AUTOSAR**

AUTOSAR (Automotive Open System Architecture) [18] is a standardized architecture for automotive software that is developed by an international consortium of automotive OEMs, Tier-1 suppliers and tool vendors. AUTOSAR offers a software component model and a three layered software architecture divided into application software, runtime environment (RTE), and basic software (e.g., drivers and communication system). Figure 4 shows an overview of AUTOSAR software architecture.

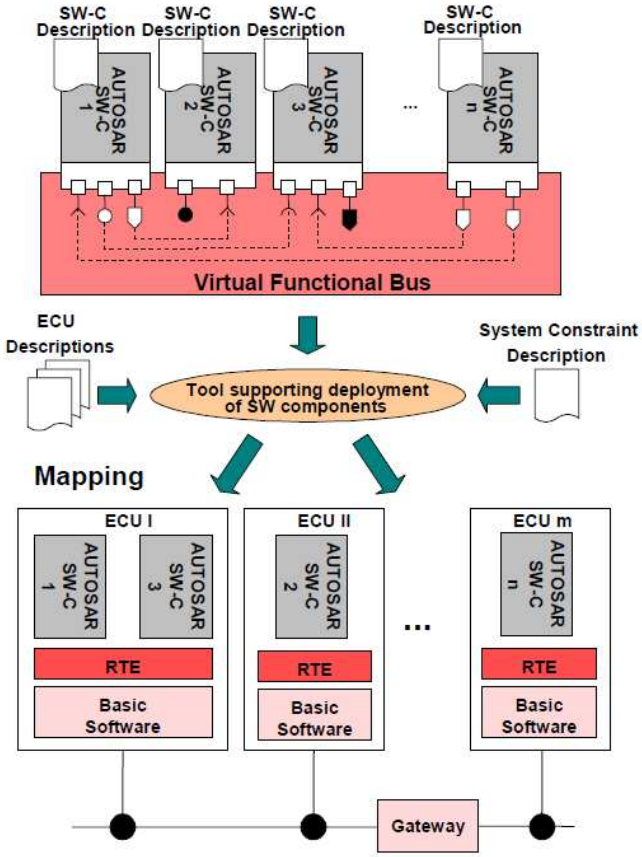

Figure 4 AUTOSAR software architecture from VFB to mapping

AUTOSAR introduces the Virtual Functional Bus (VFB) concept to separate applications from infrastructure. An application consists of interconnected "AUTOSAR Software Components". The VFB (shown in the top part of figure 4) provides standardized communication mechanisms and services for these components. The VFB acts independently from the chosen mapping of these components to the infrastructure of the interconnected ECUs (shown in the bottom part of the figure 4).

The realization of the VFB concept is possible if each AUTOSAR ECU has standardized basic software functionalities and interfaces. Figure 5 shows the layered architecture of an AUTOSAR ECU, which basically identifies an application layer and the AUTOSAR Basic Software (BSW). These parts are linked via the AUTOSAR Runtime Environment (RTE). That means the RTE can be interpreted as the runtime implementation of the VFB on a specific ECU.

The RTE realizes an intermediate layer between the hardware independent application software components and the hardware dependent basic software components.

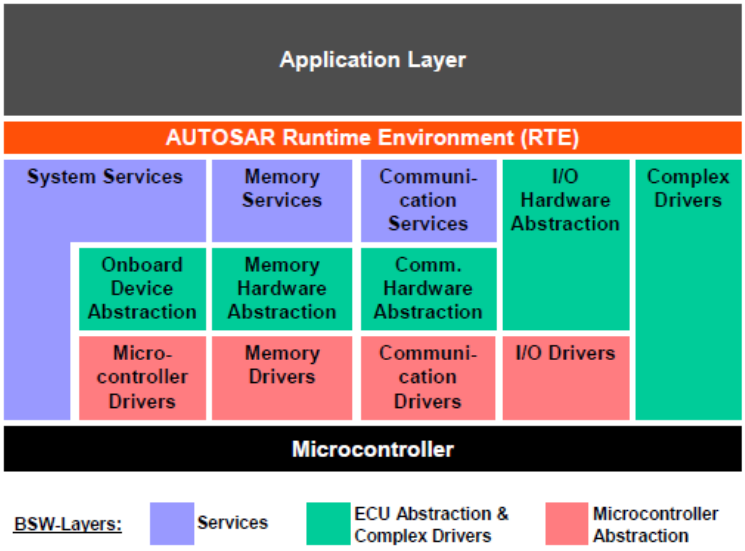

Figure 5 The AUTOSAR ECU layered architecture

The reuse of software components between different vehicle platforms, OEMs and suppliers is one of the major goals of AUTOSAR. Therefore a methodology supporting a distributed, function-driven development process was created [50]. AUTOSAR specifies also compatible software interfaces at application level. However, the functional contents of the application modules and components are different and related to the corporate identity and the desired characteristics of the car manufacturer or its system suppliers.

AUTOSAR has developed a metamodel which precisely defines the concepts used to describe a self-contained AUTOSAR system and a methodology. For example, software models are organized into units called *"SoftwareComponents"* [51]. Those components encapsulate the implementation of the functionality and the behaviour they provide, and simply expose well-defined connection points called ports. In particular, atomic software components are entities that support an implementation and hold behavioural entities called *"RunnableEntity"*. A runnable is an entity that can be executed and scheduled independently from any other runnable entity. AUTOSAR gives also concepts to describe the basic software entities [51] as well as the OS (Operating System) configuration [52], RTE configuration [58] and Hardware topology description [54] (ECU, Bus, etc)

Timing aspect is considered in AUTOSAR through its timing extensions [55] that allow modelling the timing information of the system through concepts that express timing properties and constraints on events and event chains (inspired by the concepts and semantics defined in TADL in order to ease integration of AUTOSAR models with EAST-ADL models).

- **MARTE**

MARTE (Modeling and Analysis of Real Time Embedded Systems) is the OMG standard dedicated for the modeling of real time systems. It provides means and constructs for modeling non functional properties and time concepts [56].

MARTE offers also a dedicated framework for model-based scheduling analysis [13]. This modeling framework provides a rich set of concepts for modeling end-to-end flows, software and hardware resource platform and for allocation of application modules to platform resources.

2.1.4. Model-based Approaches Evaluation

As mentioned previously, the aforementioned model-based approaches were developed to deal with specific automotive challenges (EAST-ADL, TADL, and AUTOSAR) and more generally with real time systems challenges (MARTE). As presented in the first chapter, automotive system development challenges can be categorized in four points:

- Mastering system complexity
- Reducing software development time and cost
- Defining seamless development activity supported by a seamless tool chain
- Ensuring system dependability, especially timing correctness through verification and validation.

Table 1 summarizes the capabilities of the studied model-based approaches against the aforementioned needs.

Mastering system complexity

Looking at the EAST-ADL/AUTOSAR modeling process, we can conclude that there is a good potential to master system complexity. In fact developing automotive systems using these approaches is based on modeling the system architecture starting from abstract functional description until implementation detailed description. Hence, at early design phases, designers focus only on functional aspects abstracting away implementation-related details. In addition, hardware details can be described separately only starting from the Design level.

At Implementation level, using AUTOSAR allows also mastering the system complexity. In fact, AUTOSAR defines different views to enable the description of self-contained software

architecture. In the VFB view, the focus is made on the description of software components and their communication regardless of the platform and the mapping (of software component to ECUs) chosen. In ECU view, the configuration of the ECU may be described through describing the configuration of the RTE and the OS. Finally, in the system view, the focus is made on the system topology by describing e.g. the ECUs and communication buses used by the system. In addition, as shown in figure 5, AUTOSAR offers a layered software architecture giving the possibility to deal separately with the application software, basic software and the hardware. Application software complexity is also reduced as each application software component can be described independently from other software components.

TADL focuses only on modeling the timing aspect of systems by relying on the modeling process offered by EAST-ADL and AUTOSAR.

MARTE also focuses only on modeling the timing aspect of systems without any modeling process support. However, the MARTE scheduling analysis framework allows modeling the different scheduling and timing related features in separate views (application, software/hardware resources, allocations, etc). This allows the designer to focus separately on each aspect without involving details from other views.

Reducing development time and cost

Using the EAST-ADL/AUTOSAR approach, development cost and time can be reduced as there is a good potential for easier reuse of software and hardware components and hence saving the time and cost required for redeveloping them.

In addition, using a model-based approach allows a better representation of system information (using models). Thus, the time required to collect such information (for further use) is significantly reduced during the development process.

Defining seamless development process and tool chain

The modeling process of EAST-ADL/AUTOSAR seems to be interesting as it gives the possibility to design the system architecture starting from abstract functional description until detailed implementation description. However, this process defines only the abstraction levels and the modeling concepts to be used at each level. It does not give any guidance about model refinement from a level to another. In addition it proposes no tool chain

support that allows describing and validating the system architecture along the development process.

As mentioned previously, TADL relies on the modeling process offered by EAST-ADL and AUTOSAR. In addition, a methodology has been defined to describe how the concepts defined by this language can be used at each abstraction level.

MARTE focus only on giving concepts for timing modeling without defining any modeling process or methodology

Enabling timing verification

From a timing verification point of view, the aforementioned model-based approaches attempted to give means for the development of time critical systems. This is mainly ensured through giving concepts for expressing timing properties and constraints on models (TADL, AUTOSAR and MARTE).

However, supporting timing verification by these approaches is limited only to giving such means and concepts. In fact several methodological problems remain unsolved by these approaches, such as:

- How to integrate timing verification during the model-based development process?
- Which timing verification techniques should be used during each development phase?
- How to develop analyzable model to enable a particular timing verification and how to use provided concepts?

To enable model-based timing verification, these approaches should be complemented by a new one that allows answering these questions.

Table 1 Modeling approaches capabilities

	EAST-ADL/AUTOSAR	TADL	MARTE
Master system complexity	Development through abstraction levels (From abstract functional description to detailed implementation). Enable the designer to focus on different aspect at different levels Software is organized into separate software components Software architecture described through different views (VFB, ECU, System) Layered software architecture (application software, basic software, hardware)	Focuses on annotating structural elements with timing information Relies on the means offered by EAST-ADL and AUTOSAR to master complexity	Focus on modeling timing information without giving a modeling process Scheduling analysis models can be organized on separate views (application, platform, allocation, etc)
Reduce development time and cost	Potential for easier reuse of software and hardware components (save redevelopment time and cost) Reduce information collection time through using models	Reduces information collection time through using models	Reduces information collection time through using models
Define seamless development activity	Define only the abstraction levels No guidance for model refinement and transformation No tool chain defined to enable architecture description and validation	Relies on the modeling process of EAST-ADL/AUTOSAR Methodology defined to describe how to use TADL concepts based on EAST-ADL/AUTOSAR structural elements	No modeling process/methodology is defined No tool chain defined
Enable timing verification	Give only concepts to express timing information (timing properties & constraints) No guidance for model-based timing verification (how to integrate timing verification, how to develop analyzable models, which tools to use, how to use results, etc)		

2.2. Timing Verification: Scheduling Analysis

2.2.1. Introduction

Since 1980s, many models, methods and tools were proposed to check if a real time system fulfills its requirements (e.g. Petri nets [19], synchronous languages [20], etc). One of these methods, usually called scheduling analysis is a part of a larger set of quantitative methods, the real time scheduling theory. Based on a schedulability test, scheduling analysis allow verifying the schedulability of a task set. Schedulability tests are based on the calculation of the worst case response time of a task, which is the longest time between the activation of a task and its subsequent completion. Once the worst-case response time is known, the feasibility of a task can be checked by comparing its worst-case response time to its deadline. In next sections, we present the most known results achieved in schedulability analysis in term of schedulability tests and scheduling analysis tools development.

2.2.2. Schedulability Tests: Brief Historical Review

In this section, we present a historical review of the most known results achieved within schedulability test development for fixed-priority monoprocessor systems.

In 1973, Liu and Layland published a paper on the scheduling of periodic tasks that is generally regarded as the foundational and most influential work in fixed priority real time scheduling theory [21]. They made the following assumptions:

- All tasks are periodic
- All tasks are released at the beginning of period and have a deadline equal to their period
- All tasks are independent, i.e., have no resource or precedence relationships
- All tasks have fixed computation time or, at least, an upper bound on their computation time which is less than or equal to their periods
- No task may voluntary suspend itself
- All tasks are fully preemptible
- All overheads are assumed to be null
- There is just one processor.

Based on this model, Liu and Layland gave a sufficient utilization-based condition for the feasibility of a fixed priority task set scheduled with the rate monotonic algorithm (RMA) [21]. They proved that a set of n periodic tasks, each having a computation time C_i and a period T_i is feasible with this algorithm if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$$

Due to the limitations of Liu and Layland test (pessimistic condition, unrealistic task model with deadlines equal to periods, task priorities have to be assigned according to the rate monotonic policy) more complex feasibility tests were developed to address the above limitations. In 1987, Lehoczky et al. [22] gave the first exact schedulability test for the Liu and Layland task model. Concurrently, another group of researchers looked at the problem of determining the worst case response time of a task. Joseph and Pandya [23] and Audsley et al. [24] developed independently an algorithm to compute the worst-case response time R_i of a task τ_i as the least-fixed-point of the following recursive equation:

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

In 1982, Leung [25] considered fixed priority scheduling of a set of tasks with deadlines less than their periods. Lehoczky [26] considered another relaxation of the Liu and Layland model to permit a task to have a deadline greater than its period. The Lehoczky approach uses the notion of "busy-period". A "level *i* busy period" is defined as the maximum time for which a processor executes tasks of priority greater than or equal to the priority of task *i*. Lehoczky shows how the worst-case response time of a task *i* can be found by examining a number of windows, each defined to be the length of the busy period starting at the window, and each window starting at an arrival of task *i*. In the early 1990, Tindell [27] extended the Lehoczky response time analysis providing an exact test for tasks with arbitrary deadlines.

A further relaxation of Liu and Layland task model is to permit tasks to have specified offsets (phasing). Tindell proposed in [28] a test for fixed priority tasks in which task offsets can be taken into account. This test has been later extended by Palencia and Gonzalez to take into account static and dynamic task offsets [29].

Wang and Saksena [30] introduced a feasibility test where they take into account non-preemptible tasks in addition to preemptible ones.

The development of scheduling analysis tools implementing such schedulability tests lies at the very core of scheduling analysis issue. In the next section, we give an overview of currently available scheduling analysis tools.

2.2.3. Scheduling Analysis Tools Presentation

While the number of scheduling analysis tools is constantly increasing, they also vary widely in terms of analysis capabilities and supported features.

- **MAST**

MAST [31] is an open source tool developed by the University of Cantabria in Spain. MAST is still under development and is intended to allow modeling real time applications and performing scheduling analysis for them. The tool offers a suite of scheduling analysis tests, ranging from classic RMA for fixed priority monoprocessor systems to more sophisticated analyses for EDF (Earliest Deadline First) schedulers [21] and distributed systems. In MAST, each real time situation is described through a set of concurrent transactions [41]. A transaction represents the execution of a set of activities triggered by an external event. An activity is an instance of an operation. The output of each activity is an internal event that may in turn activate other activities. Events may have timing requirements associated with them. Activities follow a predecessor/successor relationship with the possibility for an activity to have multiple successors or multiple predecessors. Each activity is bound to a single schedulable resource (task) and a schedulable resource refers to one processing resource. This way, the concept of activity encapsulates the allocation of the operation on a single schedulable resource and the allocation of the schedulable resource on a single processing resource. Table 2 and 3 summarize respectively the most important required inputs for the analysis as well as the output result of MAST.

Table 2 MAST required inputs

Required input information	**Description**
Processing Resource	They represent the processing capacity of a hardware component that executes some of the modeled system activities (Regular Processor) or message transmission (Packet Based Network).
Scheduling Server	They represent schedulable entities in a processing resource (e.g. OS task)
Shared Resource	They represent resource that are shared among different threads or tasks and that must be used on a mutually exclusive way.
Operation	It represent a piece of code or a message
Transaction	A transaction represents a flow of executing activities that are interrelated. A transaction is defined with a list of external events, a list of internal events and their timing requirements and a list of activities
External Event	It represents an event that activates a transaction. It can be e.g. periodic or sporadic.
Activity	It represents an instance of an operation to be executed by a scheduling server. An activity is defined by an input event, and output event, an operation and the scheduling server hosting this operation
Internal Event	It is an event that is generated by an activity. It can trigger the activation of another activity within the same transaction.
Timing Requirement	Represents the timing requirement imposed on the instant of generation of an internal event. It represent a deadline or a maximum jitter on the generation instant of the event.

Table 3 MAST output results

Output result	Description
System/processing resource/transaction slack	If positive, it represents the percentage by which all the execution times of all the operation contained in the global system (or used by the processing resource or the transaction) may be increased while still keeping the system schedulable. If negative it is the percentage by which all these execution times have to be decreased to make the system schedulable.
Worst/best/average Transaction response time	Represents the worst/best/average response time of the transaction (generation of the output event of the transaction) with reference to the external event of the same transaction.
Processing resource utilization	It measures the relation, in percentage, between the time that the processing resource is being used to execute activities and the total elapsed time.
Operation slack	The percentage by which the execution time of that operation may be increased (or decreased) while keeping the system schedulable (or to make the system schedulable)

- **Cheddar**

Cheddar [32] is also open source and is developed and maintained by the University of Brest in France. This tool is designed for checking task temporal constraints of a real time application. Cheddar is based on an analysis framework that includes most of classical real time schedulability tests such as RMA and EDF. In Cheddar, an application is defined by a set of processors, buffers, shared resources, messages and tasks [40]. In the most simple task model, each task periodically performs a treatment. This "periodic" task is defined by three parameters: its deadline, its period and its capacity that represents a bound on the execution time of the job performed by this task. Table 4 and 5 summarize respectively the most important required inputs for the analysis as well as the output result of Cheddar.

Table 4 Cheddar required inputs

Required Input	Description
Processor	They represent the processing capacity of a hardware component that executes some of the modeled tasks
Task	It represents the schedulable entity in the processor. A task is characterized by a priority a computation time, an activation period and a deadline.
Network	It represents e.g. communication buses
Shared resource	They represent resource that are shared among different tasks and that must be used on a mutually exclusive way.
Message	Represent messages that are exchanged between tasks
Buffer	They represent stocking elements for the information exchanged between tasks that read/write in the buffer

Table 5 Cheddar output results

Output result	Description
Task response time	The time between the activation and the termination instants of the task
Processor utilization	It measures the relation, in percentage, between the time that the processing resource is being used to execute activities and the total elapsed time.

- **Rapid-RMA**

Rapid-RMA [33] is a commercial tool developed by Tri-pacific Software Company. Rapid-RMA allows performing analysis based on rate monotonic and deadline monotonic [34] algorithms. A Rapid-RMA system is composed of a set of tasks allocated to hardware resources (CPU, BUS). Each task is characterized by its period, deadline, priority and computation time. Table 6 and 7 summarize respectively the most important required inputs for the analysis as well as the output result of Rapid-RMA.

Table 6 Rapid-RMA required inputs

Required Input	Description
Task	It represents the schedulable entity in the processor. A task is characterized by a priority a computation time an activation period and a deadline.
Node	It represent the hardware resource with processing capacity that executes some of the modeled tasks
Bus	It represents the communication medium used to exchange message between some of the modeled nodes

Table 7 Rapid-RMA output results

Output result	Description
Task completion time	It represents the task response time (time between the activation until the termination of the task)
Processor utilization factor	It measures the relation, in percentage, between the time that the processing resource is being used to execute activities and the total elapsed time.

- **Chronval**

Chronval [35] is a commercial tool produced by the Inchron Company. The tool allows performing scheduling analysis for single and distributed systems. Unlike other scheduling analysis tools that are based on schedulability tests from scheduling theory, Chronval is based on the "real time calculus" technique [57]. The tool allows calculating task response times for an OSEK[2] compliant system. In this tool, a system is seen as a set of tasks. Each task is associated with a source that allows specifying its activation pattern. Task deadlines are specified as requirements that constrain the maximum delay between the task activation and its termination instants. Table 8 and 9 summarize respectively the most important required inputs for the analysis as well as the output result of Chronval.

[2] OSEK : Open Systems and their interfaces for the Electronics in Motor Vehicles

Table 8 Chronval required inputs

Required Input	Description
Task	It represents the schedulable entity in the processor.
Source	It is an element that allow to represent the activation pattern of each task
ECU	It represent a hardware resource with processing capacity
Bus	It represents the communication medium used to exchange message between some of the modeled nodes
Timing requirement	Enables to specify a task deadline or a deadline on a flow formed by several tasks

Table 9 Chronval output results

Output result	Description
Task worst case response time	It represents the worst response time (time between the activation until the termination of the task)
Event spectrum	Shows the variation of the available and the remaining processor capacity for each task

- **SymTA/S**

SymTA/S [36] is a commercial tool developed by the Symtavision Company. The tool is said to be based on schedulability tests that extend previously mentioned classical tests to take into account automotive specific constraints (these constraints will be detailed in the next chapter). It allows performing analysis for both single and distributed systems. In SymTA/S, each real time situation is described through a set of tasks hosting a number of runnables. A runnable represents the execution of a non-preemptive piece of code. Each task in SymTA/S is characterized by an activation pattern, a priority and a deadline. Table 10 and 11 summarize respectively the most important required inputs for the analysis as well as the output result of Chronval.

Table 10 SymTA/S required inputs

Required Input	Description
Task	It represents the schedulable entity in the processor. A task is characterized by a priority, an execution time (if it hosts no runnables) and an activation pattern
Runnable	It represents a non-preemptible executable entity in a task.
ECU	It represent a hardware resource with processing capacity
Bus	It represents the communication medium used to exchange message between some of the modeled nodes
Max response time	It represent the deadline on task or path execution
Path	A path represents a flow of tasks or runnables executing successively and communicating variables

Table 11 SymTA/S output results

Output result	Description
worst case response time	It represents the worst case response time for a task or a path
Processor utilization	It measures the relation, in percentage, between the time that the processing resource is being used to execute activities and the total elapsed time.

2.2.4. Scheduling Analysis Evaluation

Scheduling analysis seems to be a good candidate to perform timing verification for real time systems. Using this technique, there is a good potential to allow detecting timing errors early (only based on a task model) preventing hence costly time-related design mistakes to be detected late.

However, to enable timing verification for automotive systems using such technique, a **need for schedulability tests and tools that fit well automotive needs and constraints is obvious**. In addition, there is currently **no guidance about how to integrate such verification technique during the development process**.

2.3. Conclusion

As shown in the previous sections, several model-based approaches and methods were developed to bring solutions for automotive needs such as mastering system complexity during development, allowing reuse, reducing development time and cost, etc.

To ensure real time system dependability, many scheduling analysis tests and tools were developed as a means for early timing verification.

An obvious lack today in these approaches, is guidance for enabling the integration of timing verification during a seamless model-based development process. In this thesis work, we propose to define an approach for a methodology describing a model-based timing verification process for automotive systems.

3. Work Orientation and Approach Feasibility

3.1. Approach Principle and Feasibility Issues

In our approach, based on existing solutions, we propose to combine some model-based approaches to define our development process. Then, we aim to define a methodology to enable timing verification during each phase of this process. Figure 6 shows an overview of the principle of the targeted process.

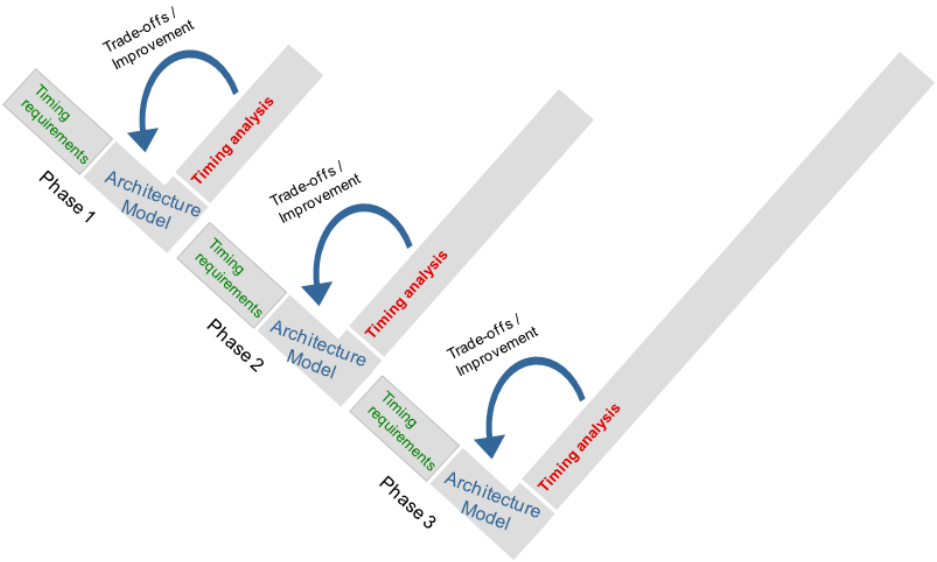

Figure 6 Overview of a model-based timing analysis process

At each phase of the development process, the designer has a set of timing requirements as inputs. S/he designs and models the system architecture (structure, behaviour, etc…) that

should satisfy these requirements. S/he then performs timing analysis to verify whether the proposed architecture does satisfy them. Based on the results of this analysis, the designer determines what improvements are needed in the architecture or what tradeoffs could be made to meet the corresponding timing requirements. The designer may need to perform this activity iteratively until a valid model is obtained. Based on the architecture designed during the current phase, the designer determines the requirements that should be satisfied during the next development phase when refining the system architecture.

To define the model-based development process, we propose to use and adapt some of the available modelling approaches. For timing verification, we suggest to use scheduling analysis as a verification technique in this process.

From a feasibility point of view, to be able to define such model-based scheduling analysis process for automotive systems based on existing solutions, we need to satisfy the two following requirements:

- ***The modelling process should be based on modelling languages that are expressive enough to enable scheduling analysis-aware modelling for automotive applications***

To verify this requirement, we need to evaluate the expressivity of available modelling languages. This will enable us to decide which language (s) to use for our modelling process or how to combine some of these languages to define this process. Based on the defined development process, we can also decide how and in which phase we can integrate scheduling analysis.

- ***Scheduling analysis should be usable to perform timing verification for automotive applications***.

To prove the usability of scheduling analysis for automotive systems, we need to identify a tool that satisfies scheduling analysis needs for automotive systems. This tool should implement schedulability tests that take into account all automotive needs. To identify such tool, we will evaluate the capabilities of available scheduling analysis tools against automotive needs in term of scheduling analysis.

3.2. Modeling Languages Expressivity Evaluation

In this section, we evaluate the expressivity of the aforementioned modeling languages, EAST-ADL/TADL, AUTOSAR and MARTE to enable scheduling-analysis aware

modeling. This evaluation is done against automotive application modeling needs to enable scheduling analysis.

The first paragraph characterizes the required modeling features. Next, we highlight the capabilities and limitations of the studied modeling languages with respect to those requirements.

3.2.1. Modeling Needs for Automotive Applications to Enable Scheduling Analysis

We organize the modeling features needed for scheduling analysis into the four following categories:

Application workload. Modeling languages should enable describing the application workload which represents the processing load of the system. It represents the different operations (functions) executed in the system and contending for use of processing resources and other shared resources. An operation may represent a small segment of code execution as well as the sending of a message through a communication medium. Operations are generally organized in processing flows (set of related operations/functions). To make the analysis possible, modeling languages shall enable specifying the execution /transmission time (worst, best or average) for operations/messages.

Application timing behavior. The application timing behavior represents the timing information of the different operations or processing flows involved in the system under analysis. Timing information contains both timing description (timing properties) and timing constraints. Timing description contains the specification of the triggering of system operations or processing flows (recurrence, activation jitters, etc.). Most available scheduling analysis tools allow analyzing systems with various triggering patterns such as periodic, sporadic, singular, etc. For those activation patterns, it is necessary to specify the period or the minimum inter-arrival time of the triggering events. Timing constraints must be met by the system operations or flows. They are represented basically by operation deadlines, output jitter bounds and end-to-end deadlines.

Resource platform. It represents the concrete architecture and capacity of hardware (e.g., CPU or buses) and software (e.g. OS tasks) resources. For hardware resources such as processors, modeling languages should allow e.g. the description of the scheduler used. For a more accurate analysis, it may be also necessary to specify the processor overheads (e.g. context switch overhead). For software resources such as tasks, it is necessary to specify the

task nature (preemptive, non-preemptive, etc.) as well as its priority. Involved shared resources should also be described.

Allocation. To get an analyzable model, modeling languages should enable specifying the allocation of the operations to software resources (e.g. tasks) and the allocation of software resources to hardware resources (e.g. processors).

3.2.2. Modeling Languages Capabilities

Table 12 contains a summary of the extent to which the surveyed modeling languages cover the features considered. It gives the set of modeling concepts offered by each language to cover the above-mentioned features.

Application workload. Modeling application workload differs significantly in these languages. For example, the *"ADLFunctionType"* and *"ADLFunctionPrototype"* concepts of EAST-ADL allow modeling the functions executed in the system. EAST-ADL gives also means to specify function execution times through the *"ExecutionTimeConstraint"* concept. It allows specifying worst, best or average execution time for each EAST-ADL function [47].

In AUTOSAR, The system workload is described through two categories of elements: runnable entities [51] and basic software module entities [52]. Runnable entities are the smallest code-fragment that are provided by an application software component and are subject to scheduling by the underlying operating system. Runnable entities are specified in the system model as a part of the internal behavior of software components. Basic software entities are also subject to scheduling and contend for use of processing resources. A basic software entity represents the smallest code fragment that can be described for a basic software module or cluster.

In AUTOSAR, it is possible to specify the execution time for both runnable entities and basic software entities as *"ResourceConsumption"* (when describing the corresponding software component implementation or basic software module implementation). The resource consumption element provides information about memory and time consumption for each software component implementation or basic software module implementation. Maximum, minimum and nominal execution times can be specified.

MARTE models the application workload as a set of processing flows called *"End-to-end flows"*. They describe interrelated units of processing work called *"steps"* and which contend for the use of processing resources with other end-to-end flows [56]. MARTE gives the

possibility to specify the execution time of a step through the attribute *"execTime"* that allows specifying a worst or best step execution time.

Application timing behavior. To model application timing behavior elements, EAST-ADL relies on TADL concepts. TADL allows attaching timing description and timing constraints to the events and event chains describing the timing behavior of the system. For example it is possible to describe the triggering pattern of an event (periodic, sporadic, etc) or the maximum latency of an event chain.

MARTE, itself, uses the notion of end-to-end flows to express timing constraints such as an end-to-end deadline imposed on a flow of steps or simply a step deadline. MARTE models the triggering of an end-to-end flow through the element *"WorkloadEvent"* that allows specifying the triggering pattern of each flow (periodic, sporadic, etc).

AUTOSAR allows the modeling of the application timing behavior features through its timing extensions [55]. Timing extensions allow specifying the timing description and the timing constraints of the system. They are used to describe the timing behavior in different views: the virtual functional bus view (VFB timing), the software components view (Swc timing), the basic software module view (Bsw module timing), the system view (system timing) and at the ECU view (ECU timing).

On each level, processing flows are described through the event and event chain concepts (inspired from TADL concepts).

AUTOSAR Timing constraints can be attached to both event chains and events. For an event, timing constraints specify its arrival pattern as well as its occurrence jitter. Supported arrival patterns in AUTOSAR are: periodic, sporadic, burst, concrete and arbitrary. For event chains, it is possible to specify their latencies. A latency timing constraint restricts the time duration between the occurrence of the stimulus and the occurrence of the corresponding response of that chain.

Resource platform. Modeling of software and hardware resource platform is more or less supported by the different languages. EAST-ADL supports modeling of hardware resources through the concepts of *"Node"* (to represent an ECU) and *"LogicalBus"* (to represent communication buses). However, EAST-ADL does not give any means to model software resources such as OS tasks during the Analysis and Design levels. In fact, EAST-ADL relies of AUTOSAR concepts to describe this feature starting from the Implementation level.

MARTE, itself, gives the possibility to model both hardware and software resources. MARTE distinguishes two kinds of processing resources; *"ExecutionHost"*, which includes for example processors and coprocessors, and *"CommunicationHost"*, which includes resources such as networks and buses. Processing resources can be characterized by throughput properties such as processing rate, efficiency properties such as utilization, and overhead properties such as blocking times and clock overhead times. Software resources can be modeled in MARTE as *"SchedulableResource"* or *"CommunicationChannel"*. On one hand, a schedulable resource is a kind of active protected resource that is used to execute steps. In a real time operating system (RTOS), this is the mechanism that represents a unit of concurrent execution, such as a task, a process, or a thread. On the other hand, a communication channel provides concurrency to communication steps.

The *"SharedResource"* concept of MARTE allows modeling the shared resources involved in the system.

AUTOSAR allows specifying the system hardware resources when describing the system topology in the system view [54]. The *"ECUInstance"* concept allows defining the ECUs used in the topology. Communication networks can be specified through the *"CommunicationCluster"* concept that represents the main element to describe the topological connection of communicating ECUs. For each communication cluster, we can define one or more *"PhysicalChannel"* that describe the transmission medium that is used to send and receive information between two communicating ECUs, as well as the protocol used for the communication.

AUTOSAR allows describing the software resources involved in the system when defining the OS configuration [53]. Tasks are specified through the *"OsTask"* concept that represents an OSEK task. Task priority can be specified using the attribute *"OsTaskPriority"*. The attribute *"OsTaskSchedule"* allows specifying whether the task is preemtible or not. Interrupts are supported through the *"OsISR"* concept that represents an OSEK interrupt service routine.

AUTOSAR Shared resources may be specified using the *"OsResource"* concept, used to coordinate the concurrent access of tasks and ISRs to shared resources. The attribute *"OsTaskResourceRef"* of the OS task element allows listing the shared resources accessed by the specific task.

Allocation. EAST-ADL/TADL gives means to describe the allocation of functional entities described at the design level to hardware resources. This is done through the concepts *"FunctionAllocation"* that represent an allocation constraint binding an *"AllocateableElement"* (computation function or communication connector) to an *"AllocationTarget"* (computation or communication hardware resource). However, allocation of functions to OS tasks cannot be described in EAST-ADL (this is due to the fact that EAST-ADL relies on the description of such information at the Implementation level using AUTOSAR). Unlike EAST-ADL, MARTE offers a set of concepts to develop a complete allocation model (allocation of steps to schedulable resources or communication channels and allocation of schedulable resources/communication channels to execution and communication hosts). The MARTE concept *"allocate"* allows associating elements from a logical context, application model elements, to named elements described in a more physical context, execution platform model elements. The *"allocated"* concept allows describing entities that can be allocated to a hosting element.

The allocation of tasks to hardware resources is performed in AUTOSAR during the ECU configuration process. The configuration of a particular ECU used in the system involves the configuration of the OS and of the runtime environment RTE [58]. The OS configuration contains among others the definition of the different OS tasks involved. Hence, this indicates that the defined tasks are allocated to the ECU which is subject to configuration.

The mapping of runnable entities and basic software module entities to OS tasks is part of the RTE configuration. The mapping of runnable entities to OS tasks is based on the mapping of the *"RTEEvent"* that activate those runnable entities. In a similar way, basic software module entities are mapped to OS tasks by mapping the *"BswEvent"* that activate them.

Table 12 Modeling supports for scheduling analysis

Needed modeling Features	Concepts offered by modeling languages		
	EAST-ADL/TADL	**MARTE**	**AUTOSAR**
Application Workload	ADLFunctionType/Prototype. ExecutionTimeConstraint (worst, best, average)	End-to-end flow. Step, ExecTime	Runnable Entity. Bsw Entity. Swc Implementation. Bsw Implementation. Resource Consumption. Measured/estimated execution time
Application Timing Behavior	Event chains. Event chains related to architecture events. Event occurrence constraints. Event chain latency constraints, synchronization constraints	End-to-end flows deadlines, step deadlines, triggering workload event, workload event arrival pattern	Event chains. Events. Event activation constraints. Event chain latency constraints, synchronization constraints
Resource Platform	Hardware resources: Node, Logical bus No software resource description	Hardware resource: execution host, communication host Software resource: schedulable resource, communication channel Shared resource	Hardware resource: ECU instance, communication cluster, physical channel Software resource: OS task, Os ISR, Os resource
Allocation	means for linking function prototypes to hardware entities at design level No means to describe allocation of functions to software resources	Allocation of steps to schedulable resources/ communication channels Allocation of schedulable resources/communication channels to execution/communication hosts concepts: allocate, allocated	Allocation of software resources to hardware resources (OS configuration mechanism) Allocation of runnable entities and basic software entities to software resources (RTE configuration mechanism)

3.3. Scheduling Analysis Tools Evaluation

To prove the usability of scheduling analysis to perform timing verification for automotive applications, we propose to evaluate the capabilities of available scheduling analysis tools to select most convenient tool(s) for our process.

The first section characterizes the required analysis features. Next, we highlight the capabilities and limitations of the studied tools with respect to those requirements.

3.3.1. Scheduling Analysis Needs for Automotive Applications

This section characterizes the architecture of automotive applications. Such characterization suffices for the purpose of this part, which is to identify the timing analysis needs of automotive systems and hence the requirements that should be met by analysis tools. It serves, finally, to provide an informal, comparative review of capabilities provided by the selected tools. For a better understanding, we will assign an identifier to each requirement that we denote REQx where x is the requirement number. Table 13 summarizes the characterization of the identified requirements.

Today's automotive systems have evolved constantly and now offer even more challenging features that can be summed up as follows:

Limited hardware resources. Today, CPU load, has become day-to-day issue and is the very basis for the design of automotive systems. For these reasons, scheduling analysis is required to determine, or at least estimate, the processor performance needed for a given design. Hence, *Analysis tools should have techniques to determine the processor utilization [REQ1].*

Timing constraints. In addition to limited hardware resources, automotive applications must deal with many kinds of timing constraints. These may concern task or function deadlines or maximum jitters on task activation instants. Automotive tasks may have hard deadlines (e.g. for safety functions) or soft deadlines (for body comfort functions). Moreover, these tasks may have deadlines that are less, equal or greater than their periods.

In addition, the end-to-end delay after data is read by a sensor and the output generated from it and passed to an actuator (known as "data age") is crucial to control model stability. Scheduling analysis is hence needed to verify if those constraints are met or not. To enable this verification, scheduling analysis tools have to meet certain requirements that we summarize as follows:

When describing the system under analysis

- *Analysis tools should allow specifying task or function deadlines [REQ2]*
- *Analysis tools should allow specifying jitters related to the function or task activation instants [REQ3]*

- *Analysis tools should allow specifying end-to-end timing constraints [REQ4]*

(An end-to-end timing constraint is a deadline imposed on the delay of an end-to-end flow formed by executing steps in the system)

When analyzing the system

- *Analysis tools should allow analyzing tasks with deadlines that are less, equal or greater than their periods [REQ5]*
- *Analysis tools should have techniques to verify whether end-to-end constraints are respected [REQ6]*

Heterogeneous activation pattern. In automotive task model, tasks can be time triggered or event triggered. Event triggered tasks are activated by the arrival of events that can be periodic, sporadic or singular (arrives only once). Time triggered tasks are periodic tasks that are activated at predetermined points in time. In automotive, there are two kinds of periodic tasks, timing tasks and engine-synchronous tasks. Timing tasks have timing recurrences (e.g. 1ms, 10ms, etc) (they are simply classic periodic tasks). Engine-synchronous task are activated by the arrival of events related to the engine-running. The recurrences of these events are expressed in engine angle degree rather than time (e.g. 2°crank). In fact these recurrences depend on the Camshaft and Crankshaft positions that vary with the engine speed (The camshaft wheel is the element of the engine that allows the opening and the closure of intake and exhaust valves. The crankshaft wheel is the part of the engine that translates reciprocating linear piston motion into rotation). Hence, expressing the period of such tasks in time depends also on the engine speed. For instance, for a 6 cylinder system, a task that should be activated each 120°crank has got a recurrence of 3.3ms at 6000rpm and 13.33ms at 1500rpm (engine-synchronous tasks are hence periodic tasks in the angular base and aperiodic tasks in the classic time base). This variable aspect of recurrence should be taken into account by scheduling analysis tools:

- *Analysis tools should allow specifying periodic, sporadic and singular activation [REQ7]*
- *Analysis tools should allow describing and analyzing system with engine-synchronous tasks [REQ8]*

Distributed architecture. In conventional automotive system design, a function may be distributed over many ECUs (Electronic Control Units) into a network that may even use multiple protocols. Most used protocols are CAN, LIN and FlexRay [37]. For such

distributed functions, it is important to guarantee end-to-end response times. In addition, in such complex architectures, optimization of network resource consumption and message scheduling requires knowledge of the impact of network properties such as network overheads and driver overheads, and of different communication protocols. Consequently, scheduling analysis tools have to satisfy the following requirements:

- *Analysis tools should allow easy description of distributed systems with multiple ECUs and communication buses [REQ9]*
- *Analysis tools should have techniques to analyze multiprocessor systems [REQ10]*
- *Analysis tools should have techniques for CAN, LIN and FlexRay [REQ11]*
- *Analysis tools should allow taking into account processor overheads (basically context switch overhead) and network overhead (network driver overheads) [REQ12]*

Task concurrency and dependency. In automotive systems, tasks may be dependent. This dependency results basically from task chaining which means that a task is activated by another task. Automotive tasks may also have activation offsets. For engine synchronous tasks, their offsets vary also with the engine speed.

Concerning the concurrency issue, in automotive design, although tasks are concurrent, different tasks may have the same priority level. As most automotive applications are based on OSEK [38], these tasks are scheduled using the FIFO algorithm (First In First out) as a second scheduling protocol. Moreover, automotive tasks are of three kinds: preemptive tasks, cooperative tasks and interrupts. The execution of cooperative tasks can be interrupted by higher priority cooperative tasks only at predefined points called *schedule points*. Figure 7, shows an example of a system with preemptive and cooperative tasks. Task T1 is a preemptive task having the highest priority, task T2 and T3 are both cooperative tasks, T2 has got higher priority than T3. As the figure shows, T2 waits until the schedule point of T3 to start executing, while T1, being preemptive, interrupts T2 before its schedule point.

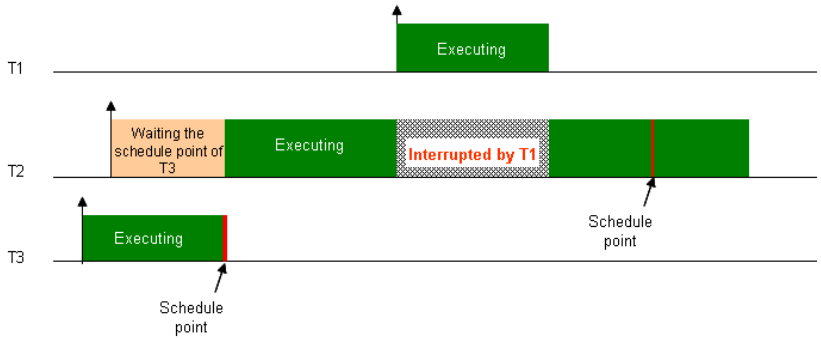

Figure 7 Preemptive and Cooperative Tasks

To enable an accurate scheduling analysis, analysis tools have to support the description and analysis of such a task model and hence:

- *Analysis tools should allow describing task dependency resulting from task chaining [REQ13]*
- *Analysis tools should allow using FIFO as second scheduling algorithm for tasks having the same priority level [REQ14]*
- *Analysis tools should allow specifying preemptive, cooperative tasks and interrupts [REQ15]*
- *Analysis tools should allow describing and analyzing systems with constant and variable offsets [REQ16]*

Table 13: Requirements on scheduling analysis tools

Requirement	Description
REQ1	Analysis tools should have techniques to determine the processor utilization
REQ2	Analysis tools should allow specifying task or function deadlines
REQ3	Analysis tools should allow specifying jitters related to the function or task activation instants
REQ4	Analysis tools should allow specifying end-to-end timing constraints
REQ5	Analysis tools should allow analyzing tasks with deadlines that are less, equal or greater than their periods
REQ6	Analysis tools should have techniques to verify if end-to-end constraints are respected
REQ7	Analysis tools should allow specifying periodic, sporadic and singular activation
REQ8	Analysis tools should allow describing and analyzing system with engine-synchronous tasks
REQ9	Analysis tools should allow easy description of distributed systems with multiple ECUs and communication buses
REQ10	Analysis tools should have techniques to analyze multiprocessor systems
REQ11	Analysis tools should have techniques for CAN, LIN and FlexRay
REQ12	Analysis tools should allow taking into account processor overheads (basically context switch overhead) and network overhead (network driver overheads)
REQ13	Analysis tools should allow describing task dependency resulting from task chaining
REQ14	Analysis tools should allow using FIFO as second scheduling algorithm for tasks having the same priority level
REQ15	Analysis tools should allow specifying preemptive, cooperative tasks and interrupts
REQ16	Analysis tools should allow describing and analyzing systems with constant and variable offsets

3.3.2. Scheduling Analysis Tools Capabilities

In this section, we consider the aforementioned scheduling analysis tools, MAST, Cheddar, Rapid-RMA, Chronval and SymTA/S. Table 14 summarizes the coverage provided by these tools with regard to the requirements described above. Full explanations are given in subsequent paragraphs.

REQ1: MAST allows the designer evaluating his processor or network performance by calculating either its global utilization or a more limited scenario such as utilization by context and interrupt switch activities. The tool likewise enables him to see to what extent operations executed on the processing resource are schedulable. This entails calculation of processor or network slack, i.e. the percentage increase in execution times that is compatible with keeping the system schedulable.

Cheddar allows performing certain feasibility tests based on calculation of the processor utilization factor [21]. Depending on the resulting factor, the tool tells the user whether a task set will be schedulable or not. Cheddar does not calculate processor or network slack.

Rapid-RMA allows calculating the processor utilization for periodic and aperiodic tasks. In addition to quantitative results, it displays also a graphic showing the utilization of the processor by each kind of tasks and the unused percentage of the processor capacity.

Chronval does not calculate a value showing the global utilization of the processor by the different tasks. However, through a graph called "event spectrum viewer", it is possible to visualize the variation of the available and the remaining processor capacity for each task.

For each processor, SymTA/S calculates its global utilization but also elementary utilization for each task. This kind of result is interesting, it allows the designer identifying the tasks having the biggest load and hence the possible changes in case of overloaded processor.

REQ2: MAST defines the concept of operation that represents a piece of code or the sending of a message. The tool allows specifying timing constraints on operations through the concept of timing requirement. The latter can be specified on the output event of an activity (represents the execution of an operation). A timing requirement may be a deadline or a maximum jitter imposed on the generation instant of the output event of an activity. MAST supports both hard and soft deadlines. Cheddar and Rapid-RMA support this feature differently by allowing specification of deadlines on tasks themselves.

To describe task deadlines, Chronval allows assigning a timing requirement to a task. This requirement allows specifying a bound on the delay between the activation event of the task and its termination event. SymTA/S, itself, allows specifying a max response time for each task.

REQ3: MAST defines the concept of external event that serves to trigger the execution of a flow of activities (transaction). The tool allows specifying a maximum jitter on the arrival time of an external event but this is only possible for periodic events. Cheddar supports this feature by allowing specifying a maximum lateness on task wake up time through the concept jitter.

Rapid-RMA does not allow specifying jitter bounds for the activation instants of aperiodic tasks

To describe the activation of a task, Chronval uses the concept of source. A source is an element that is connected to a task to describe its activation patterns such as its period (or minimum inter-arrival time) and its activation jitter. This feature is also supported by SymTA/S that allows specifying a jitter value for periodic, sporadic and pattern tasks [59].

REQ4: MAST meets this requirement by allowing the specification of a deadline on the generation instant of the output event of an execution flow of activities (transaction) with reference to the external triggering event. Contrarily to MAST, specifying end-to-end constraints is supported neither by Cheddar nor by Rapid-RMA.

In Chronval, specifying end-to-end timing constraints is also supported through the concept of requirement. To specify an end-to-end constraint on a flow of tasks, one can specify a requirement between the activation event of the first task and the termination event of the last task in the flow.

SymTA/S uses a similar approach, specifying end-to-end timing constraints is supported through the concept of path in SymTA/S. A path represents a flow of tasks or runnables executing successively and communicating variables. SymTA/S gives the possibility to specify a max response time for the path.

REQ5: Except Rapid-RMA that requires task deadlines to be equal to task periods, all of the other tools allow specifying and analyzing tasks with deadlines that are less, equal or greater than their periods.

REQ6: MAST allows calculating the response time of the output event of a transaction and compares this with end-to-end constraints imposed on the system. Cheddar allows calculating end-to-end response times based on the holistic approach defined by Tindell for distributed systems in [39]. These end-to-end response times include message transmission delay and buffer memorization delay.

Rapid-RMA, itself, has no means to verify end-to-end constraints involving more than one task.

As for deadlines, Chronval calculates end-to-end response times and compares them with end-to-end requirements. SymTA/S uses the same approach by calculating the response time for each path and comparing it with path max response time. In SymTA/S, a path response time is the sum of the response times of the tasks involved in the path and the sampling delays.

REQ7: Triggering patterns are captured in MAST through external events that activate transaction execution. MAST external events may be periodic, singular, sporadic, unbounded or bursty.

In Cheddar, there is no distinction between a task and its triggering. Cheddar does not, in fact, consider triggering events but rather focus on tasks themselves. In Cheddar tasks may be periodic, aperiodic, sporadic, etc [40]. Cheddar also makes it possible for the designer to specify new activation patterns (User-defined activation pattern) without modifying the implementation of the tool [40]. This same facility is provided by MAST, but the tool implementation should be modified (As it is an open-source tool)

Rapid-RMA and SymTA/S use the same approach as Cheddar, allowing hence specifying the activation pattern of a task without having recourse to event concept. Rapid-RMA allows specifying periodic and aperiodic tasks. SymTA/S, itself, allows describing sporadic and periodic tasks that may have activation jitters. Singular tasks are described through the aperiodic pattern in Rapid-RMA; this kind of tasks cannot be described in SymTA/S.

Chronval uses the notion of source to describe the activation of a task. Chronval sources allow describing periodic, sporadic and singular tasks.

REQ8: As mentioned previously, engine-synchronous task periods and deadlines vary depending on the engine speed. This means that for a fixed engine speed, these tasks can be considered as purely periodic tasks with constant deadlines. Hence to be able to analyze a

system with such kind of tasks, using the studied tools, we need to perform the analysis for a fixed engine speed. This is due to the fact that all of the studied tools consider only one timing base in which task parameter values can be expressed (period, deadlines, etc). However this is very limiting due to the fact that a worst-case response time determined for a particular speed is not necessarily valid for other engine speeds. To solve this problem, SymTA/S gives the possibility to perform analysis for variable engine speed. This is done based on a scripting support that allows expressing the parameters of these tasks as a function of engine speed and then incrementing the engine speed and performing the analysis for each speed. Compared with other tools, this approach is quite interesting as it allows determining worst case response times for different engine speeds. However, a special care should be taken when choosing the incrementation step of the speed. In fact a large step enables a fast analysis but many transitory speeds are missed. Choosing a small incrementation step allows covering more transitory speeds but the analysis takes much more time.

REQ9 & REQ10: All of the studied tools allow describing and analyzing distributed systems. In addition, all of them implement scheduling techniques for multiprocessor systems.

MAST enables description of the networks involved in a system being analyzed through the concept of Packet Based Network. It represents a network that uses some kind of real time protocol based on non-preemptible packets for sending messages [41]. MAST supports the following transmission kinds: Simplex, Half duplex and Full duplex (see [41] for more details about these transmission kinds).

Cheddar is designed to perform scheduling simulation of message-sharing applications distributed on several processors. It allows specifying networks with three kinds of communication protocols (bounded delay, jitter delay and parametric delay) [32].

Rapid-RMA allows describing and analyzing distributed systems through the multiple node analysis. The tool allows describing the buses used for the communication in the system under analysis as well as the time overheads associated to the access to these communication media. However, the tool gives no means to describe the bus properties such as the communication protocol used.

SymTA/S and Chronval also allow describing and analyzing systems with multiple ECUs and communication buses.

REQ11: Except SymTA/S which allows describing and analyzing systems with CAN and Flexray buses, none of the other tools have analysis techniques dedicated for these buses. LIN bus is not supported by any of the studied tools.

REQ12: MAST has means for independent description of overheads for both processor and network. In fact, it allows specifying either worst, best or average context switch overhead when describing system processors. For networks, MAST allows specifying packet overheads that represent the overheads associated with sending each packet because of the protocol messages or headers that need to be sent before or after each packet.

Cheddar and SymTA/S, on the other hand, allow specifying the context switch overhead value associated to the activation of each task, but no network overheads may be described in these tools.

Rapid-RMA allows taking into account time overheads associated with the acquisition or the release of a resource such as a memory or a bus. For processors, the tool allows specifying the context switch rate, which is the amount of time the CPU takes to change from executing one task to another.

Chronval, itself, does not give any means to describe processor or network overheads.

REQ13: Unlike MAST and Rapid-RMA, SymTA/S, Cheddar and Chronval allow specifying task chaining. In Chronval, each task has got a "connection" field. In this field, it is possible to describe an activation source for the task or to specify that this task is activated by another task. For each SymTA/S task, it is possible to describe a "caller" that represents another task that activates it.

REQ14: All of the studied tools allow specifying tasks with the same priority. However, only SymTA/S and Cheddar give the possibility to use FIFO as second scheduling algorithm for these tasks.

REQ15: Systems having preemptive and cooperative tasks as well as interrupts can be described and analyzed by SymTA/S, Chronval and rapid-RMA. All of them allow describing non-preemptible sections for each cooperative task. This feature is supported neither by MAST nor by Cheddar as both of them consider only a fully preemptive system that may have interrupts.

REQ16: All studied tools allow describing and analyzing tasks with static offsets. Variable offsets are not supported by these tools. However, for engine-synchronous task offsets which

depend on the engine speed, the scripting support of SymTA/S can be used to analyze systems having such offsets.

Table 14 Scheduling analysis tools capabilities

Requirements	**The requirement is satisfied by the tool**				
	MAST	Cheddar	Rapid-RMA	SymTA/S	Chronval
REQ1	Yes	Yes	Yes	Yes	No
REQ2	Yes	Yes	Yes	Yes	Yes
REQ3	Yes	Yes	No	Yes	Yes
REQ4	Yes	No	No	Yes	Yes
REQ5	Yes	Yes	No	Yes	Yes
REQ7	Yes	Yes	Yes	No (no singular activation)	Yes
REQ8	No	No	No	Yes	No
REQ9	Yes	Yes	Yes	Yes	Yes
REQ10	Yes	Yes	Yes	Yes	Yes
REQ11	No	No	No	Yes (for CAN and Flexray)	No
REQ12	Yes	Yes	Yes	Yes	No
REQ13	No	Yes	No	Yes	Yes
REQ14	No	Yes	No	Yes	No
REQ15	No	No	Yes	Yes	Yes
REQ16	No (no variable offsets)	No (no variable offsets)	No (no variable offsets)	Yes	No (no variable offsets)
Covered features/uncovered features	10/6	11/5	8/8	15/1	10/6

3.4. Conclusion and Approach Directions

In our approach, we propose to align our model-based development process with the EAST-ADL/AUTOSAR modeling process. This choice is due to fact that this process gives a good support to model automotive architecture from an abstract functional description until a detailed implementation. In addition this choice is motivated by the fact that Continental supports the use of EAST-ADL and AUTOSAR (as mentioned at the end of the first section

of this part). However, the EAST-ADL/AUTOSAR process presents only the abstraction levels and modeling concepts that can be used at each level. It gives guidance neither about how models can be developed (e.g. which modeling diagrams to use) nor about how these models can be refined from one level to another. For these reasons, our methodology needs to enrich this process with guidance for model development, transformation and refinement as well as the views to be developed at each level to obtain a complete analyzable model.

Based on the evaluation of the expressivity of the different modeling languages (see section 3.2.2), we can conclude that MARTE and AUTOSAR are the most expressive languages to enable scheduling analysis-aware modeling. In fact, both of them give all the necessary means to develop an analyzable model and perform scheduling analysis. Hence, there are two possibilities to integrate scheduling analysis in the chosen EAST-ADL/AUTOSAR process:

1) The first possibility is to perform scheduling analysis at the design level of the process by completing EAST-ADL models with MARTE concepts to get an analyzable model and hence perform complete scheduling analysis as described in [45]. In [45], we show how to complete EAST-ADL models using MARTE concepts to describe software (e.g. OS tasks) and hardware resources (e.g. ECUs) as well as the allocation of functions to OS tasks and the allocation of OS tasks to hardware resources. Based on the developed model, we show how to perform scheduling analysis using the scheduling analysis tool MAST. This is done based on an automatic transformation of EAST-ADL/MARTE models to a MAST model as described in [60].
2) The second possibility is to perform scheduling analysis at the implementation level based only on AUTOSAR concepts (as AUTOSAR gives all the necessary information to develop an analyzable model).

As we aim at defining a seamless and coherent timing analysis process, it is not possible to perform scheduling analysis both at the design and the implementation level. To avoid the redundancy of timing analysis between the design and the implementation level, we decided to perform scheduling analysis only at the implementation level (based on AUTOSAR concepts) and to complete this by a more "abstract" timing analysis at the analysis and design levels (this "abstract" analysis will be described with more details in the next

paragraphs). This choice is also motivated by the fact that Continental supports the use of AUTOSAR.

Starting timing verification at the implementation level is, however, quite late. In our approach, we suggest then to start earlier, at the analysis level. We believe that this is the earliest level against which timing verification can be performed. In fact, the EAST-ADL Feature level is rather dedicated to capture vehicle features with product line description, without the details needed to perform any relevant timing analysis (e.g., it offers no descriptions of the internal architecture of the vehicle functions). Our model-based timing analysis process thus consists of the following three usual phases: Analysis phase, Design phase and Implementation phase. The timing verification performed during analysis and design phases is a sort of "abstract analysis" that sets for a preparatory work for the scheduling analysis activity that will be performed during the implementation phase. We call thus the verification activity during these phases (analysis and design phases) "timing analysis" rather than scheduling analysis.

During analysis and design phases, we propose to determine time budgets to be allocated to the system under design and to its sub-functions to ensure compliance with the input timing requirements during each phase. To determine such budgets, we propose to complement the EAST-ADL structural views with timing views that we will annotate using TADL concepts. The time budgets determined during each phase will be used as input for the timing analysis performed during the next phase.

As we start capturing the hardware entities at the design level, we propose also to start evaluating hardware resource capacities at this level. To do so, an "abstract" model for allocation of functional elements to hardware resources should be developed (by abstracting software resources such as OS tasks). As this allocation model aims to represent only the allocation of functional elements to hardware resources (without involving OS tasks), this model can be developed using only EAST-ADL concepts for allocation modeling. This way, based on the EAST-ADL allocation model, a scheduling analysis tool can be used to evaluate the load of each processing resource by calculating its utilization. This analysis can also be performed based on a model that combines EAST-ADL concepts for functional and hardware modeling and MARTE concepts for allocation modeling similarly to the approach described in [45]. The advantage of the second alternative is the possibility to use the automatic transformations that are already implemented [60] to transform MARTE models to a scheduling analysis tool model. In our methodology, we choose this latter alternative

(combining EAST-DL and MARTE concepts) as using MARTE automatic transformations would enable us reducing the time required to perform the needed timing analysis (processor load evaluation)

From a tool support point of view, the evaluation work performed for the scheduling analysis tools show that many of the automotive needs are met by some of the evaluated tools (which proves the usability of scheduling analysis to perform timing verification for automotive systems). However, SymTA/S seems to be the most complete and the most convenient for automotive systems. Hence, in our approach, we propose to use this tool to perform scheduling analysis during the implementation phase. Nevertheless, other tools such as Cheddar or MAST are used in our methodology to evaluate hardware resource capacities during the design phase based on the calculation of the processor utilization.

Part II: Methodology for Model-Based Timing Analysis Process

In this part, we present a methodology that describes a model-based timing analysis process. This process is defined based on available EAST-ADL/AUTOSAR modeling process presented in the previous part.

This part is composed of four chapters. The first chapter gives a general overview of the model-based timing analysis process. The second, third and fourth chapters detail the modeling and timing analysis activities performed respectively during the analysis, design and implementation phases.

1. Methodology Overview & Process Phases

The methodology presented in this part describes a model-based timing analysis process. The methodology defines both the modelling process and the timing analysis process.

1. The modelling process describes the models that should be developed in each phase to enable a particular timing analysis. It shows how these models are refined from one phase to another and how timing models are derived from architecture models. It also describes the modelling views needed for every analysis type.
2. The timing analysis process describes the kind of analysis to be performed during each phase and how analysis results can be used for the next phase. It also indicates which tool can be used to perform each kind of analysis.

Throughout the remainder of this part, the vehicle function developed using the proposed methodology is referred to as the "sub-system" (as it represents a part/sub-system of the vehicle itself).

As already stated previously, our process entails three phases. Each of them comprises, two activities, i.e. development of the analyzable model for the sub-system, and performance of timing analysis based on this model. The next paragraphs give a brief description of these analyzable model development and the timing analysis activities. Figure 8 shows a general overview of the timing analysis and modelling activity for each phase of the process. Chapters 2, 3 and 4 provide more details on the architecture model developed at each stage.

- **Analysis phase**

During this phase, a functional architecture view is developed based on EAST-ADL concepts for functional modelling. This view depicts the sub-system under development in its vehicle environment. Based on this view, a second view called timing view is derived to enable the timing analysis of this phase. The timing analysis performed during this phase aims to verify correct integration of the sub-system into the vehicle in terms of timing compatibility. The designer has a set of vehicle end-to-end requirements that involve the sub-system being designed and the other vehicle functions/sub-systems that interact with it (a detailed explanation of these requirements is given in the next paragraphs). For each vehicle end-to-end requirement, the designer determines a time budget to be allocated to the sub-system, to ensure compliance with this requirement. Each sub-system time budget determined during the analysis phase serves as a constraint for the next phase – design.

- **Design phase**

During this phase, the functional breakdown of the sub-system is modelled by detailing the functional blocks that constitute it. The hardware resources used by the sub-system are also modelled during this phase. The designer performs, hence, two kinds of timing analysis. The first consists in refining the time budgets allocated to the sub-system during the analysis phase. Based on the sub-system time budgets determined at analysis level for each vehicle end-to-end requirement, the designer determines the time budgets to be allocated to each functional block. S/he thus continues complying with vehicle end-to-end requirements as sub-system architecture is refined. Each functional block time budget represents a timing constraint that has to be met during the implementation phase.

The second timing analysis of this phase explores the hardware architecture to identify the best target hardware platform, while suitably allocating functional blocks to hardware resources. Our approach relies on empirical exploration to conduct the analysis. The latter is performed on the basis of a scenario for allocating functional blocks to the chosen ECUs, after evaluation of the utilization of each ECU. Note that during this phase, we do not take into consideration OS tasks but limit analysis to the functional model, the hardware platform and the allocation of functional blocks to ECUs. Based on the obtained ECU utilization values, the designer determines the best allocation scenario. This scenario subsequently serves as a constraint for refining the allocation model in the implementation phase.

- **Implementation phase**

During this phase, a complete model of the software and hardware architecture of the sub-system is developed by further refining the models and the timing results of the design phase. The complete model contains all the information required to perform a complete scheduling analysis (application, hardware and software resources, allocation, etc.).

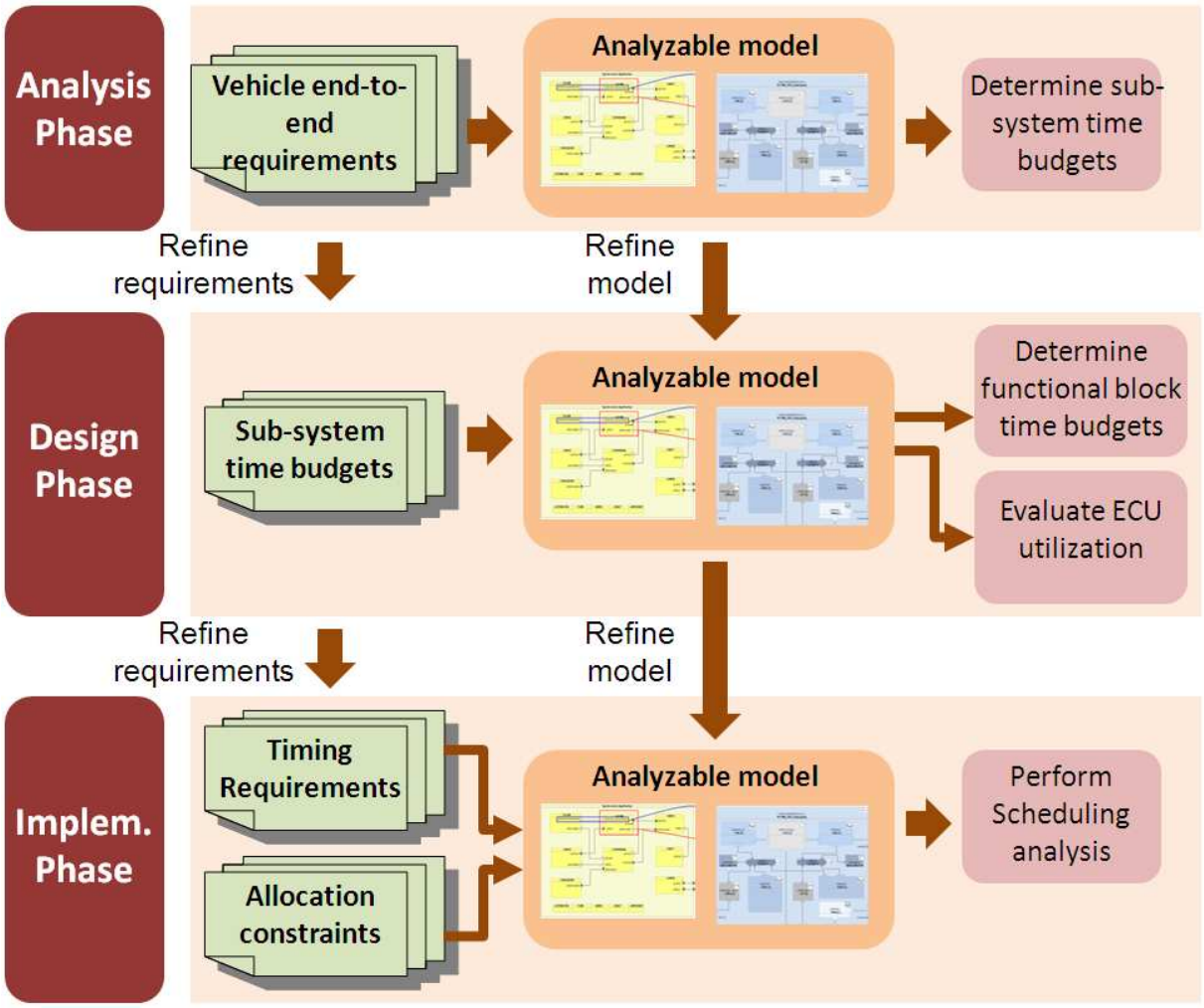

Figure 8 General overview of the model-based timing analysis process

2. Analysis Phase

2.1. Analysis Objectives and required Analyzable Model

2.1.1. Timing Analysis Objectives

The timing analysis of this phase consists in determining a set of time budgets for the sub-system under development. These time budgets are determined with respect to a set of vehicle end-to-end requirements that the designer should respect.

Time budget

A time budget represents a constraint on the response time of the sub-system. It represents a deadline that we allocate to the sub-system to ensure compliance with a vehicle en-to-end requirement.

Vehicle end-to-end requirement

A vehicle end-to-end requirement is a requirement that impose a maximum delay on a flow formed by several vehicle sub-systems including the sub-system under development. To explain more the concept of vehicle end-to-end requirement, let's consider the example of the cruise control sub-system of Figure 9a. The cruise control is used to maintain vehicle speed to a speed set point desired by the driver. Based on driver requests that are acquired through a switch sensor, the cruise control performs the desired action (e.g. calculate speed set point, increase/ decrease set point, etc) and then sends a torque request to the torque set point sub-system to maintain the vehicle speed to the speed set point. The cruise control communicates also with the brake controller sub-system that informs him about the braking pedal status.

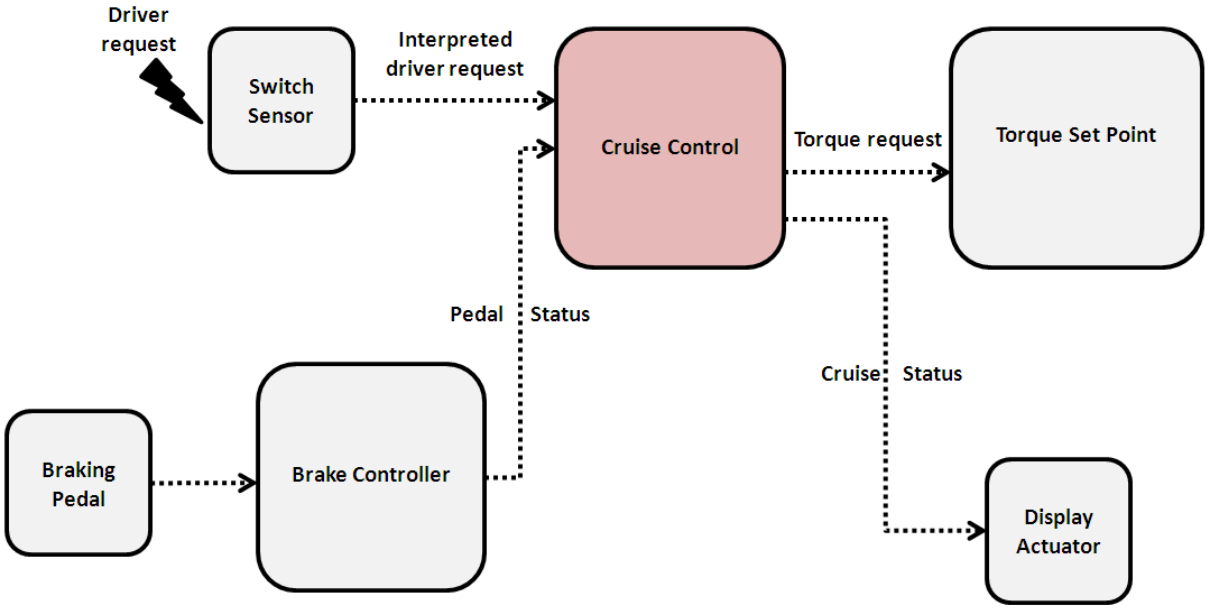

Figure 9a Example of the cruise control

An example of a vehicle end-to-end requirement is the following: "**When the driver depresses the braking pedal, cruise control should be deactivated within 300ms**". This requirement imposes a maximum delay on the execution flow starting from the depressing of the braking pedal until the cruise control sends an output (null torque request) to the torque set point sub-system (cf. figure 9b)

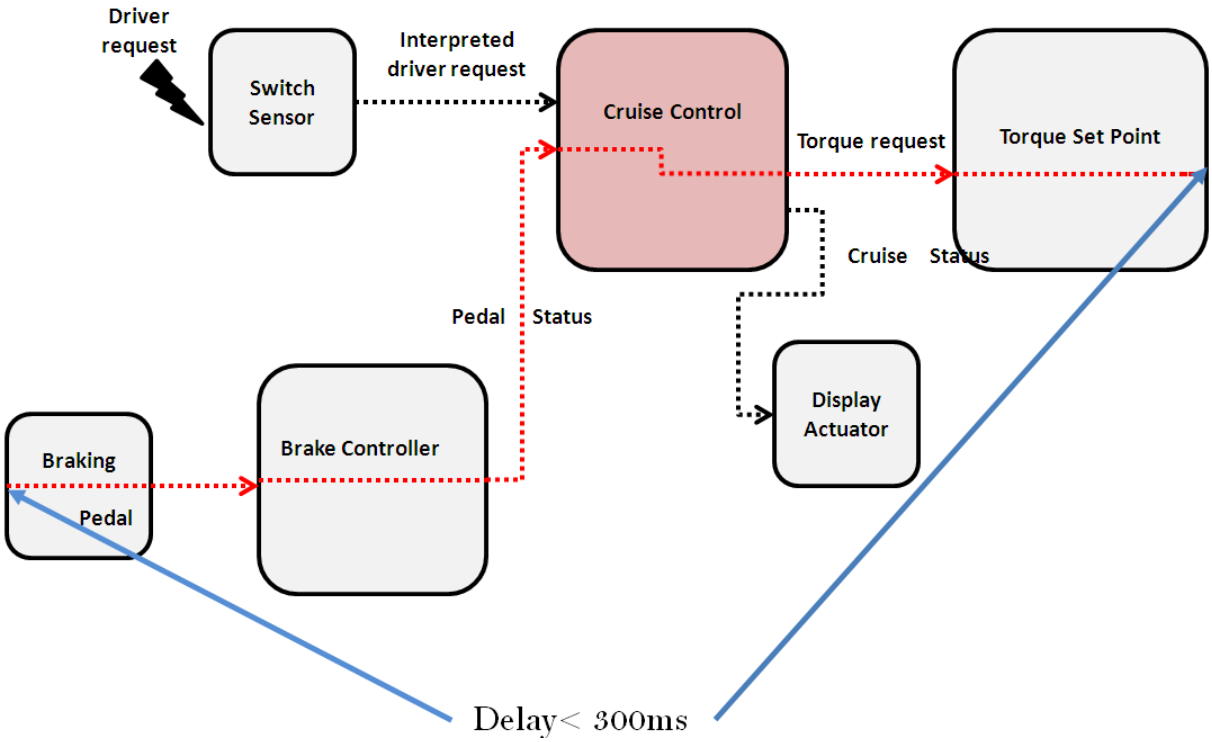

Figure 9b Example of the cruise control

2.1.2. Analyzable Model Minimum Features

To be able to determine the time budgets to be allocated to the sub-system under development, the model developed should contain the minimum information enabling such analysis. We organize this information in two categories: the vehicle functional architecture and the vehicle timing architecture

- **Vehicle functional architecture**: It should represent the functional decomposition of the vehicle by showing the vehicle sub-systems (including the sub-system under development) and their interactions.
- **Vehicle timing architecture**: It represents a set of end-to-end flows formed by the vehicle sub-systems (including the sub-system under development). These end-to-end flows should be annotated with the vehicle end-to-end requirements that should be respected in this phase.

In the next section, we present the development of the minimum analyzable model in our methodology by annotating some UML diagrams with EAST-ADL and TADL concepts.

2.2. Solutions for Analyzable Model and Timing Analysis

In this section, we present the solution of our methodology to develop the minimum analyzable model and the heuristics for the timing analysis in this phase.

2.2.1. Development of Analyzable Model

To develop the analyzable model that contains the minimum information presented in 2.1.2, we develop the following views: "analysis functional view" and "analysis timing view". The term "analysis", as used here, refers to the analysis phase. Figure 9c gives an overview of these two views.

- **Analysis Functional View**: This view represents the features of the vehicle functional architecture presented previously (cf. section 2.1.2). To model this view, we use EAST-ADL concepts for functional modelling and UML composite structure diagrams to tangibly represent said concepts. The vehicle is modelled as a white box that shows its functions/sub-systems, including the sub-system under development and the latter's interaction with other vehicle functions. Note that the sub-system is depicted here as a black box. To develop this view, we use an UML editor that implements an UML profile for EAST-ADL, this enables us using UML diagrams and annotating them with EAST-ADL concepts. The following guidelines should be respected to develop this view:
  - The vehicle should be modelled as an UML class (container)
  - Each vehicle sub-system (including the sub-system under development) should be modelled as an UML property and stereotyped with *"AnalysisFunctionType"* from EAST-ADL
  - Each vehicle element representing a sensor or an actuator should be modelled as an UML property and stereotyped with *"FunctionalDevice"* from EAST-ADL.
  - The interaction between vehicle elements should be modelled by UML connectors and stereotyped with *"FunctionConnector"* from EAST-ADL.
  - The communication interface of each element should be modelled as an UML port and stereotyped with *"FlowPort"* from EAST-ADL.

- **Analysis Timing View**: This view represents the features of the vehicle timing architecture presented previously (cf. section 2.1.2). To model this view, we use UML

sequence diagrams that we annotate with TADL concepts to model the event chains (end-to-end flows) and annotate them with TADL constraints. At the end of the next chapter, a paragraph explains how this timing view is developed using sequences diagrams and TADL concepts. To develop this view, the following guidelines should be respected:

- Each flow of sub-systems should be modelled as a UML interaction and stereotyped with *"EventChain"* form TADL.
- Each sub-system involved in the flow should be modelled as a lifeline with an action execution specification. The action execution specification should be stereotyped with *"EventChain"* and specified as an *"EventChainSegment"* for the whole UML interaction.
- Each message should be stereotyped with *"Eventchain"* from TADL and *"DataMessage"* (this concept will be detailed in the next section)
- Each vehicle-end-to-end requirement should be specified as a TADL *"ReactionConstraint"* for the whole UML interaction.

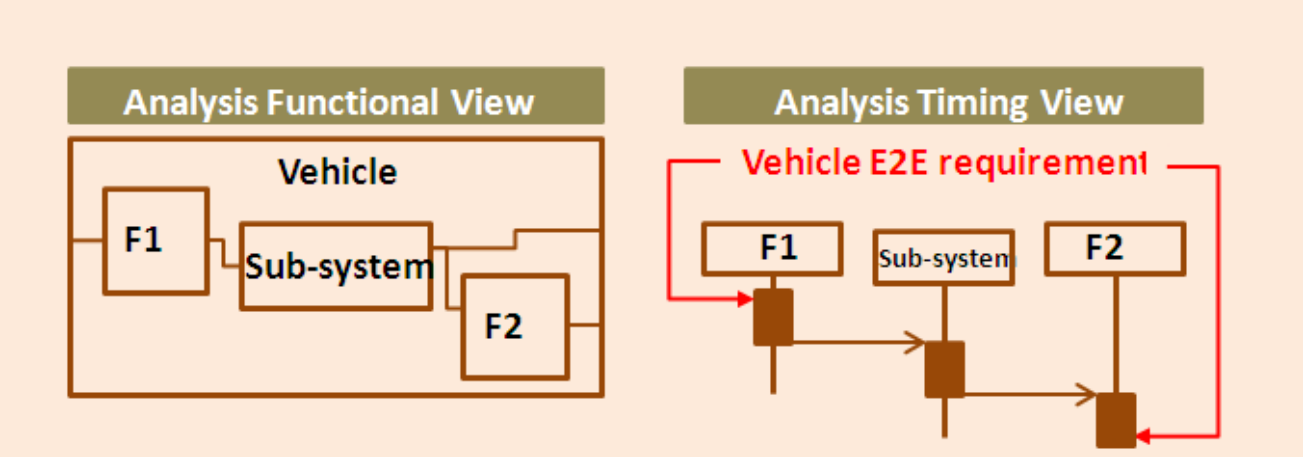

Figure 9a Analyzable model overview of the analysis phase

2.2.2. Determination of sub-system time budgets

2.2.2.1. Introduction

For each vehicle end-to-end requirement, the designer determines a time budget to be allocated to the sub-system ensuring compliance with this requirement. Time budgets can be determined using a tool whose input is the timing view of the analyzable model and whose output is a time budget for each specified end-to-end requirement. This operation can

also be performed manually based on designer expertise (to facilitate the process, we assume here that the *time budgets for the other vehicle functions/subsystems are already known*). In our methodology, we also suppose that *for each vehicle end-to-end requirement, we obtain exactly one time budget for the sub-system.*

Once timing analysis is completed, the designer possesses a set of sub-system time budgets that ensure compliance with the vehicle end-to-end requirements. Such time budgets are the input constraints that the designer needs to consider when refining sub-system architecture at the design stage. Each time budget namely represents an internal end-to-end constraint that should be satisfied when describing the sub-system functional blocks at design level.

2.2.2.2. Sub-system Time Budgets

To determine sub-system time budgets, the designer has a set of vehicle end-to-end requirements that involve several vehicle sub-systems/functions including the sub-system under development. These vehicle sub-systems communicate together through exchanging data. Let's consider the example shown by Figure 10. The considered sub-system communicates with five functions within the vehicle as shown in the figure (For the clarity of the models, we do not show the EAST-ADL stereotypes in the following figures, however detailed models are shown in the examples presented in the next part of this manuscript).

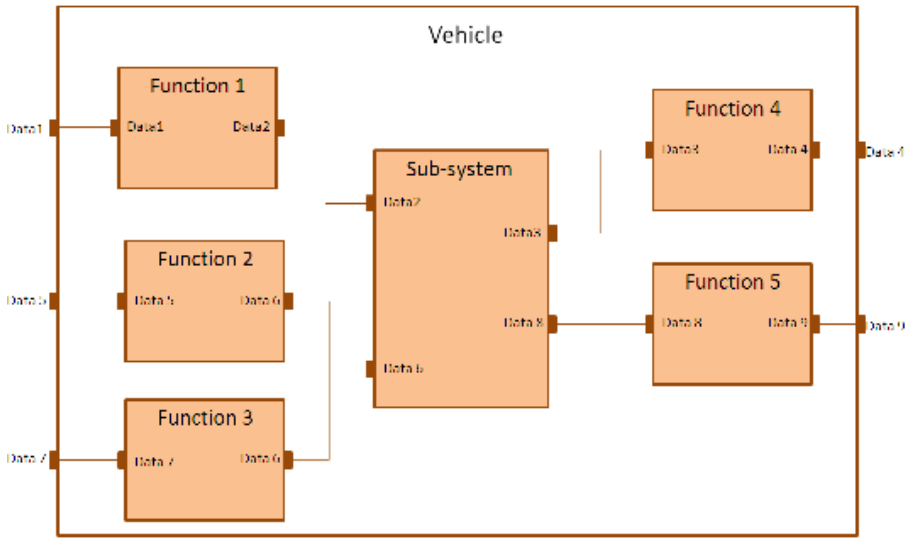

Figure 10: example of a sub-system functional analysis view

Let's consider the following vehicle end-to-end requirement that we call Req: "*From the activation of "function 1" until the termination of "function 4", the duration should not exceed*

100ms". Figure 11 shows the flow of vehicle functions involved in this vehicle end-to-end requirement (function 1, sub-system, function 4)

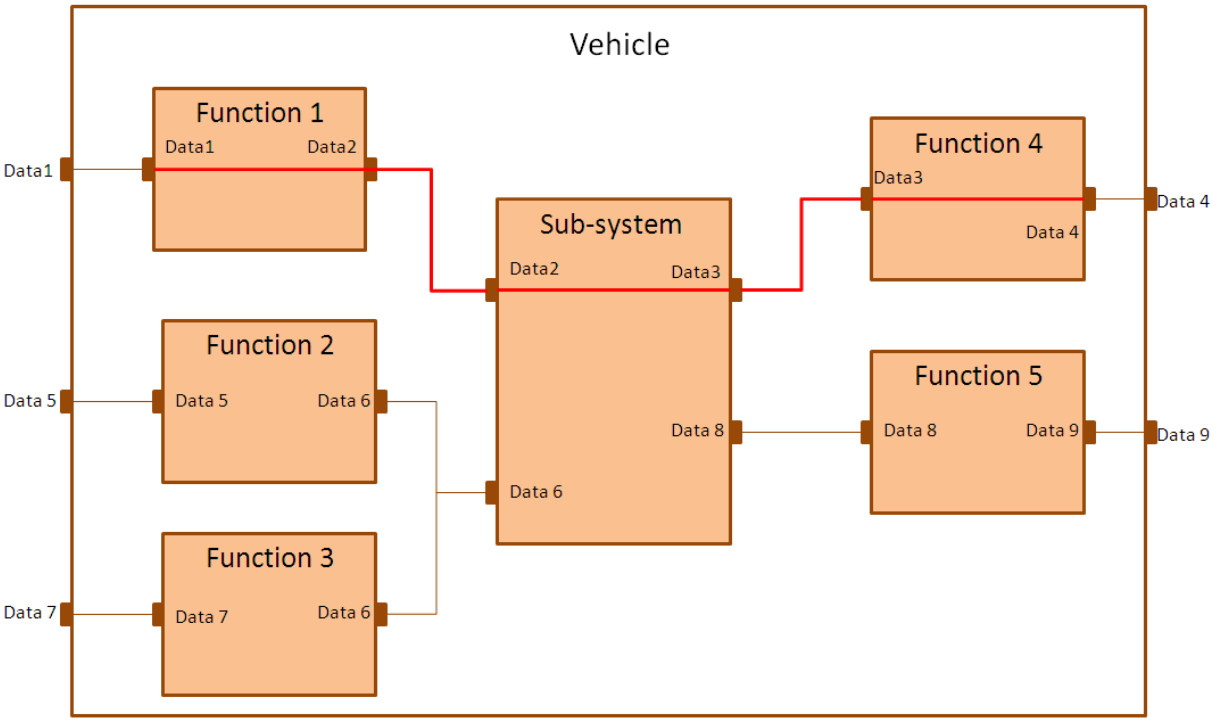

Figure 11: Flow of vehicle functions involved in Req

Let's suppose that the time budget of "function 1" is 20ms and the time budget of "function 4" is 50ms. Thus, the time budget that should be allocated to the sub-system for compliance with this vehicle end-to-end requirement is 30ms (let's call it TB). This time budget means that from the reception of "data 2" by the sub-system until the production of "data 3", the duration should not exceed 30ms. Let's call each flow within the sub-system (i.e., from the reception of an input data by the sub-system until the production of an output data) "sub-system internal flow". Hence, the time budget TB imposes a constraint on the delay of the sub-system internal flow "reception of data 2-production of data 3".

A particular use case should be considered when determining sub-system time budgets. Let's consider the following two vehicle end-to-end requirements:

- Req 1: *From the activation of "function 2" to the termination of function 5, the duration should not exceed 200 ms.*

- Req 2: *From the activation of "function 3" to the termination of "function 5", the duration should not exceed 150 ms.*

Figure 12 shows the flow of vehicle functions involved in each vehicle end-to-end requirement (the broken line depicts the flow of functions involved in Req 1 and the solid line represents the flow corresponding to Req 2)

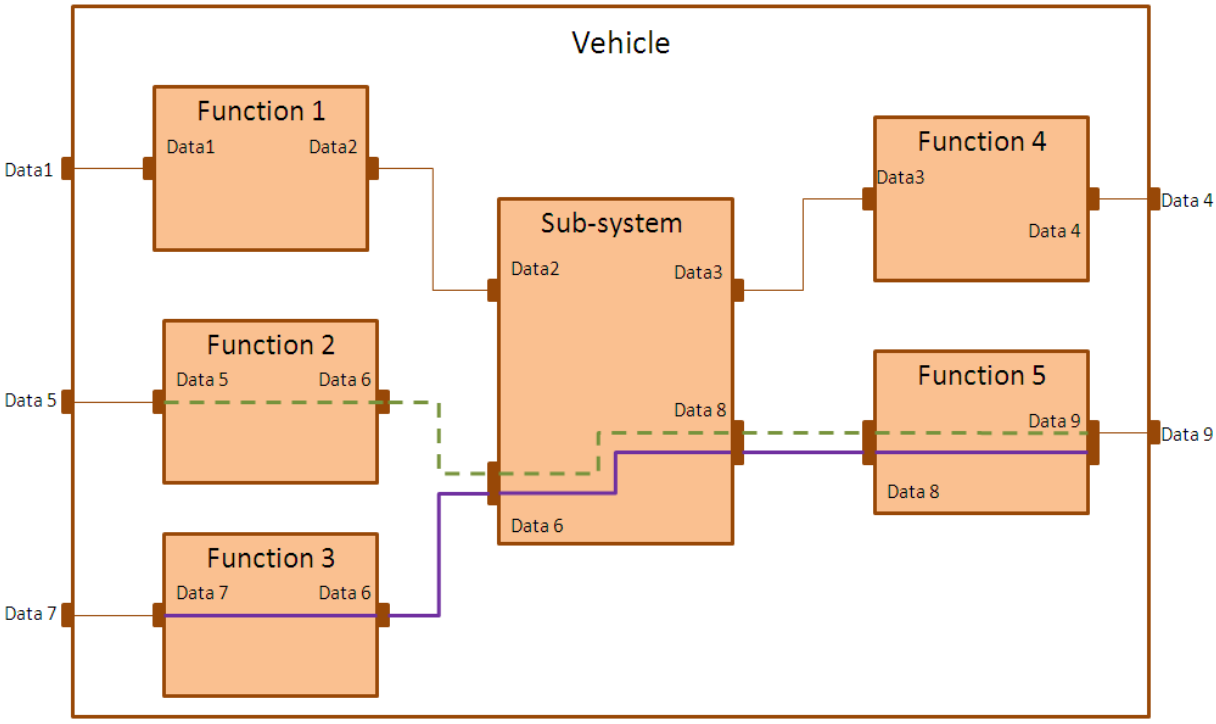

Figure 12: flow of vehicle functions involved in Req1 and Req 2

As stated earlier, for each vehicle end-to-end requirement, we determine a sub-system time budget that allows respecting this vehicle end-to-end requirement. Hence, in our case, we determine two time budgets (let's call them TB 1 and TB 2) for the sub-system. However, as the figures show, our sub-system acquires "Data 6" from both "function 2" and "function 3". This means that both TB1 and TB2 impose a constraint on the same sub-system internal flow (reception of "data 6"-production of "data 8"). In this case, we should decide which time budget to keep for the remaining of the work. Two cases should be considered based on the operating mode of the sub-system (*an operating mode corresponds to a particular state of the sub-system depending on the interaction of the sub-ystem with its environment. For example, depending on the detection of a failure, the sub-system can be in a failure mode or in a nominal mode (without failure)).*

- Case 1: if each time budget corresponds to a different sub-system operating mode (e.g. one time budget correspond to the activation mode of the sub-system and the

other one to its deactivation mode), then the two time budgets should be kept and the analysis performed during next steps should consider each operating mode separately.

- Case 2: if the two time budgets correspond to the same sub-system operating mode, then we keep only the smallest time budget and the further analysis should be performed considering only this time budget.

3. Design Phase

During the design phase, the system architecture model obtained in the analysis phase is further refined and two timing analysis activities are carried out. The first consists of refining the sub-system time budgets determined at analysis level. The second is an exploration of the hardware architecture based on an evaluation of processor utilization for each functional-block-to-ECU allocation scenario. To evaluate this utilization, the designer should have previously estimated the execution times required for each functional block.

3.1. Refinement of Sub-system Time Budgets

Refining the sub-system time budgets determined during the analysis phase means evaluating the time budgets to be allocated to the functional blocks so that vehicle end-to-end requirements are still met after design-phase refinement of sub-system functional architecture. To determine the functional block time budgets, the same approach is used as for the analysis phase. This requires first developing an analyzable model that contain the minimum information for such analysis.

3.1.1. Analyzable Model

3.1.1.1. Analyzable Model Minimum Features

We organize the features of the minimum analyzable model in two categories: the sub-system functional architecture and the sub-system timing architecture.

- **Sub-system functional architecture**: It should represent the functional decomposition of the sub-system by showing the functional blocks that compose it and their interactions.

- **Sub-system timing architecture**: It represents a set of end-to-end flows formed by the sub-system functional blocks. These end-to-end flows should be annotated with the sub-system time budgets that have been determined in the previous phase- analysis.

3.1.1.2. Solution for the Analyzable Model

To represent the minimum features of the analyzable model, we develop two views, a functional view ("design functional view") and a timing view ("design timing view"). Note that the term "design" in these views and the following discussion refers to the design phase.

- **Design Functional View**: This view represents the features of the sub-system functional architecture mentioned previously (cf. 3.1.1.1). It refines the Analysis functional view of the analysis phase. The sub-system modelled as a black box during the analysis phase is therefore depicted here as a white box showing the functional blocks and the interactions between them. This view is also developed using EAST-ADL concepts for functional modelling, and UML composite structure diagrams. To develop this view, the following guidelines should be respected:
  - The Sub-system should be modelled as an UML container class and stereotypes with "*DesignFunctionType*" from EAST-ADL.
  - Each functional block should be modelled as an UML property and stereotyped with "*DesignFunctionPrototye*".
  - The interaction between the functional blocks should be modelled by UML connectors and stereotyped with *"FunctionConnector"* from EAST-ADL.
  - The communication interface of each element should be modelled as an UML port and stereotyped with *"FlowPort"* from EAST-ADL.
- **Design Timing View**: This view represents the features of the sub-system timing architecture presented previously (cf. 3.1.1.1). It refines the analysis timing view of the analysis phase. It depicts a set of flows formed by the functional blocks making up the sub-system. For each sub-system time budget determined during the analysis phase, we model an end-to-end flow containing the functional blocks concerned by the budget. For example, if we determine a time budget to be allocated to the sub-system during its activation, we model an end-to-end flow of sub-system functional blocks that participate in sub-system activation and we specify said budget as an end-to-end constraint on this

flow. In the same way as for the analysis phase, this view is modelled using UML sequence diagrams annotated with TADL concepts to model event chains and timing constraints. At the end of this chapter, a paragraph explains how this view and the timing view of the analysis level are developed using sequence diagrams and how these views are derived from functional views. The following guidelines should be respected to develop this view:

- Each flow of functional block should be modelled as an UML interaction and stereotyped with *"EventChain"* form TADL.
- Each functional block involved in the flow should be modelled as a lifeline with an action execution specification. The action execution specification should be stereotyped with *"EventChain"* and specified as an *"EventChainSegment"* for the whole interaction.
- Each message should be stereotyped with *"Eventchain"* from TADL and *"DataMessage"* (this concept will be detailed in the next section)
- Each sub-system time budget determined in the previous phase should be specified as a *"ReactionConstraint"* from TADL for the whole UML interaction.

3.1.2. Determination of Time Budgets for Functional Blocks

For each sub-system time budget (modelled as an end-to-end constraint in the design timing view), the designer determines a time budget to be allocated to each functional block to satisfy the constraint. Distribution of the time budgets to the functional blocks is based on the expertise of the designer and the nature of each functional block. For example, a functional block performing a simple signal transformation will have a small time budget. One performing complex processing that requires much more time will then have a larger time budget. Budget allocation should take place in such a way that the overall time budget determined for the sub-system is likewise met. After this timing analysis, the designer possesses a number of time budgets for each functional block. Each such functional block time budget corresponds to a different sub-system operating mode and should be met during said operating mode (e.g. a function that participates in sub-system activation and failure detection will have a time budget for each of these operating modes). The functional block time budgets determined during this phase are used during the implementation phase, after the system functional architecture is transformed into a software architecture with software

components and runnable entities. These time budgets are then refined to determine the time budgets to be allocated to the runnable entities or to end-to-end flows formed by a number of communicating runnable entities, etc. The latter represent input constraints for the scheduling analysis activity performed at the implementation stage.

Figure 13 shows an overview of the model development and the timing analysis of the design phase that refines the models and the timing results of the previous phase –analysis.

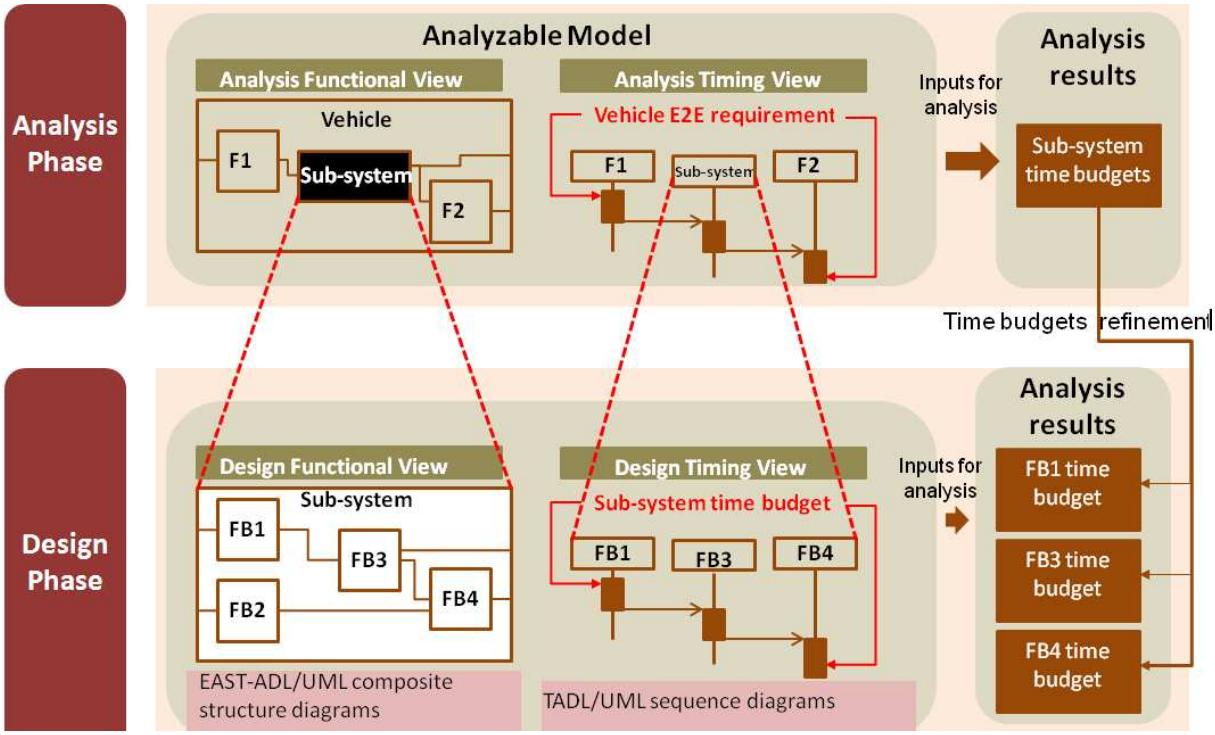

Figure 13 model and timing results refinement from analysis to design phase

3.2. Performance of Hardware Architecture Exploration

At this stage, we assume that the hardware platform to be used by the sub-system has been already chosen (this is done to comply with the current automotive development process, in which new sub-systems are integrated into a vehicle for which there is a pre-existing software and hardware resource platform). The analysis performed here is thus geared to ensuring correct integration of the sub-system with other vehicle functions in terms of requested processor load. Based on a functional block-to-available ECU allocation scenario, the designer evaluates the load requested by the sub-system for each processor. This allows him to determine the allocation scenario that best satisfies any constraints s/he might have with regard to processor utilization. Once the analysis results are known, the designer decides whether to distribute functional blocks over many ECUs or to allocate them to the

same ECU and which functions can be so allocated. To perform this evaluation, it is necessary to develop an analyzable model that contains the minimum information necessary for this analysis.

3.2.1. Development of an Analyzable Model

3.2.1.1. Analyzable model minimum features

In scheduling analysis, to evaluate the utilization of a processor, one needs to specify:

- ✓ The executing processors
- ✓ The executable entities on these processors and their execution times and activation periods
- ✓ The allocation of the executable entities to the processors

Hence, to perform this evaluation, we organize the minimum information needed for the analyzable model in three categories:

- **Sub-system functional architecture**: It represents the functional blocks that compose the sub-system under development (these functional blocks represent the executable entities that contend for the use of the executing processors). The execution time and the activation period of each functional block should be specified. These parameters can be determined based on designer expertise, measurements or knowledge of former versions developed for the sub-system.
- **Hardware platform**: It represents the hardware resources on which the functional blocks can execute. For our analysis, we don't need to model the software resources such as OS tasks.
- **Allocation**: It represents the allocation of the functional blocks to the hardware resources. For this analysis, we don't model the allocation of the functional blocks to the software resources but we allocate the functional blocks directly to the hardware resources.

3.2.1.2. Solution for Analyzable Model

To end up with the minimum analyzable model necessary for this analysis, we developed a modelling framework that combines EAST-ADL and MARTE to model the information

necessary for the analysis. This modelling framework is composed of three views: design functional view, hardware platform view and allocation view.

- **Design Functional View**: This view represents the features of the sub-system functional architecture (It is not different from the view described earlier, in which sub-system functional blocks are described using EAST-ADL functional modelling concepts). As also mentioned above, these functional blocks are represented as *"DesignFunctionPrototypes"*. These *"DesignFunctionPrototypes"* are typed by *"DesignFunctionTypes"* for which we specify the execution times estimated during the previous step using the EAST-ADL concept *"ExecutionTimeConstraint"*. The activation period of each functional block is specified through the concept "*Trigger*" of EAST-ADL (this concept allows describing the activation pattern of an EAST-ADL *FunctionType*). The guidelines for the development of this view have been described in 3.1.1.2.

- **Hardware Platform View**: This view represents the features of the hardware platform (cf. 3.2.2) In this view, we represent the hardware resources (e.g. ECUs) that are used by the sub-system. To model the view, we use UML composite structure diagrams. EAST-ADL concepts for hardware modelling are supplemented here by MARTE concepts for hardware resource platform modelling. The following guidelines should be respected to develop this view:

  - The hardware platform should be modelled by a UML container class and stereotyped with "*SaResourcePlatform*" from MARTE.

  - Each execution hardware resource (e.g. ECU) should be modelled by a UML property and stereotypes with "*SaExecHost*" from MARTE and "*Node*" from EAST-ADL.

  - Each Communication hardware resource (e.g. bus) should be modelled by a UML property and stereotyped with "*SaCommHost*" from MARTE and "*LogicalBus*" from EAST-ADL.

- **Allocation View**: This view represents the features of the allocation (cf. 3.2.2). In this view, we use a key concept from MARTE which is *"SaAnalysisContext"*. This concept helps to bind the model elements to a particular evaluation scope. The core of the binding concept is the allocation of functions executed in the scenario of interest, to the

resource platform (Note that during this phase, we abstract the software resource platform such as OS tasks and allocate functional blocks directly to hardware resources). Such allocation is carried out by specifying a UML composite diagram stereotyped as *"SaAnalysisContext"*. In this way, the composite diagram contains two main parts representing the sub-system design functional view with the functions to be allocated and the hardware platform view respectively. To represent the allocation relationships, MARTE concepts for allocation are used. Functional blocks are stereotyped as *"allocated"*. This stereotype allows specifying the resource to which the function is allocated. A dependency connector is drawn between each function and its hosting resource and stereotyped as *"allocate"*. The following guidelines should be respected to develop this view:

- A UML container class should be modelled and stereotyped with "*SaAnalysisContext*" from MARTE.
- The allocation relationships should be modelled with UML dependency connectors and stereotyped with "*allocate*" from MARTE.
- Each functional block should be modelled as a UML property and stereotyped with "*allocated*" from MARTE.

Figure 13a shows an overview of the analyzable model needed for hardware architecture exploration.

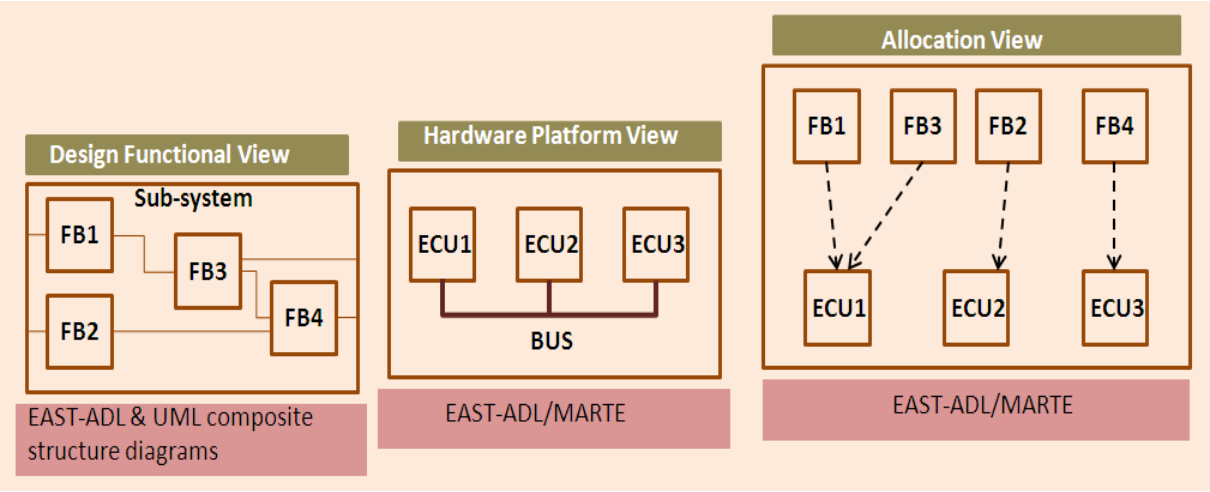

Figure 13a Overview of the analyzable model for hardware architecture exploration

3.2.2. Evaluation of Processor Loads

3.2.2.1. Principle

Starting from the allocation view of the model, a scheduling analysis tool can follow the links of the model to extract the information that it needs to perform processor load evaluation (function execution times, allocation, hardware resource parameters, etc). As the original aim of scheduling analysis tools is to verify if a task set is schedulable or not, all of them require specifying the OS tasks involved in the sub-system. However, in our approach, we abstract the OS task model during this phase, showing only the allocation of functional blocks to hardware resources. Therefore, to be able to use a scheduling analysis tool for our purpose, our model should be transformed in an accurate way to obtain the model required by the tool. Some scheduling analysis tools require a description of the allocation of functions to OS tasks and the allocation of OS tasks to processing resources. Other tools require only the allocation of OS tasks to processing resources. In both cases, to be able to use such tools to analyze our model, each functional block defined in that model should be transformed into an OS task in the analysis tool model (or into an OS task allocating only one function). The execution time determined for each functional block should be then assigned to the defined OS task (or to the function that it allocates). As our goal here is not to perform complete scheduling analysis, but just to evaluate processor loads (without timing constraint verification), the choice of the priorities to be assigned to the different tasks is not important (to calculate processor utilization, one needs to specify only the task execution times and activation periods without specifying their priorities).

3.2.2.2. Tool Use and Model Transformation

To evaluate processor loads, we claim to use the scheduling analysis tools MAST, cheddar or SymTA/S (cf. section 3.4. of part I). In this section we show the mapping that should be performed to transform the analyzable model to a cheddar or MAST model (we encourage the use of these two tools as they are open source and free, SymTA/S will anyway be used to perform complete scheduling analysis in the implementation phase). Table 14a and 14b show respectively the mapping of the elements of the analyzable model to a MAST and Cheddar model. Note that an automatic transformation is already implemented from MATE models to MAST in the context of another research work.

Table 14a Mapping of analyzable model elements to Cheddar elements

Analyzable model element	Stereotype	Cheddar element
Functional Block	DesignFunctionprototype, Allocated	Task
Execution hardware resource	Node, SaExecHost	Processor
Communication hardware resource	LogicalBus, saCommHost	Network
Functional block execution time	ExecutionTimeConstraint	Task computation time
Functional block activation period	Trigger	Task period
Allocation relationship	Allocated, Allocate	Task property called "processor"

Table 14b Mapping of analyzable model elements to MAST elements

Analyzable model element	Stereotype	MAST element
Functional Block	DesignFunctionprototype, Allocated	Transaction with only one activity representing a Scheduling server hosting only one Operation
Execution hardware resource	Node, SaExecHost	Processing resource (regular processor)
Communication hardware resource	LogicalBus, SaCommHost	Bus (packet based network)
Functional block execution time	ExecutionTimeConstraint	Operation execution time
Functional block activation period	Trigger	Transaction external event
Allocation relationship	Allocated, Allocate	Activity parameters for the specification of the scheduling server and operation

Using Sequence Diagrams to Represent Timing Views at Analysis and Design Levels

The objective of this section is to describe how system timing views are represented at analysis and design levels. As specified in the methodology description, analysis and design functional views are represented using UML composite structure diagrams annotated with EAST-ADL concepts for functional modeling. At each level, the aim is to derive from the functional view a timing view where we can represent the end-to-end constraints to be satisfied when determining the necessary time budgets.

To represent the timing views, we opted for the use of TADL concepts to represent constrained end-to-end flows by means of events and event chains. The questions to be answered are the following:

- How to move from the EAST-ADL/Composite structure diagram model elements to TADL elements?
- How to represent the TADL timing view using an UML behavioral diagram?

1. From EAST-ADL/composite structure diagram models to TADL

Objective: We have as input a composite structure diagram representing the interaction between several functions. Some flows formed by these functions are submitted to end-to-end constraints. We want to represent these flows and their constraints using TADL events and event chains. How to map the elements of the EAST-ADL/composite diagram model with TADL events and event chains?

Solution: Each flow of functions will be represented as a TADL event chain. As our aim is to specify a time budget for each function involved in the end-to-end flow, each arrival of data on the input port of a function and the production of data in the output port of a function will be considered as an observable event and modeled as a TADL event. Consequently, each function involved in the end-to-end flow will be represented as an event chain segment.

Figure 14 shows an example of an EAST-ADL model developed using UML composite diagram. The end-to-end flow formed by the functions function_1, function_2 and function_3 is submitted to an end-to-end constraint as shown in the figure. To derive the timing view from this model, we identified the observable events (here we

considered that the arrival of data 1 at the input port of the container system and the arrival of this same data at the input port of function_1 occurs at the same instant so we considered only one observable event (event 1), we did the same to produce data 4 (delegation delays are neglected))

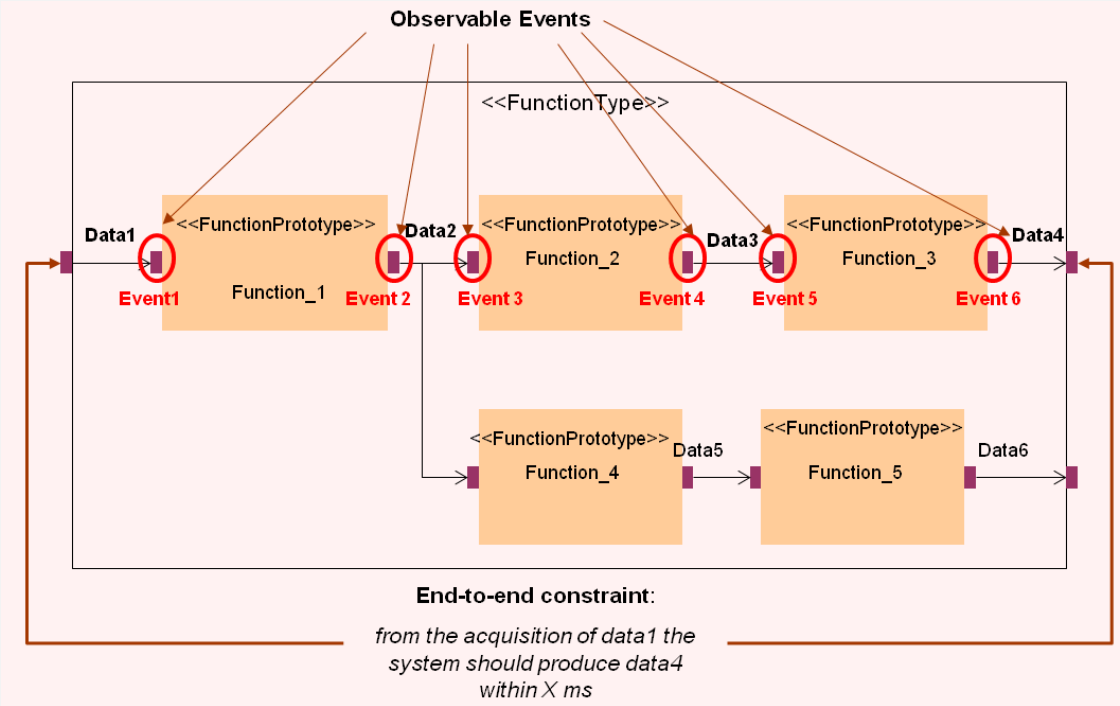

Figure 14 Observable event in EAST-ADL functional model

Figure 15 shows the deriving of the TADL timing view from the EAST-ADL/composite structure view

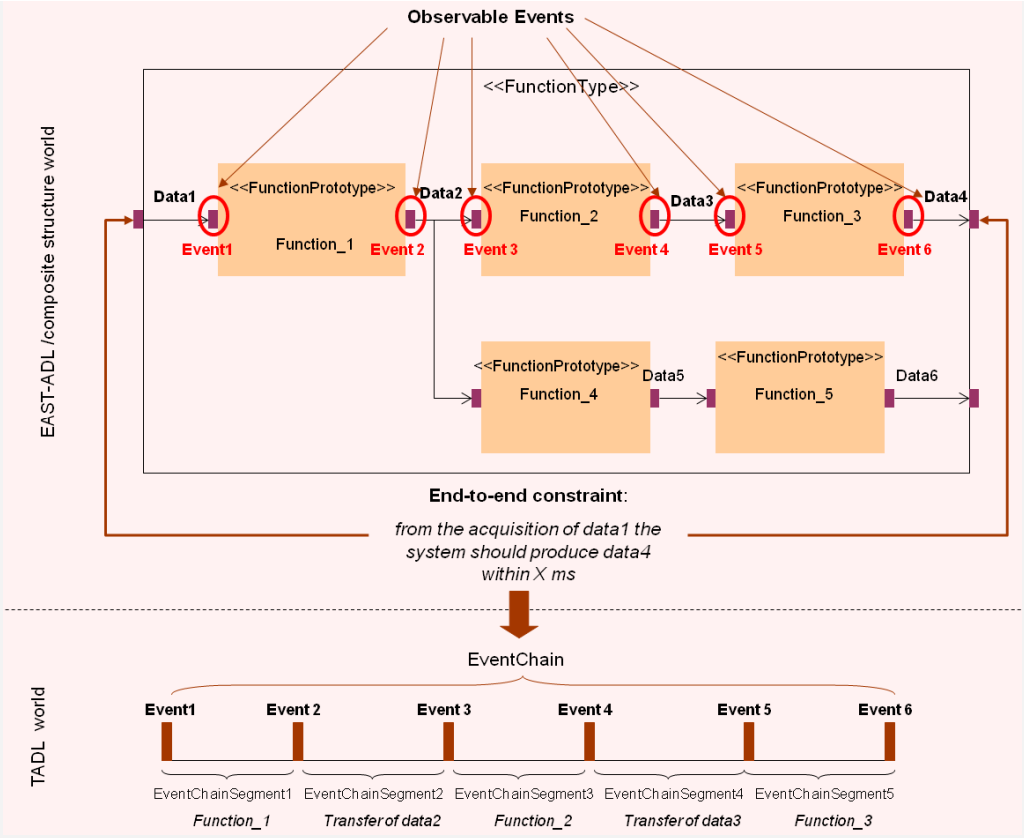

Figure 15 From EAST-ADL functional view to TADL view

2. Representing TADL timing views using sequences diagrams

Now, as we defined how to use TADL concepts to model the timing views, the question that we should answer is how to represent concretely this timing view?

In our approach, we propose to use UML behavioral diagrams. According to the UML 2.0 specification [7], seven UML diagrams can be used to specify the behavior of a system: Activity, Sequence, Communication, Interaction Overview, Timing, Use Case and State Machine diagrams. In this work, the closest diagram to model the required timing views is sequence diagram. Sequence diagrams represent a particular scenario of communication between collaborating components. Sequence diagrams do not focus only on message passing but also the chronological order of this communication. This fits well our case as we want to represent and end-to-end flow of functions representing a particular scenario of communication between these functions. To have an accurate representation of TADL end-to-end flows with sequence diagrams we should first answer the following questions:

- What are the observable events in a sequence diagram?; this will allow us defining the elements to be annotated with TADL events

- In EAST-ADL, communication between functions is assumed to be asynchronous based on data exchange, how to represent this in sequence diagrams?

To answer these questions, let's remind first some notions in sequence diagrams: A sequence diagram represents the message interchange between lifelines. A message defines different ways of communication between lifelines of one interaction, generally involving a pair of sender and receiver. Message may be of the following kinds: synchronous or asynchronous operation call, asynchronous signal post, creation or delete of an object, or a reply message. In UML2, a message owns generally two message ends: one refers to the event occurrence related to the posting of the message, while the other refers to the event occurrence related to the receipt of the message. Currently, due to its initial intent, the UML2 interactions chapter defines only specific events dedicated to either operation-based message or signal-based message. For each lifeline it is possible to associate an Execution Specification that represents the execution of an action or behavior within the lifeline. Each execution specification occurrence is associated to two events that represent respectively the start and the end of the action or behavior execution

Observable events in sequence diagrams

As stated before, each message in a sequence diagram is associated to two event occurrences, the first relates to the sending of the message and the second to the reception of the message. We consider then each sending event and reception event of a message as an observable event (and hence these events will be stereotyped with TADL events). Each function involved in the end-to-end flow will be modeled as a lifeline containing an Action Execution Specification. The events representing the start and end of each occurrence of an action execution specification will be considered also as observable events and stereotyped with TADL event.

Data based communication issue

The main paradigm for communicating within sequence diagrams is the message that involves either operation call-based or signal-based communication. This is not sufficient for our purpose, because EAST-ADL2 enables also structural entities (the «FunctionTypes») to communicate by data-passing. So, we need to extend the message concept as defined in the chapter interaction to enable UML sequence diagrams to support data-based communication. As shown in figure 16 we define then the stereotype

«DataMessage». This latter owns a property value, which models the data value conveyed by the message.

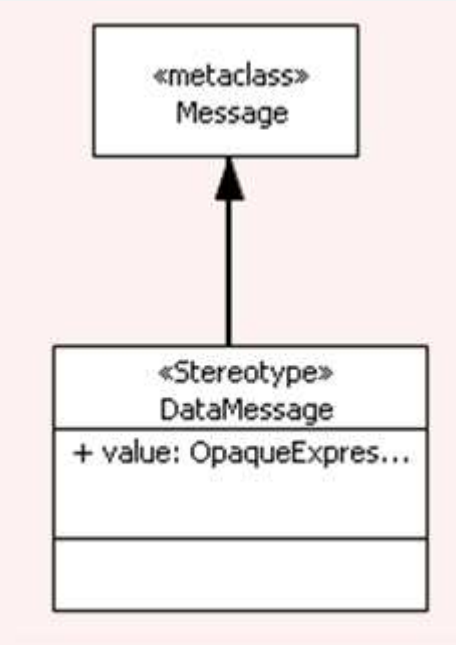

Figure 16 Definition of the DataMessage concept

As mentioned previously, UML2 interactions define events related to either operation-based message or signal-based message. We need then to extend also the UML2 Event concept to enable events related to data-based communication. As shown in figure 17, we define an abstract class «DataEvent» that extends the UML2 Event concept. This class is specialized by «RecieveDataEvent» and «SendDataEvent» to express respectively events related to the reception and the sending of a DataMessage.

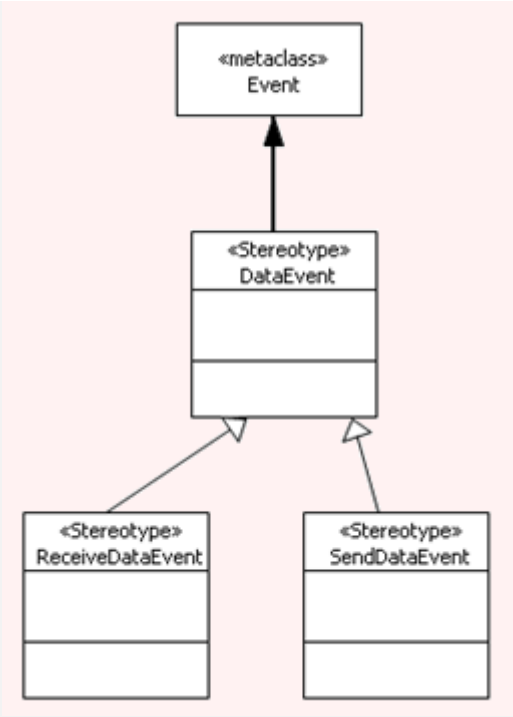

Figure 17 Definition of the DataEvent concept

This same extension approach is described in [62] by Gérard and Servat who defined a MARTE annex for EAST-ADL modelling (this annex has been added to the MARTE last release specification).

Figure 18 shows an overview of the timing view obtained for the example presented in figure 15

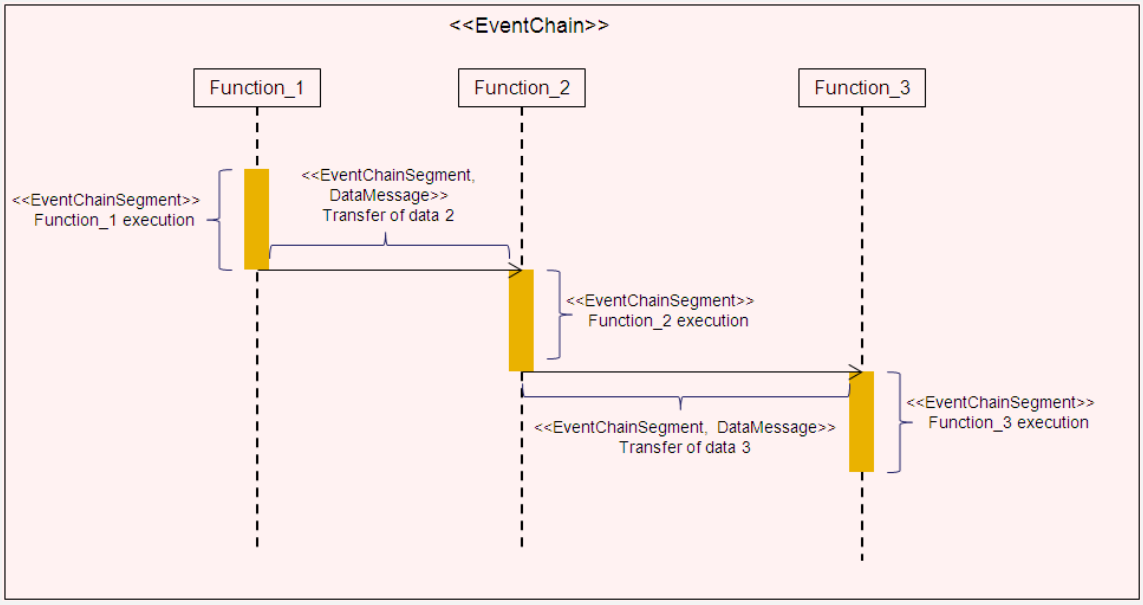

Figure 18 Timing view example

4. Implementation Phase

During this phase, the sub-system functional architecture modelled in the design phase is refined and transformed into software architecture described using software components and runnable entities. The hardware and software platform is also refined and the mapping (allocation) is specified (mapping of runnable entities to OS tasks and mapping of OS tasks to hardware resources). A complete scheduling analysis can thus be performed during this phase, since all the required information is available (OS task model, allocation, timing information, etc). In the same way as for the two previous phases, an analyzable model should be developed. This is done using AUTOSAR concepts. Beforehand, however, the designer needs to obtain timing information for the runnable entities involved in the system. By timing information, we mean the execution times of the runnable entities and their timing constraints.

4.1. Determination of Runnable Entity Timing information

As mentioned at the beginning of this section, the timing information considered here concerns the execution times and timing constraints for runnable entities.

4.1.1. Estimation of Runnable Entity Execution Times

Depending on the choices made to transform the system functional architecture of the design level into software architecture, the designer estimates the execution times of the runnable entities by taking into account the execution times determined for the functional blocks during the design phase. If, for example, a functional block is transformed into a software component with a single runnable, this runnable will have the same execution time as the functional block. The execution times determined during this phase are used to annotate the application view of the analyzable model (more details on model views are provided in a later paragraph)

4.1.2. Determination of Runnable Entity Timing Constraints

In the same way as for execution times, the timing constraints to be respected in this phase depend on the transformation choices made and the time budgets determined for functional blocks at the design stage. If, for instance, a functional block is transformed into a software component with two runnables executing successively, the time budget determined for this functional block at design level is considered as an end-to-end constraint from the activation of the first runnable until the second runnable has executed. The timing constraints

determined during this phase are used to annotate the timing behaviour view of the analyzable model.

4.2. Development of the Analyzable Model

4.2.1. Analyzable Model Minimum Features

To enable scheduling analysis, the analyzable model should contain the following features that we organize into four categories:

- **Application workload**: The application workload represents the processing load of the system. It represents the different operations (functions/runnable entities) executed in the system and contending for use of processing resources and other shared resources. An operation may represent a small segment of code execution as well as the sending of a message through a communication medium. Operations are generally organized in processing flows (set of related operations/functions). To make the analysis possible, scheduling analysis requires the specification of the execution /transmission time (worst, best or average) for operations/messages.

- **Application Timing behavior**: The application timing behavior represents the timing information of the different operations or processing flows involved in the system under analysis. Timing information contains both timing description (timing properties) and timing constraints. Timing description contains the specification of the triggering of system operations or processing flows (recurrence, activation jitters, etc.). Most scheduling analysis tools allow analyzing systems with various triggering patterns such as periodic, sporadic, singular, etc. For those activation patterns, it is necessary to specify the period or the min inter-arrival time of the triggering events. Timing constraints must be met by the system operations or flows. They are represented essentially by operation deadlines, output jitter bounds and end-to-end dead-lines.

- **Resource Platform**: It represents the concrete architecture and capacity of hardware (e.g., CPU or buses) and software (e.g. tasks) resources. For hardware resources such as processors, the model should contain the description of the scheduler used. For a more accurate analysis, it may be also necessary to specify the processor overheads (e.g. context switch overhead). For software resources such as tasks, it is necessary to

specify the task nature (preemptive, non-preemptive, etc.) as well as its priority. Involved shared resources should also be described.

- Mapping (allocation): It represents the allocation of the operations to software resources (e.g. tasks) and the allocation of software resources to hardware resources (e.g. processors).

In the following section, we describe how, based on AUTOSAR concepts, we develop such minimum analyzable model.

4.2.2. AUTOSAR Analyzable Model

The minimum analyzable model developed during this phase contains four views (application view, timing behaviour view, resource platform view and a mapping view). To model each view, concepts from different AUTOSAR templates are used. The different views of this phase are obtained as a refinement of the model of the previous phase, design. Figure 19 shows an overview the model refinement from the design to the implementation phase.

- **Application View**: This view represents the application workload features (cf. 4.2.1) and represents mainly the software architecture of the sub-system using software components and runnable entities. This view is developed as a transformation and refinement of the sub-system design functional view developed at design level. Transformation of the design functional view into a software application view depends on the choices made by the designer. S/he may choose to transform each functional block into a software component with one or more runnables [61]. Due to some constraints, s/he may also choose to concatenate two functional blocks in a single software component. In this view, two aspects are modelled for each software component: component behaviour, where runnable entities and their triggering events are described, and component implementation, where runnable entity execution times can be specified. To develop this view using AUTOSAR concepts the following guidelines should be respected:
  - The sub-system software architecture should be modelled by a set of software component (these software components correspond to the transformation of the functional blocks of the design phase)

- For each software component, an AUTOSAR *Internal Behaviour* should be specified
- Each executable operation in the sub-system should be modelled as an AUTOSAR *Runnable Entity*
- To specify the Runnable Entities execution times a *Software Component Implementation* should be described. In each software component implementation, a *Resource Consumption* should be specified where the maximum, minimum or nominal execution time of the runnable entity can be specified.

- **Timing behaviour View**: This view describes the features of the application timing behaviour (cf. 4.2.1), the designer describes the timing behaviour of the sub-system using AUTOSAR events and event chains for which the previously determined timing constraints are specified. End-to-end constraints and runnable deadlines should, for example, be specified in this view. The following guidelines should be respected to develop this view:
  - Each processing flow of runnable entities should be modelled as an AUTOSAR *EventChain*
  - Each end-to-end constraint imposed on a flow of runnables should be specified as a *Max latency Constraint* for the corresponding event chain.
  - Each event activating the execution of a processing flow should be modelled as a *Stimulus* from AUTOSAR
  - Each event produced at the execution termination of a flow should be modelled as a *Response*
  - To describe the triggering of each processing flow, an event triggering constraint should be defined where the arrival pattern of the stimulus event can be described
- **Resource Platform View**: This view represents the features of the resource platform presented in 4.2.1. It shows the software (e.g., OS tasks) and hardware resources used in the sub-system. This view is obtained by refining the allocation view of the design phase. It namely incorporates more scheduling-oriented features such as the description of the scheduler parameters for each ECU. To develop this view, AUTOSAR concepts from

both OS configuration and System template are used. The following guidelines should be respected when developing this view:

- Each OS task should be modelled as an AUTOSAR "Os Task". Its priority can be specified using the attribute *Os Task Priority*
- Interrupts involved in the system should be described as Os Isr that represents an OSEK interrupt service routine.
- Shared resources should be specified as *Os Resource* from AUTOSAR
- Each ECU should be modelled as an *ECU instance* from AUTOSAR
- Each communication network should be modelled by *Communication Cluster* for which it is possible to specify a *PhysicalChannel* that describes the transmission medium that is used to send and receive information between two communicating ECUs.

- **Mapping View**: This view represents the mapping features described in 4.2.1. It is a refinement of the allocation view described at the design stage (here we use the term mapping rather than allocation to comply with AUTOSAR terminology). In this view, we describe allocation of the runnable entities and to OS tasks. Allocation of the OS tasks to the different available ECUs is also described. To describe the mapping of runnable entities to OS tasks, AUTOSAR concepts for RTE (Runtime Environment) configuration are used. The mapping of a runnable entity to an OS task is based on mapping of its triggering event to this task. The mapping of the OS tasks to ECUs is described using AUTOSAR concepts for OS configuration.

  To describe the mapping using AUTOSAR concepts, one should proceed as follows: The description of the tasks allocated in each ECU is performed in two steps. The first step is the definition of the OS configuration. In this configuration definition, the OS is modelled by an *ECU Configuration Module Definition* element. For this module, one should define an *ECU Parameter Configuration Container* called *OsTask*. Once this definition is done, the second step is the modeling of the concrete configuration of the OS. For this, we define an *ECU Module Configuration Value*. In this module configuration value, we define the corresponding tasks as *ECU Container Values*. These *container values* should have *OsTask* as a definition.

Mapping the runnable entities to OS tasks is done in two steps following the RTE configuration for each ECU. In the first step, which is the definition of the RTE configuration, we create an *ECU Module Definition*. To this module definition, we associate a *container definition* called *RteSwComponentInstance* in which we create another container called *RteEventToTaskMapping*. The later allows referencing the mapped *RTEEvent* and the *OS task*. The second step is the specification of the concrete mapping value of the sub-system runnable entities. This is done by creating *container values* for which we specify the elements created in the first step as definitions.

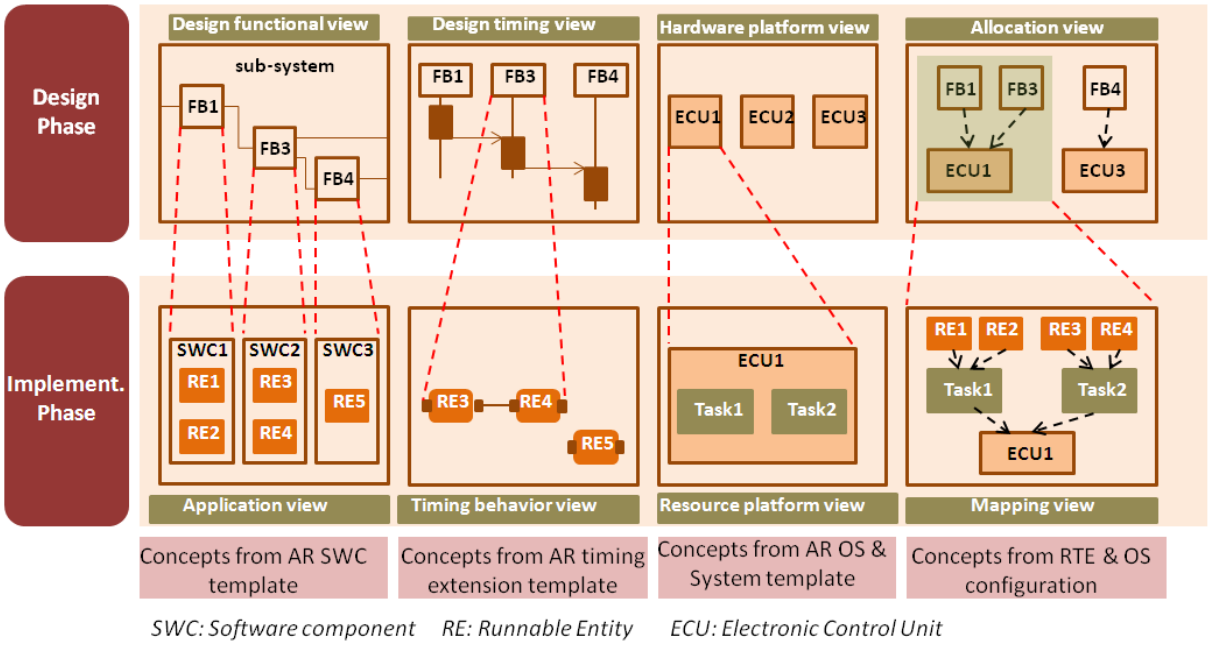

Figure 19 Model refinement from Design to Implementation phase

4.3. Performance of Scheduling Analysis

4.3.1. Principle

To perform scheduling analysis, the developed model is transformed into a model that can be read by a scheduling analysis tool. Note that, at this stage, since our goal is to perform a complete scheduling analysis (evaluation of processor loads and verification of timing constraints), the analysis should take into account all vehicle functions executed on the same resource platform used by the sub-system. The analyzable model views are not changed, but the application view should contain all the software components and runnable entities executed on the same resource platform. The resource platform view shall contain all OS tasks allocated to the hardware resources used by all functions. To verify deadlines, knowing

task priorities and preemption by other tasks is crucial; so the complete software resource platform should be described. Scheduling analysis results help the designer to validate the final architecture or assess the possible tradeoffs required to satisfy timing or load constraints. The tool used for this activity should meet the requirements listed in the section 3.3.1 of the first part of this manuscript to enable scheduling analysis for automotive systems.

4.3.2. Tool Use and Model Transformation

To perform scheduling analysis, we claim to use the scheduling analysis tool SymTA/S (cf. section 3.4 of part I). To perform scheduling analysis, the AUTOSAR analyzable model should be transformed to a SymTA/S model as shown in table 14c.

Table 14c AUTOSAR to SymTA/S model transformation

AUTOSAR analyzable model elements	**SymTA/S model elements**
Runnable entity/non-preemptible flow of runnable entities	Runnable
Event chain	Path (formed by runnable entities)
Runnable entity execution time	Runnable execution time
Event	Task activation/ runnable activation
OS Task	Task
ECU instance	ECU
Physical channel	Bus

When transforming the AUTOSAR model to a SymTA/S model, a special care should be taken when defining the runnables in SyMTA/S. In fact the concept of runnable in SymTA/S represents a non-preemptible entity executing in an OS task. Hence this can map to the concept of runnable entity in AUTOSAR but also to any non-preemptible flow of runnable entities in AUTOAR.

Part III: Methodology Deployment and Validation

In this part, we focus on the deployment and the validation of our methodology. The methodology deployment means how we intend to apply the proposed methodology to develop automotive applications. In this work, we focus on the application of the methodology to develop Engine Management Systems (EMS) at Continental. An Engine Management System (EMS) is a system used to control the engine functionalities (e.g., Combustion, injection, ignition, etc). An EMS consists of software parts implemented in an Electronic Control Unit (ECU) that can communicate with sensors and actuators.

The methodology validation is done through studying the acceptability of the methodology and through showing the extent to which this methodology provides solution for automotive software development needs determined in the first part of this work.

This part is then divided to four chapters. The first chapter presents the approach describing the application of the methodology in the context of EMS development. The approach deals with two scenarios: the development from scratch and the development by reuse. The second chapter illustrates the approach by presenting an example of the application of the methodology to two use cases: the cruise control (development from scratch) and the knock, a component used to detect "knock" and to adjust the ignition accordingly (development by reuse).

The third chapter studies the methodology acceptability through identifying the gap between the current EMS development process at Continental and the process proposed by our methodology.

The last chapter studies the extent to which this methodology provides solution for automotive software development needs determined in the first part of this work.

1. Methodology Application to EMS Development

1.1. Introduction

In this chapter, we present the deployment approach of our methodology within Continental to develop engine management systems. For a better understanding of engine management system, we have to know first how an engine is running. A four stroke engine cycle is composed of four phases:

- **Intake**: the piston moves down aspiring the fuel/air mixture (injection)
- **Compression**: the piston moves up compressing the mixture
- **Power**: a spark generated by an ignition system starts the combustion (ignition), the piston is then pushed down
- **Exhaust**: the burnt gases are evacuated

During the engine cycle, a Crankshaft wheel translates the linear piston motion into rotation, a Camshaft wheel turns to force the valve opening by pressing on the intake/exhaust valves. While the engine speed varies, the connection to the crankshaft wheel fully synchronizes the mechanical cycles of the cylinders. It is therefore useful to date engine operations not by physical time but by the crankshaft angular position.

An Engine Management System (EMS) is a system used to control the engine functionalities (e.g., Combustion, injection, ignition, etc). An EMS consists of software parts implemented in an Electronic Control Unit (ECU) that can communicate with sensors and actuators.

1.2. Engine Management System Development at Continental

This section gives a general description of the current Continental development approach of engine management systems.

Figure 20 gives a general overview about the development process of EMS at Continental.

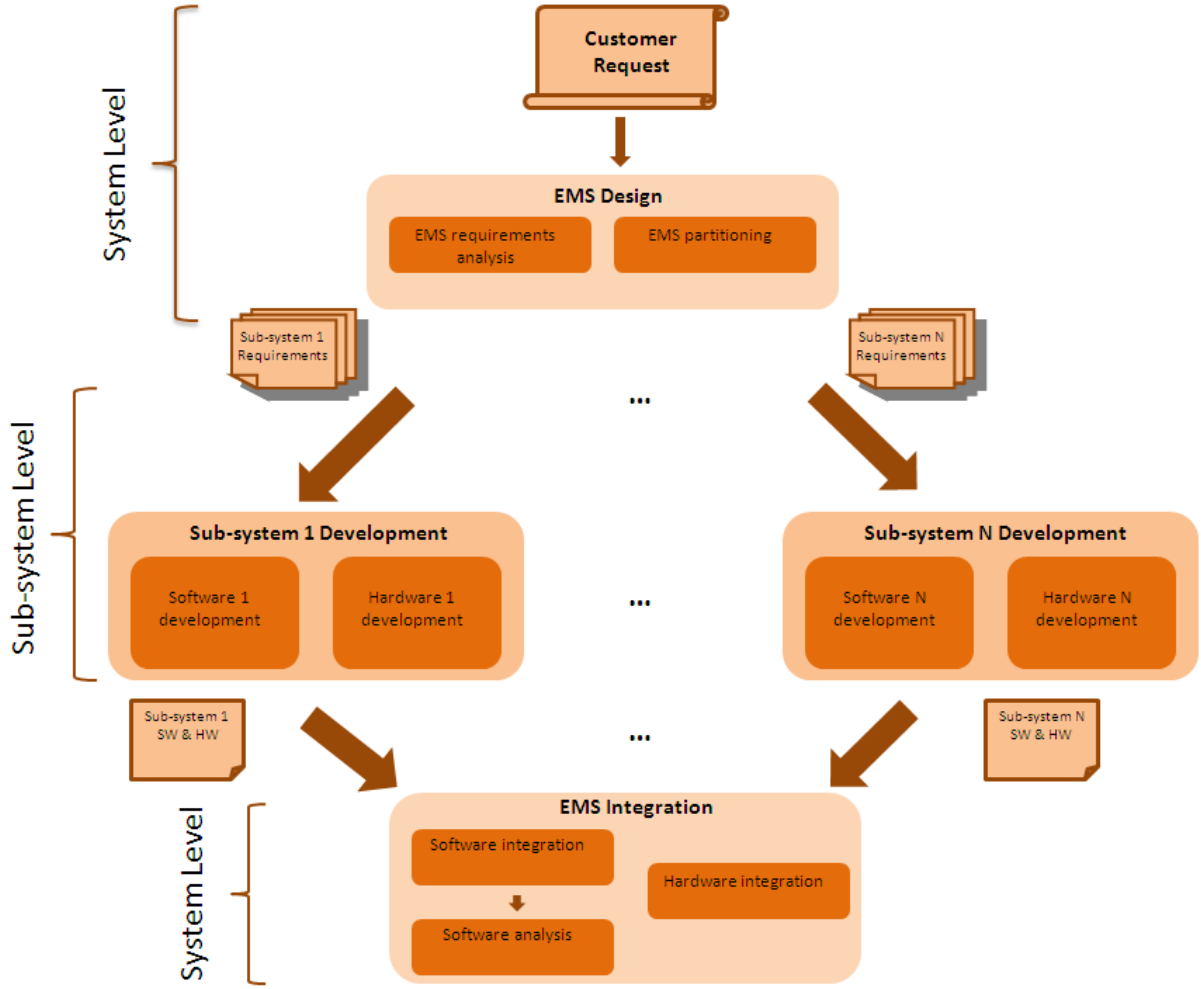

Figure 20 Current EMS development process

As the figure shows, based on the customer request, a first development phase called EMS design is performed. This phase is performed by the *EMS designer* and consists of:

1. EMS requirement analysis: This means collecting and analyzing the requirements that the EMS under development should meet. The requirements that are considered during this phase are of two kinds: functional requirements i.e., requirements that describe the functionality of the system (e.g. the system should calculate the engine speed) and performance requirements which constrain mainly the CPU load and the memory consumption of the system (e.g. CPU total load should not exceed 60%). Currently, timing requirements are not considered during this phase. This kind of requirements are expressed and analyzed very late during the software implementation of each sub-system.

2. EMS partitioning: This consists mainly in defining the needed sub-systems. For example, an EMS can require a sub-system to ensure the injection functionality

(injection sub-system), a second one for calculating the engine speed (engine speed determination sub-system) and a third one to control engine knocking during combustion (knock sub-system). Each sub-system is composed of software and hardware parts. For instance, the injection sub-system can require software parts to control the injection and hardware parts, the injectors, which execute the injection itself.

Based on the EMS requirements determined during the EMS design phase, the EMS designer determines the requirements to be satisfied by each sub-system. Then, each sub-system is developed separately by taking into account these requirements. In addition, for each sub-system, the software parts are developed separately from the hardware parts.

Once the different sub-systems are developed, the integration phase starts. This consists mainly in integrating the software parts of the different sub-systems together as well as the integration of hardware parts.

Our methodology will intervene during three steps of the current Continental process: the EMS design phase, the software development of each sub-system and the EMS integration phase. The application of our methodology in the context of EMS development will be presented in detail in the next section. Before this, let's present the current approach used at Continental to develop the software of each sub-system.

Sub-system Software development

Currently, there are two approaches for software development at Continental. The first one is purely code-centric approach and the second one is model-based approach. Unlike the code-centric approach where the algorithms are described as Word specifications and then implemented manually using C coding, in the model-based approach the functional design is performed based on Simulink [44] models that describe the defined functions and their associated algorithms. Then, based on these models the C code is generated automatically using a code generator tool. Figure 21 describes the process followed for the two approaches.

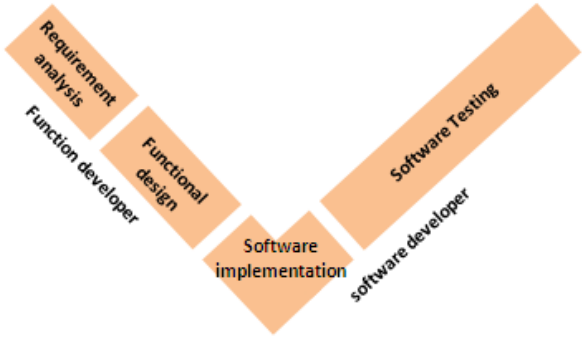

Figure 21 Sub-system software development process

At the beginning of the development process of each approach, the *function developer* starts by analyzing the requirements that should be respected when designing and implementing the needed software. In the next phase, he performs the functional design. This consists mainly in determining the needed functions to ensure the functionality of the sub-system under design and the algorithm to associate to each function. During the software implementation phase, the *software developer* implements these algorithms using C coding.

Software integration

The integration of the software parts from different sub-systems is done by the *software integrator* during the EMS integration phase. After the software integration step, the software integrator performs the software analysis. This analysis consists in:

- Verifying the proper integration of the software by analyzing the static architecture of the integrated system (data communication, input/outputs, etc).
- Verifying the timing behavior of the system by measuring the response times of the OS tasks involved as well as the global CPU load based on the C code of the integrated system.

Software reuse

The process described in figure 21 is completely followed when the software of a sub-system is developed from scratch. However, in order to save the development time and cost, engineers have usually recourse to reuse and adapt previous versions of the software. In software development at Continental, we can distinguish three categories of software reuse:

- Strong reuse: In this case, more than 80% of the new software version is reused from previous version. The modifications concern only some configuration parameters and

variables but the software "core" is not changed. This concerns e.g., the software of engine dependant sub-systems (e.g., engine speed determination sub-system)

- Medium reuse: In this case of reuse, more than 50% of the new software version is reused from previous version. The typical modification that can be done on the software is the introduction of new software modules to ensure new functionalities of the sub-system.
- Weak reuse: In this case of reuse, only the developer expertise and knowledge on previous versions of the software is reused. No software modules are reused from previous versions.

In the case of weak and medium reuse, both the function and software developer are involved and the development process described in figure 21 is completely followed. In the case of strong reuse, no functional design is performed; the software developer works directly on the existing C code to modify the needed parameters and variables.

In the next section, we describe how to apply the proposed methodology in each development case (development from scratch, strong reuse, medium reuse and weak reuse).

1.3. Migration to the New Methodology Process

This section describes how to map the current Continental development process and the process proposed by our methodology. Before describing how our methodology will be applied to develop software in the context of EMS development, let's remind briefly the different activities to be performed during each phase of our proposed development process as described in figure 22.

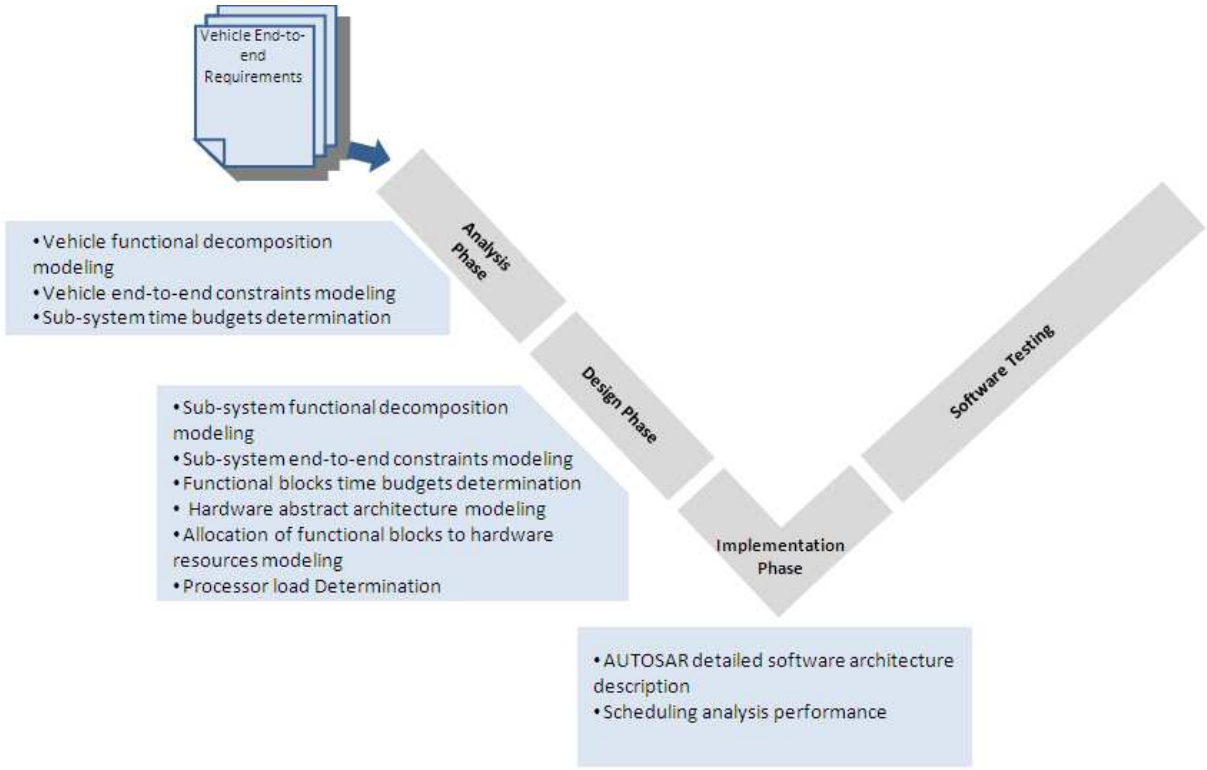

Figure 22 Proposed model-based process

1.3.1. Development from Scratch

Figure 23 shows an overview of our approach to apply the methodology to the development of EMS.

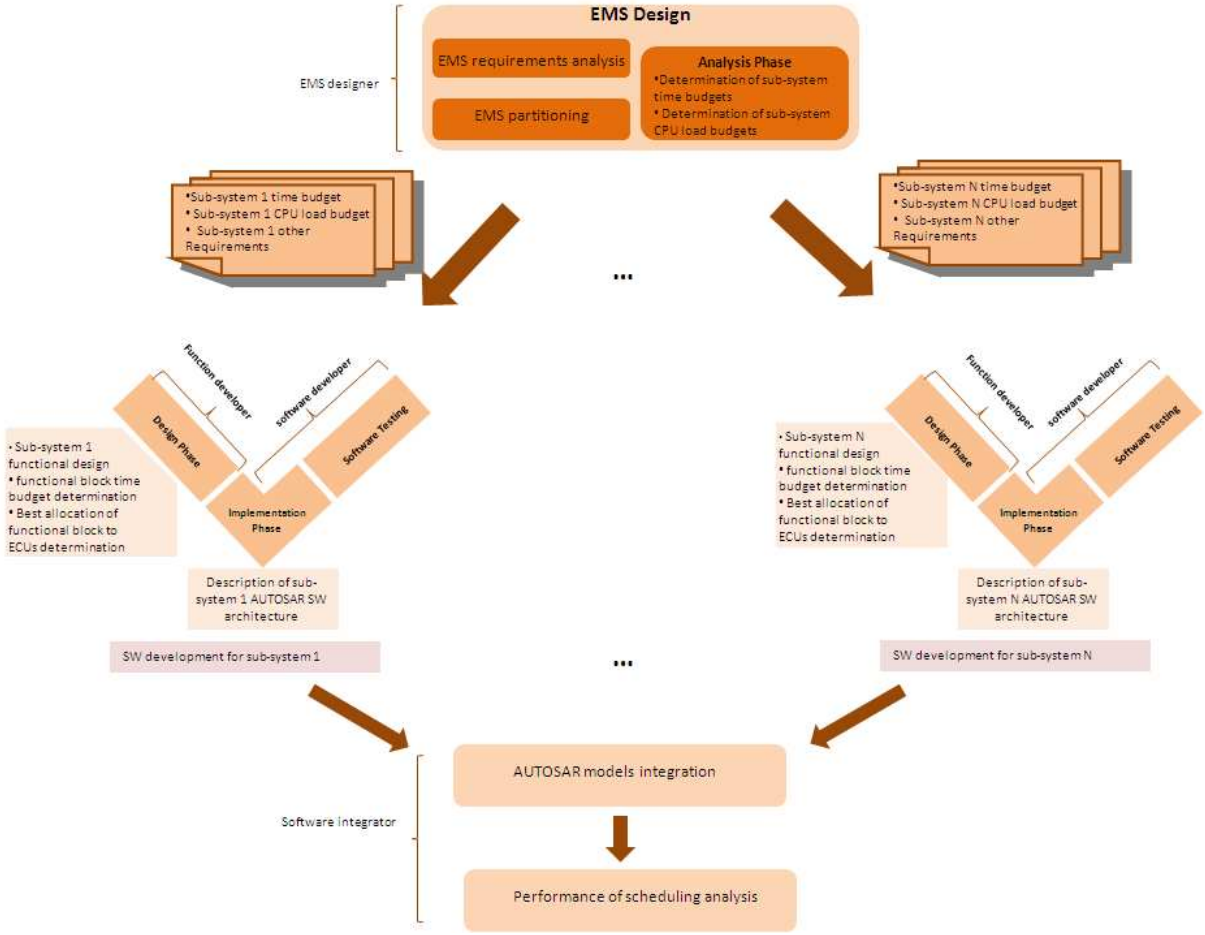

Figure 23 Application of the methodology to develop Engine Management Systems

To develop a whole engine management system, we propose to apply our development process as follows:

- We suggest mapping the activities of the analysis phase of our methodology (described in figure 22) to the EMS design phase of the current development process (described in figure 20). This means that during the EMS design phase described in figure 20, in addition to his/her current task, the EMS designer will determine and analyze what we called in our methodology the vehicle end-to-end requirements. In this case, these requirements will involve some of the sub-systems required for the designed EMS. In the remaining of this chapter, we will hence call these requirements EMS end-to-end requirements instead of vehicle end-to-end requirements. Based on the EMS end-to-end requirements, the system designer, supported by experts from each sub-system, determines the time budgets that should be assigned to each sub-system.

Let's consider, for example, an EMS that contains a sub-system for the calculation of the engine position (engine position determination sub-system). This sub-system transfers the engine position information to an injection sub-system that calculates the instant at which the injection should be performed. An EMS end-to-end requirement can be as follows: "The duration from the start of engine position determination until the injection instant is calculated, should not exceed 500ms". Based on this requirement and other EMS end-to-end requirements, the EMS designer determines hence the time budgets to assign to the engine position determination sub-system and to the injection sub-system. To determine these budgets, the EMS designer will use also his/her expertise related to previous versions of some of the involved sub-systems. This will help him/her to determine the budgets that should be assigned to the sub-systems that are developed from scratch.

As mentioned previously, in the current development process, requirements concerning the global CPU load value of the EMS are considered during the EMS design phase. In our approach, we suggest to determine, based on these requirements, the CPU load requirements for each sub-system. This means that the EMS designer should determine during this phase the CPU budget that can be assigned to each sub-system (e.g., the CPU load requested by the injection sub-system should not exceed 5%).

- We suggest applying the design and implementation phases of our methodology to develop the software of each sub-system. During the design phase of each sub-system, the function developer models the sub-system functional decomposition and determines the functional block time budgets based on the corresponding sub-system time budgets determined previously by the EMS designer. During this same phase, the function developer determines the best allocation scenario of functional blocks to available ECUs. This is done by taking into account the sub-system CPU load budget determined previously by the EMS designer.

  During the implementation phase, the software developer describes the software architecture of each sub-system using AUTOSAR models. Furthermore, based on the functional block time budgets, he determines the timing constraints that should be respected at this level.

- Once the software architecture of each sub system is described using AUTOSAR models, the software integrator will integrate the AUTOSAR models of the different sub-systems. During this phase he performs also scheduling analysis on the integrated system to verify that the timing constraints of each sub-system are respected and that the CPU load constraints are met.

1.3.2. Development by Reuse

As mentioned previously (section 1.2), in the current EMS development process, the software of the sub-systems required by the EMS can be developed by reusing and adapting previous versions of it. In this section, we propose to show how to apply our methodology to develop the software of such sub-systems by reusing the existing artifact of the previous software versions. Table 15 presents the kind of artifacts that are available from a previous software version. In the remaining of this section, we present the application of our methodology in case of strong, medium and weak software reuse.

Table 15. Example of available artifacts from previous software version

Artifact	Description
C code files	The software of each sub-system is organized into software modules. For each software module a C code file is available.
Word specifications	This artifact describes the implementation of each software module
XD models	These models are represented in an internal tool called XD. This tool is used to analyze the static architecture of the software after EMS integration. The software of each sub-system is represented by a number of software modules. Each software module is composed of a number of operations which represent the smallest executable code fragment.
Timing data base	This artifact contains the timing information of the integrated system. This information consists mainly in operation execution times, OS task response times and CPU utilization values. These data are measured by an internal tool using the C code of the integrated system

Case of strong reuse:

As mentioned previously, in this case of reuse, to develop the new version of the software, all the software modules are reused from the previous version. The modifications done on the new version are minor and concern only e.g., parameters or variable names modification. Hence, in this case of reuse, we do not need to perform the activities of the analysis and design phases of our methodology. To enable this case of reuse by using our methodology, we suggest then to work directly on the implementation phase by transforming the legacy information represented in the XD model (see table 15) of the previous software to an AUTOSAR architecture. Figure 24 shows an overview of our approach.

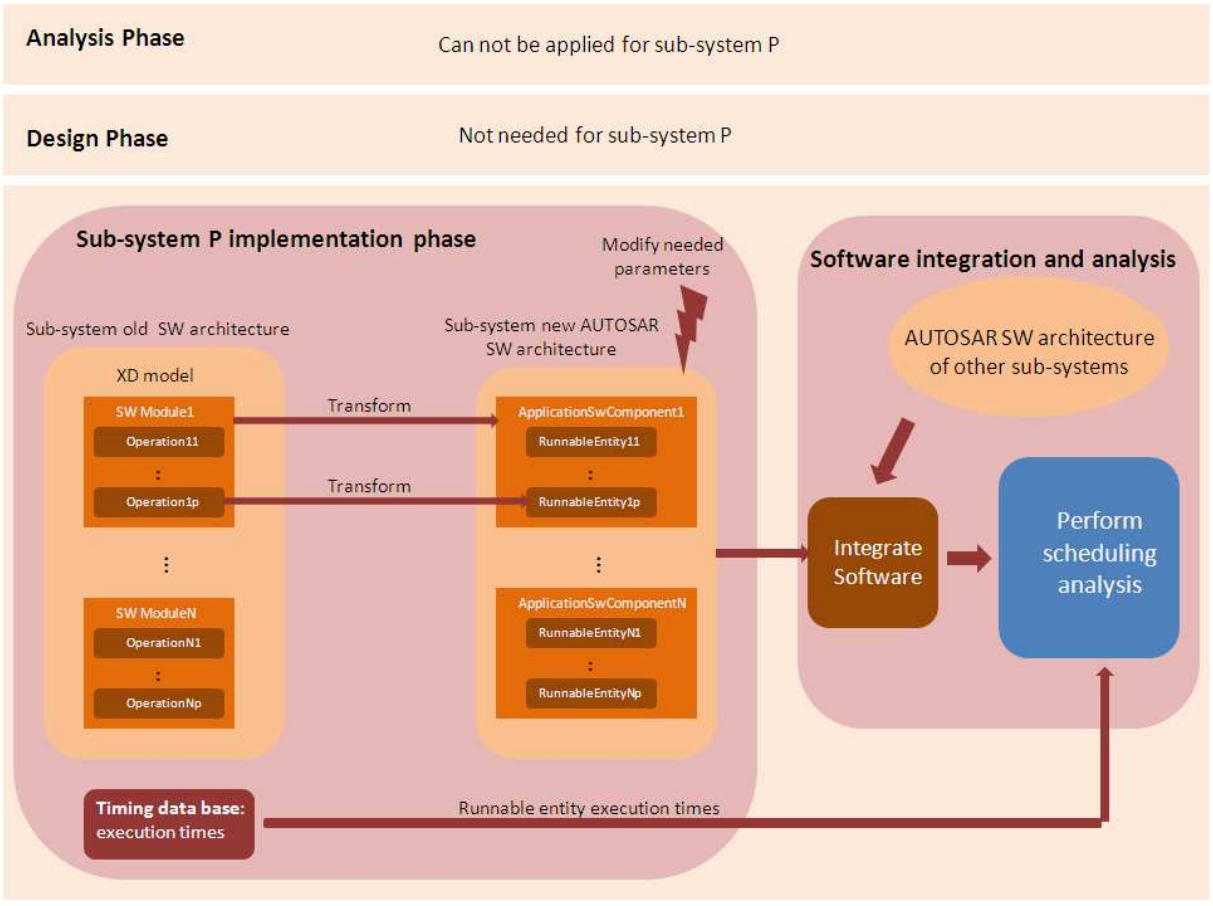

Figure 24 Application of the methodology in case of strong software reuse

Let's consider a sub-system P as described in figure 24. To develop the new version of the software of this sub-system, we do not need to determine the time budget and CPU load budget to assign to it during analysis phase. In fact these budgets should be already known (or at least can be estimated directly) from the previous software version. As the software

architecture is already available, to develop the software of the sub-system P the design phase is also not needed.

The transformation of the XD model to an AUTOSAR model should be done as follows: each software module is transformed to an AUTOSAR software component. The operations of each software module are transformed into runnable entities. Once the AUTOSAR software architecture of sub-system P is described, the software integrator integrates it with the AUTOSAR models of the software of other involved sub-systems (the software of other sub-systems is developed in the same way either by reuse or from scratch as described previously). The timing data base containing the execution times of the sub-system P operations will be used to specify the execution times of the runnable entities to enable performing scheduling analysis.

Case of weak reuse

In this case of reuse, no software modules can be reused from previous version. Hence, all the methodology phases should be applied for the development of the software in the same way as for the development from scratch. However, when performing the scheduling analysis on the integrated system, the expertise of the software integrator from the previous versions of the sub-system can be used to estimate the execution times of the runnable entities of the sub-system considered.

Case of medium reuse

In this case of reuse, we will focus on the case of adding new functions or software modules to the previous software version. We have to distinguish, then, two cases:

- If the new function or software module will interact with other sub-systems in a way that there are EMS end-to-end requirements that involve these sub-systems and the one under development, then the methodology should be applied starting from the analysis phase. This is needed to determine the new time budget to assign to the considered sub-system with this new configuration.

- If the new function or software module will interfere only internally with other software modules within the same sub-system, then the methodology can be applied starting from the design phase. The time budget to be assigned to the sub-system can be estimated directly based on the previous software version.

Figure 25 shows the approach followed during design and implementation phases for these two cases. As the figure shows, based on the XD model that describes the previous software modules and their operations, the function developer transforms during the design phase each software module into functional block. He defines then the new functional blocks needed for the new version of the sub-system. Based on this new configuration and the time budget known for the sub-system previous version, the function developer determines the time budget to assign to each functional block of the new configuration. Based on the new functional architecture, the software developer describes the new software architecture using AUTOSAR constructs. The simplest way is to transform each functional block defined at the design level to an AUTOSAR software component at the implementation level. The definition of the runnable entities for each software component is done by taking into account the information from the previous software architecture but also the new constraints on the software.

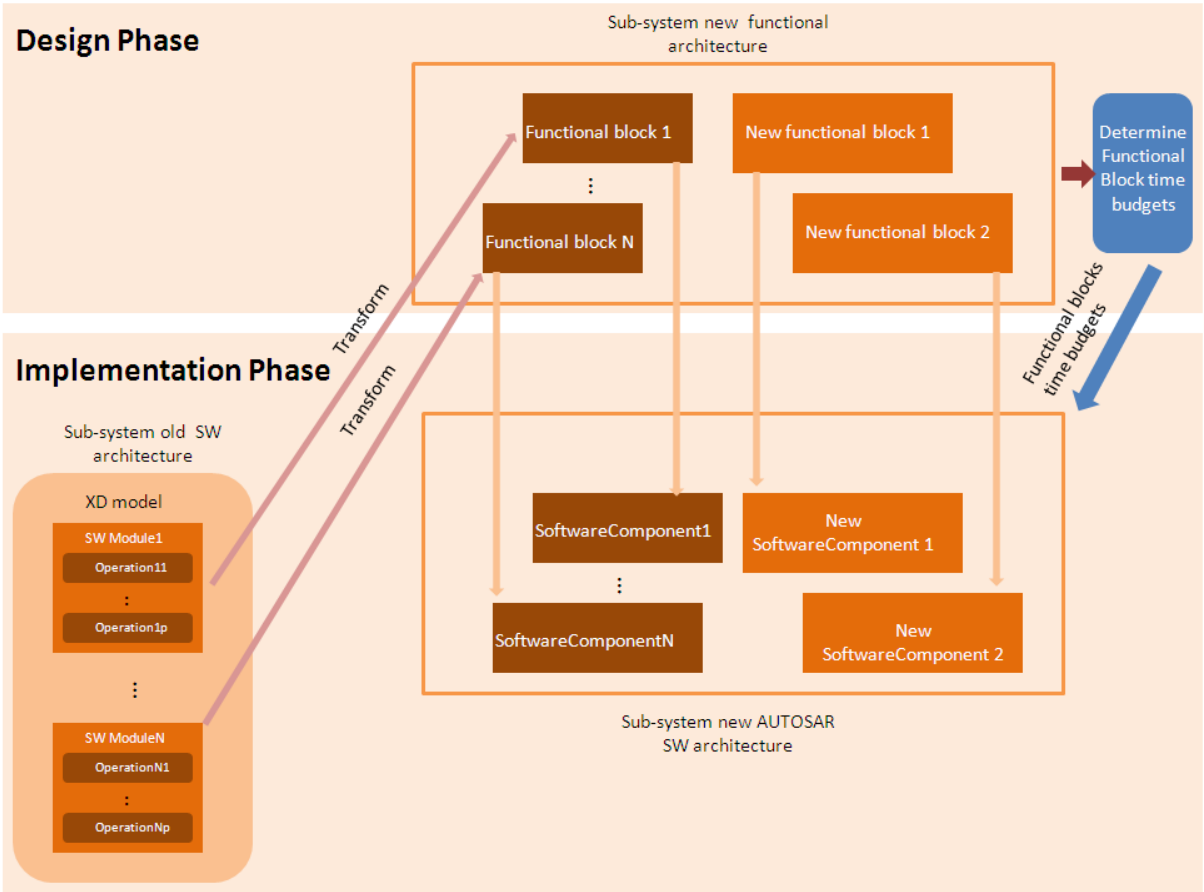

Figure 25 Application of the methodology in case of medium software reuse

In the next chapter, we present an example of the application of our methodology for the development of two sub-systems, the cruise control (development from scratch) and the knock sub-system (development by medium reuse).

2. Examples

This chapter provides an illustration of the application of our methodology to the development of engine management sub-systems. The first section presents an example of the application of the methodology to develop the cruise control sub-system from scratch. The second section deals with the scenario of development by medium reuse and considers the knock sub-system as use case. To develop the models of these two use cases; we used the Papyrus tool [42] to develop the models at the analysis and design levels and the Cessar-CT tool [43] for the models of the implementation level.

2.1. Development from Scratch: Cruise Control

2.1.1. Use Case Presentation

The application considered is the cruise control function. It is used to maintain vehicle speed at a speed setpoint desired by the driver. This functionality calls for a switch sensor that acquires the driver inputs (set cruise, cancel cruise, increase speed setpoint, etc.) and a control system that processes inputs from this sensor and other EMS sub-systems (e.g. braking sub-system) to calculate the speed setpoint and send a torque request to the torque setpoint sub-system. In this section, we show how to apply the proposed methodology to develop and analyze the software of the cruise control sub-system. In subsequent sections of this chapter, we refer to this sub-system as "cruise control".

2.1.2. Analysis Phase

In this phase, based on the given timing requirements (EMS end-to-end requirements), we determine the time budgets to be allocated to each sub-system involved in these requirements. Here we focus on the cruise control sub-system and we consider that the time budgets of other sub-systems that communicate with the cruise control are already known based on information from previous developments of these sub-systems.

2.1.2.1. EMS End-to-end Requirements

We determined two EMS end-to-end requirements to be satisfied when designing the cruise control sub-system. These have been denoted as EMS_REQ1 and EMS_REQ2 (REQ for requirement).

- **EMS_REQ1**: When the driver depresses the braking pedal, cruise control should be deactivated within 300ms.

- **EMS_REQ2**: When the driver activates cruise control, the vehicle speed setpoint should be calculated and displayed within 500ms

These two requirements concern the cruise control sub-system and other sub-systems such as the brake controller sub-system, which receives inputs from the pedal sensor indicating the status of the pedal (depressed or not) and the display actuator that receives inputs from several vehicle functions for display. In the next step, we determine the time budgets to be allocated to the cruise control in order to satisfy these two requirements. First, we need to develop a model containing the information necessary for this timing analysis.

2.1.2.2. Analyzable Model

As stated earlier, the analyzable model comprises two views, the analysis functional view and the analysis timing view.

- **Cruise Control Analysis Functional View**: Figure 26 shows the model developed for this view. This model depicts a functional decomposition of the EMS focusing on the interaction of the cruise control with other EMS sub-systems. In it, the cruise control sub-system (called "CruiseControl" in the figure) is communicating with the brake controller sub-system, the torque setpoint sub-system, the display actuator and the switch sensor that acquires the driver inputs. As the figure also shows, EAST-ADL concepts are used here; the cruise control sub-system and the other sub-systems are modelled as *"AnalysisFunctionTypes"*. Sensors and actuators are modelled as *"FunctionalDevices"*, an EAST-ADL concept that represents the functional part of a sensor or an actuator. The interaction between different sub-systems is modelled using EAST-ADL connectors called *"FunctionConnectors"*.

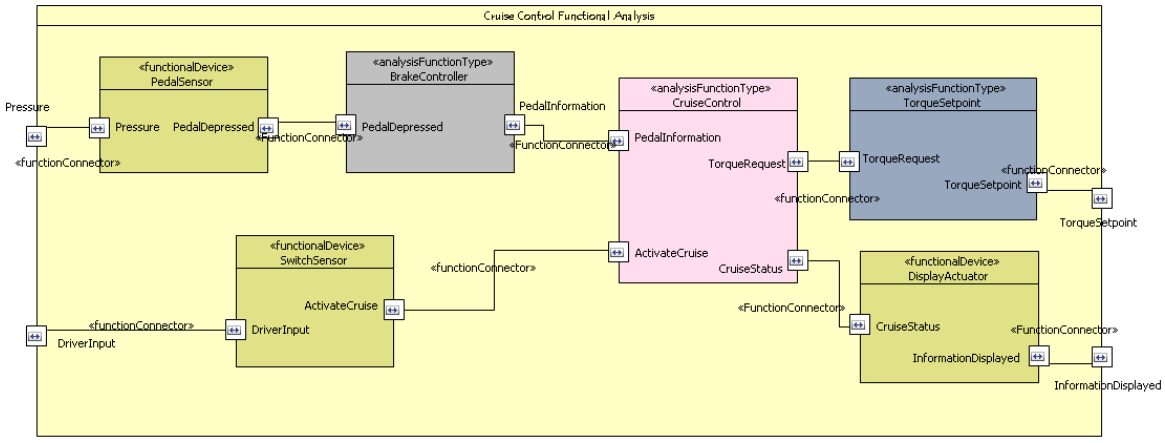

Figure 26 Cruise control analysis functional view

The end-to-end requirement EMS_REQ1 means that from the point in time at which the pedal sensor receives a pressure until the point in time the torque setpoint calculates a null torque setpoint, the time elapsed should not exceed 300ms. EMS_REQ2 means that since the switch sensor receives the driver input ordering activation of cruise control until the speed setpoint is calculated by cruise control and then displayed by the display actuator, the time elapsed should not exceed 500ms.

To determine the time budgets to be allocated to the cruise control, we developed a timing view in which these timing constraints are expressed in the model using TADL concepts.

- **Cruise Control Analysis Timing View**: Figure 27a and 28a show sequence diagrams representing the cruise control analysis timing view. For each EMS end-to-end requirement, we model an interaction that we stereotype with *"EventChain"*. Each event chain is made up of sub-chains that represent the execution of the functions involved in the interaction and the transfer of data-based messages between these functions. This way, each action execution specification and each message are modelled as sub-chains (stereotyped with *"EventChain"* and specified as *"EventChainSegment"* for the whole interaction event chain). As shown in Figure 27b and 28b, to express each end-to-end requirement, we specify for each *"EventChain"* a TADL *"ReactionConstraint"* for which we specify a *"TimeDuration"*. The latter enables to specify the upper value of the reaction constraint (For example, for the first event chain, we specify a reaction constraint called cruise_deactivation_delay. For this reaction constraint, we describe a time duration of 300ms as an upper bound value). To support data-based communication, each message is also stereotyped as *"DataMessage"*. Events associated to the sending and receiving of these massages are stereotyped respectively as *"SendDataEvent"* and *"RecieveDataEvent"* and also as TADL events. For each event chain involved in the interaction stimulus and response events are specified.

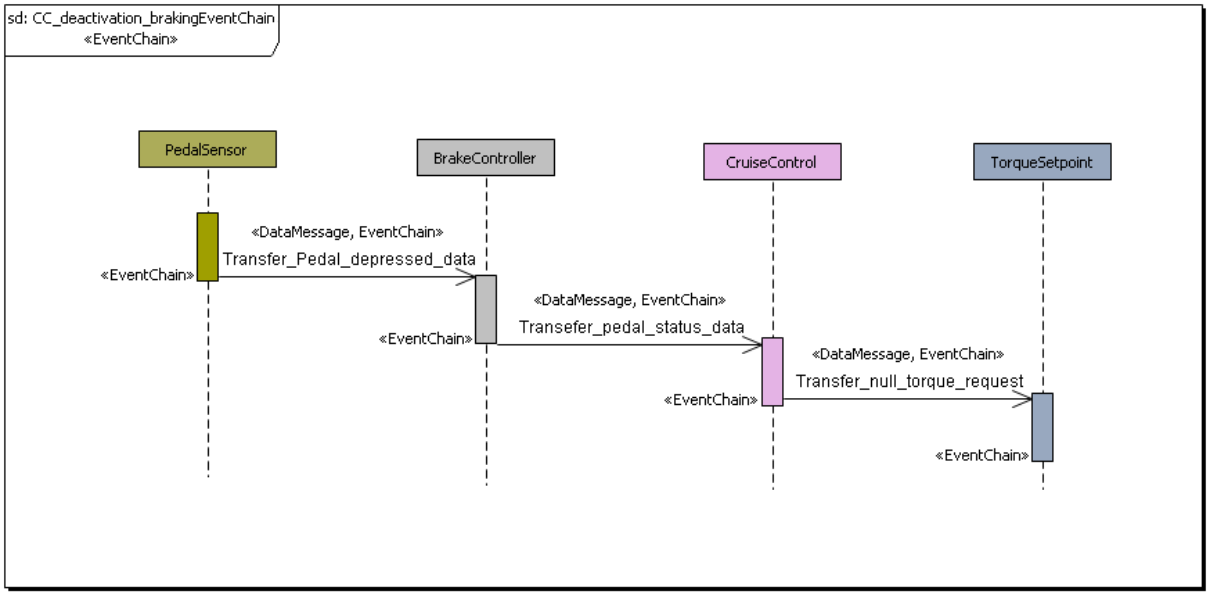

Figure 27a Cruise Control analysis timing view, deactivation event chain

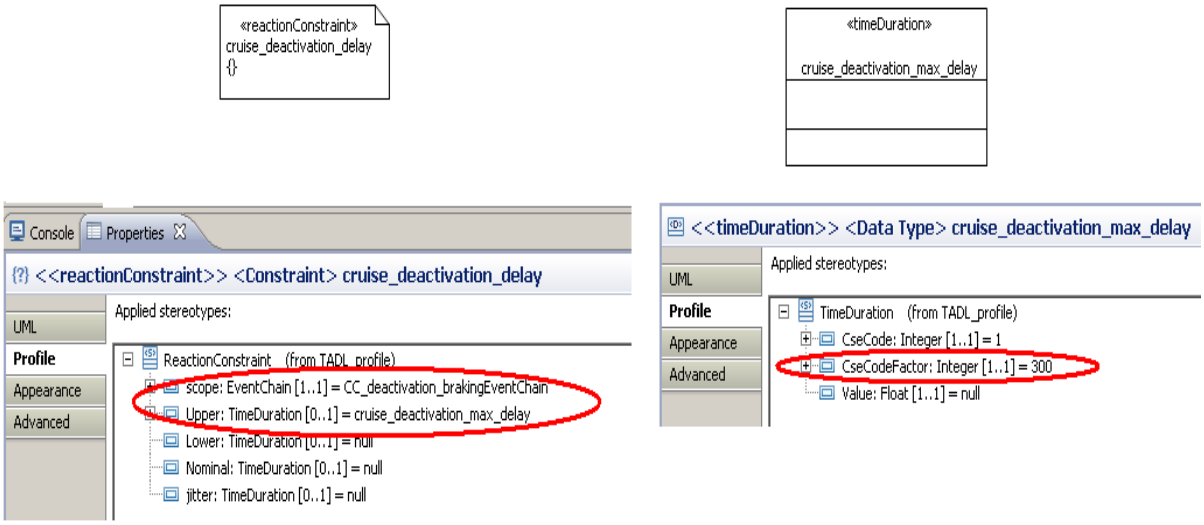

Figure 27b Specification of timing constraint for the deactivation event chain

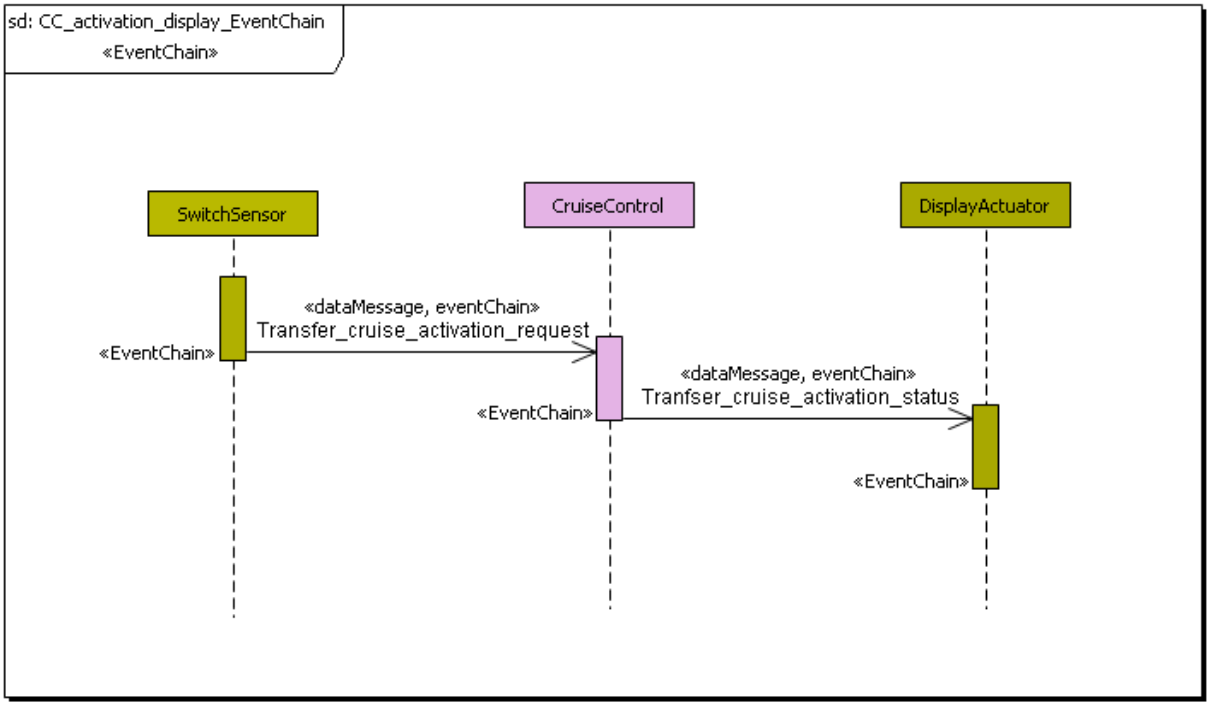

Figure 28a Cruise Control analysis timing view, activation event chain

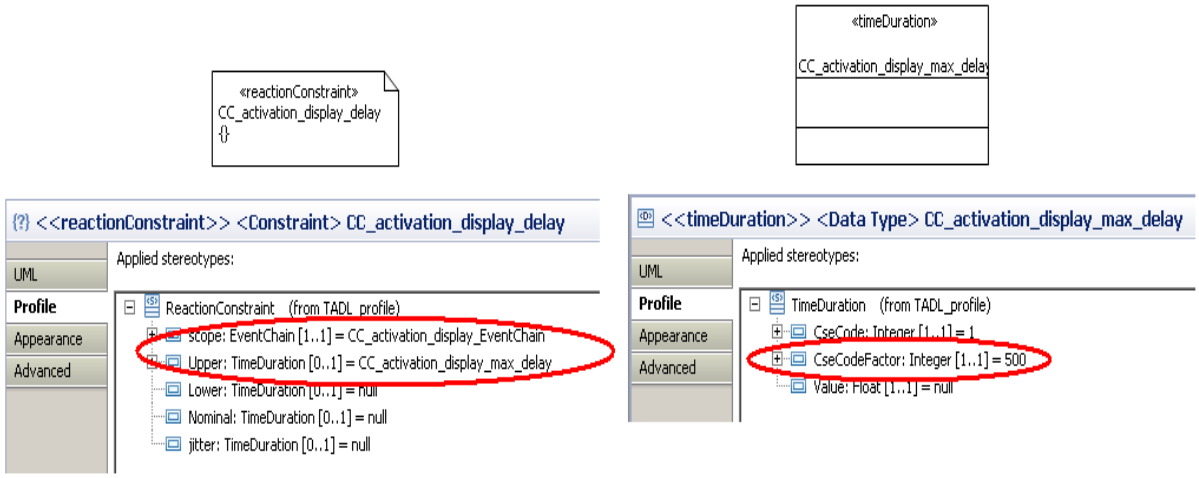

Figure 28b Specification of the timing constraints for the activation event chain

2.1.2.3. Cruise Control Time Budgets

As mentioned during the methodology description, we consider that the time budgets for the other EMS sub-systems are already known. Determining the time budgets for cruise control is then quite easy at this stage. In our example, with the help of application experts, we managed to manually determine the following time budgets, which satisfy the two previously listed end-to-end requirements. To ensure compliance with EMS_REQ1 (and taking into account the time budgets of the pedal sensor, the brake controller and the torque

setpoint sub-systems) we should allocate 100ms to cruise control deactivation. To ensure compliance with EMS_REQ2 (and taking into account the time budgets of the switch sensor and the display actuator), we should allocate 200ms to cruise control activation and speed setpoint calculation. Hence, we have the following two constraints to be satisfied when refining the cruise control functional architecture during the design phase:

- **AConst1**: Cruise control should be deactivated within 100ms.
- **AConst2**: Cruise control should be activated and speed setpoint calculated within 200ms.

2.1.3. Design Phase

In this phase, we refine the functional architecture of the cruise control by showing its functional breakdown into functional blocks. The first timing analysis performed is refinement of the time budgets determined in the analysis phase by determining the time budget to be allocated to each functional block.

2.1.3.1. Refinement of Cruise Control Time Budgets

To refine the time budgets determined during the analysis phase, we first develop an analyzable model of cruise control.

A. Analyzable Model

In the same way as for the analysis phase, the model is composed of two views:

- **Cruise Control Design Functional View**: Figure 29 shows the functional breakdown of the cruise control sub-system. We broke down the sub-system into four functional blocks: Input acquisition and interpretation is responsible for the acquisition of inputs from the switch sensor and other sub-systems and their interpretation, to deduce the desired action (activate cruise, cancel cruise, etc). Failure management is responsible for diagnosis of the cruise control inputs and limp home activation (the limp home function decides which action to take if an error is detected). Speed setpoint calculation is responsible for calculation of the desired speed setpoint. Control is responsible for calculation of the cruise control states and transitions and maintaining speed at the speed setpoint. As Figure 29 shows, the *"CruiseControl" "AnalysisFunctionType"* modelled in the analysis phase is realized here by a *"DesignFunctionType"* also called *"CruiseControl"*. Each Functional Block is

modelled as a *"DesignFunctionPrototype"* that represents an instance of a *"DesignFunctionType"*.

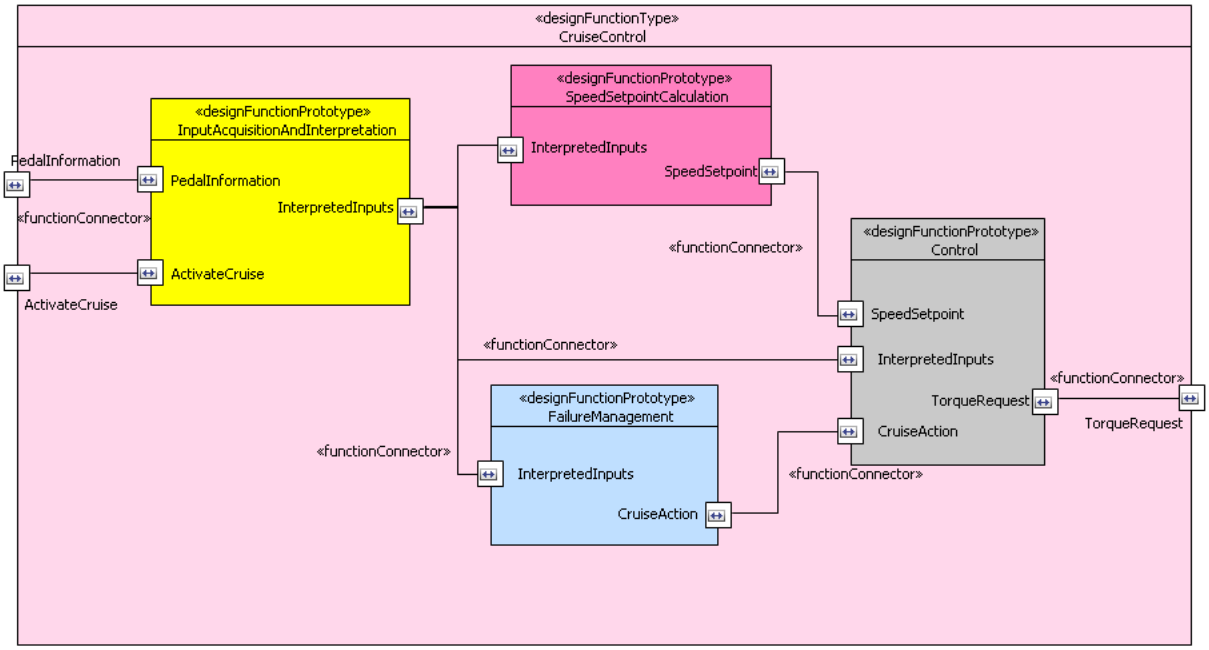

Figure 29 Cruise control design functional view

- **Cruise Control Design Timing View**: AConst1 means that from instant at which the pedal information input is acquired by the input acquisition and interpretation function until the control functional block orders a null torque request, the time elapsed should not exceed 100ms. AConst2 means that from the time of acquisition of the "activate cruise" input until calculation of the setpoint by the speed setpoint calculation and then activation of cruise control by the control functional block, the time elapsed should not exceed 200ms. To ensure the safety of the driver, a new constraint is introduced at this stage, to ensure that, if a failure is detected, cruise control is deactivated within 100ms (Aconst3). This means that from the instant at which inputs are acquired and interpreted until the detection of failure and deactivation of cruise control, the time elapsed should not exceed 100ms. Figure 30a, 31a and 32a show the sequence diagrams developed for the timing view. The first diagram shows the communication between the functional blocks involved in AConst1 (i.e. deactivation of cruise control). The second diagram shows communication between the functional blocks involved in AConst2 (i.e. activation of cruise control and calculation of the speed setpoint). The third diagram shows communication between the functional blocks involved in AConst3. In the same way

as for the analysis phase, for each event chain, we specify the corresponding reaction constraint as shown by figure 30b, 31b and 32b.

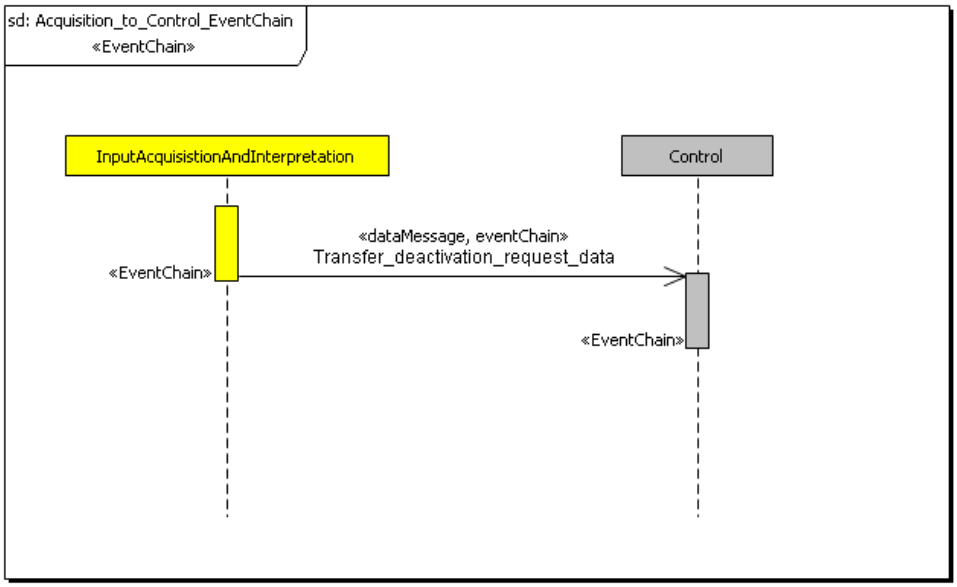

Figure 30a Cruise Control design timing view, "acquisition to control" event chain

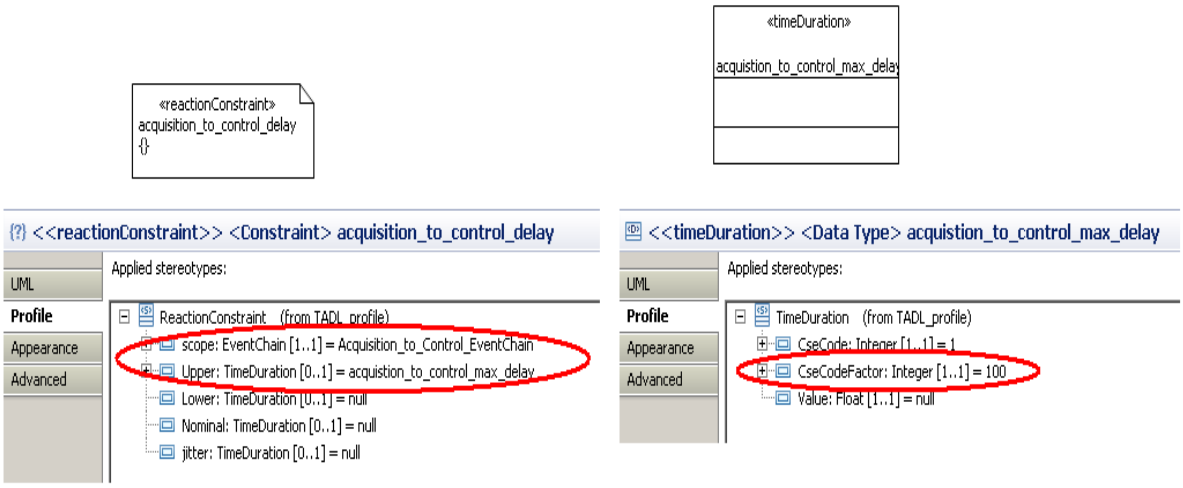

Figure 30b specification of the timing constraints to the "acquisition to control" event chain

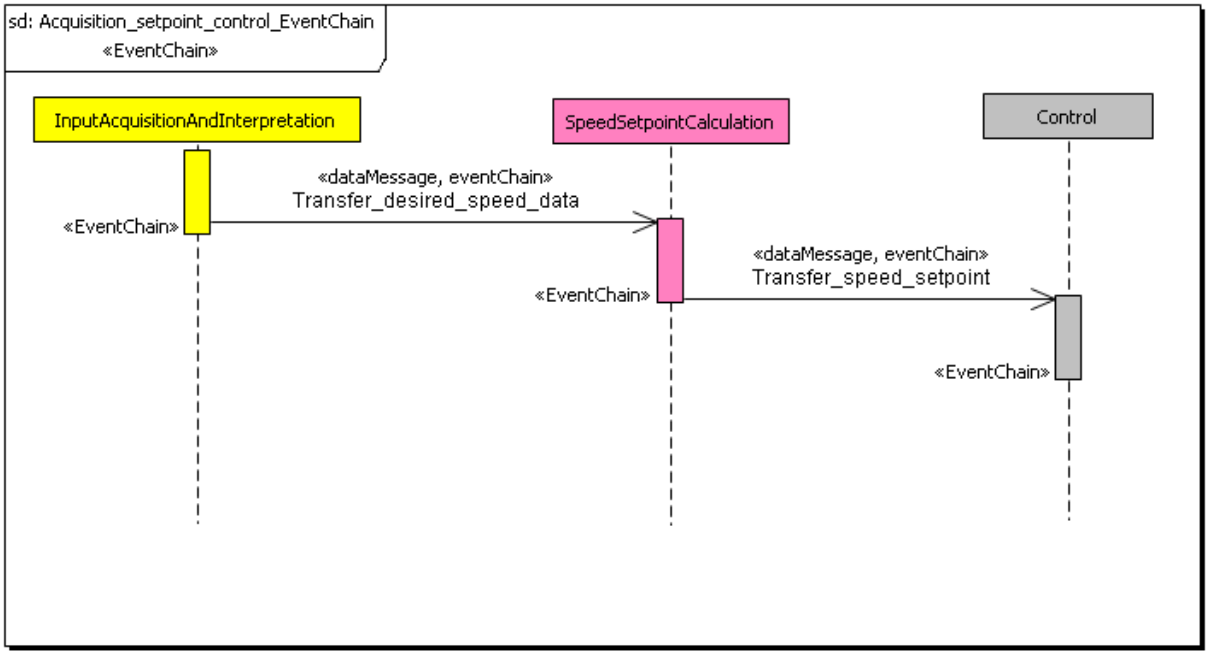

Figure 31a Cruise Control design timing view, "acquisition setpoint control" event chain

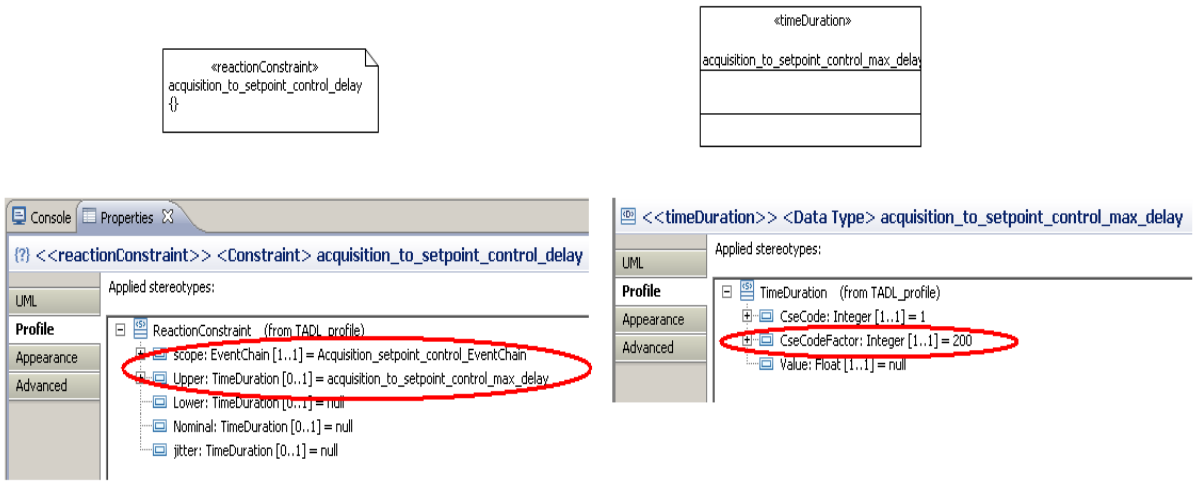

Figure 31b Specification of the timing constraint for the acquisition setpoint control event chain

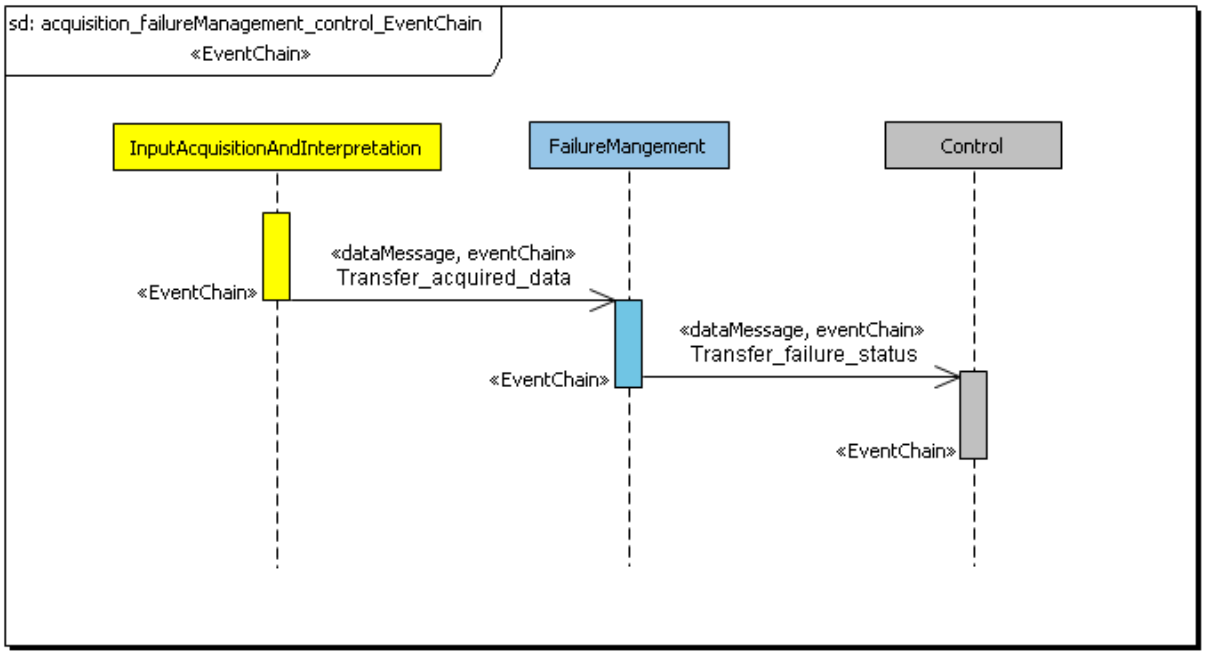

Figure 32a Cruise Control design timing view, failure event chain

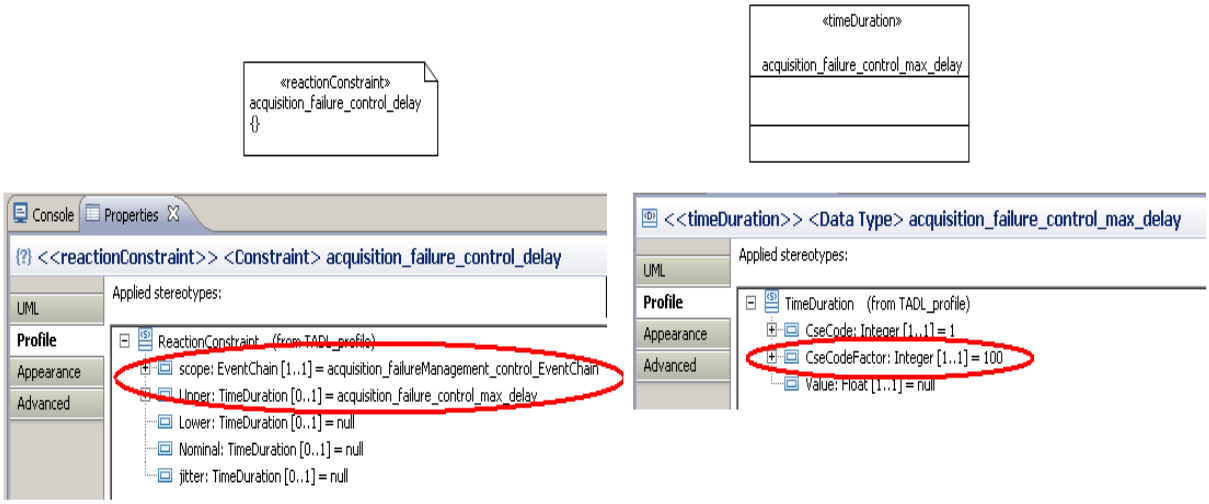

Figure 32b Specifying timing constraint to "acquisition failure control" event chain

B. Functional Block Time Budgets

We determined manually the time budgets to be allocated to each functional block for compliance with the three constraints mentioned previously. The time budgets that can be allocated to the various functional blocks are as follows:

- Input acquisition and interpretation: 30ms
- Failure management: 20ms

- Control: 50ms
- Speed setpoint calculation: 90ms

This means that, when refining the cruise control architecture at the implementation phase, we should respect the following timing constraints:

- **DConst1**: Input acquisition and interpretation should be performed within 30ms
- **DConst2**: The failure management should take place within 20ms
- **DConst3**: Control should take place within 50ms
- **DConst4**: Speed setpoint calculation should take place within 90ms

The next step is to explore the hardware architecture to determine the best allocation of cruise control functional blocks to the available hardware resources. This is done based on an evaluation of load for each processor.

2.1.3.2. Hardware Architecture Exploration

In this step, we explore the available hardware architecture, for the purpose of deciding which hardware resources to select and how to efficiently distribute cruise control functional blocks over these resources. In our case, the cruise control functions can be distributed between the engine management ECU (EMS ECU) and the body controller ECU. These two ECUs communicate via a CAN bus. Based on the load evaluation for each ECU, we determine the best functional block-to-ECU allocation scenario. This means first developing a model containing the information necessary for the analysis.

A. Analyzable Model

As explained in the description of our methodology, the model is made up of three views:

- **Cruise Control Design Functional View**: This is the view described in Figure 26. In this view, we specified the execution time of each functional block. Execution times were estimated with the help of application experts at Continental. We thus determined the following execution times:
  - Input acquisition and interpretation: 80µs
  - Failure management: 100 µs
  - Speed setpoint calculation: 120 µs

- Control: 200 µs

The EAST-ADL concept *"ExecutionTimeConstraint"* enables specification of these times for each functional block.

- **Hardware Platform View**: Figure 33 shows the hardware platform model, which we developed using a UML composite diagram. Each computation hardware resource is modelled as a *"Node"* (EAST-ADL) and a *"SaExecHost"* (MARTE) to represent resources with processing capacity that can host executable elements. The CAN bus is stereotyped by *"LogicalBus"* (EAST-ADL) and *"SaCommHost"* (MARTE). The aim of using MARTE concepts here is to enable the use of the automatic transformation implemented for MARTE models to analyze the system using the MAST tool.

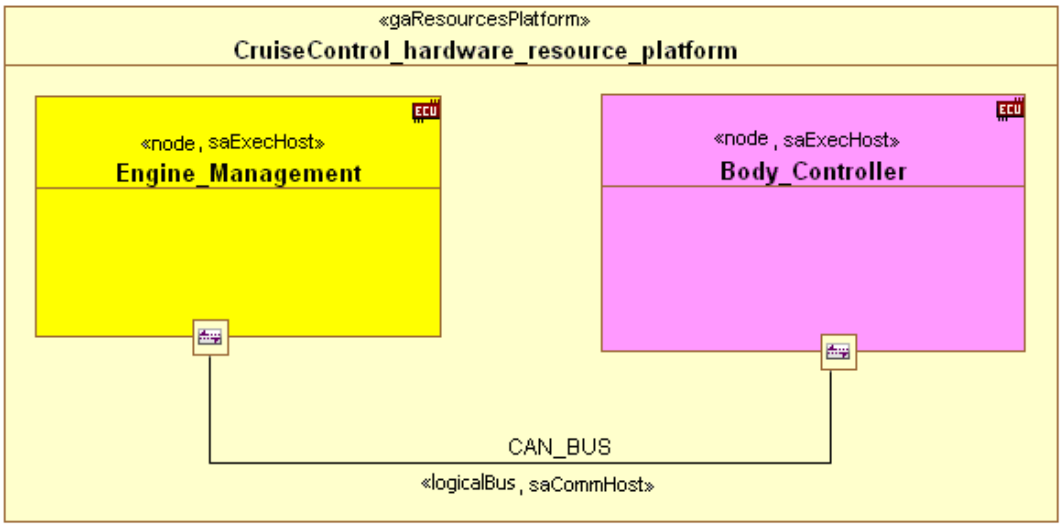

Figure 33 Hardware platform view

Using this view, we choose the best scenario for the allocation of functional blocks to the hardware platform. The best scenario is chosen according to the load requested by the functional blocks for each ECU. To do so, we model the allocation view that represents the allocation scenario to be analyzed.

- **Allocation View**: When allocating the functional blocks to the hardware platform, we must satisfy certain requirements identified with the help of vehicle dynamic architecture specialists at Continental (when applying the methodology to the development of EMS, these budget should be determined by the EMS designer during the EMS design phase):
  - The load requested by cruise control functions from the Body controller ECU should not exceed 1%

- The load requested by cruise control functions from the engine management ECU should not exceed 2% (these values are determined by taking into account the load budgets of other sub-systems that will be allocated to these ECUs).
- The failure management and the control functional block should be allocated to the same ECU to ensure speed reaction of the control system when an error is detected.

Figure 34 shows the modelling of a functional block-to-available ECU allocation scenario. Here we chose to allocate input acquisition and interpretation to the body controller ECU and the rest of the functional blocks to the engine management ECU. MARTE concepts for allocation are used in this view. Each functional block is stereotyped by *"allocated"*, a concept that allows specification of the hardware resource hosting the functional block. Moreover, a dependency connector is drawn between each functional block and its hosting ECU and is stereotyped by *"allocate"*.

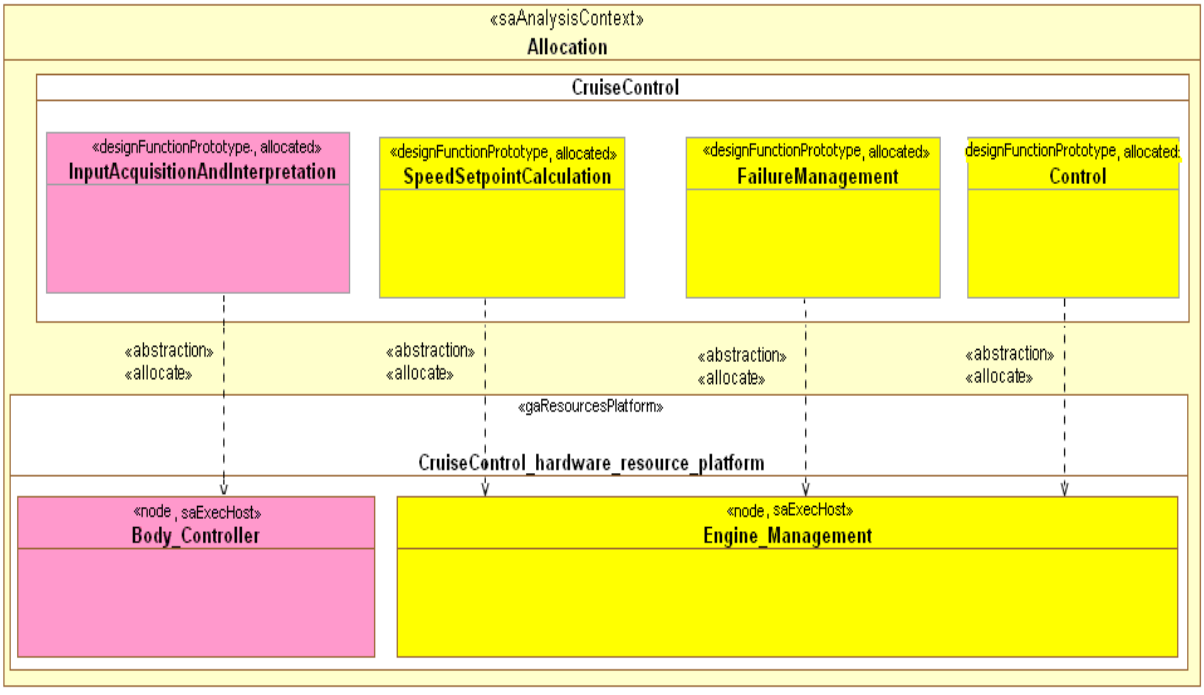

Figure 34 Allocation view

B. Processor Loads Determination

Based on the allocation scenario shown and the execution times annotated on the design functional view, we used the scheduling analysis tool MAST [31] (for which an automatic transformation from MARTE models is implemented [60]) to evaluate the load requested by the functional blocks allocated to each processor. Each functional block is transformed

into a schedulable resource hosting only one operation in the MAST model. In scheduling analysis, processor utilization is calculated based on two kinds of parameters which are the tasks/functions execution times and the tasks/functions periods regardless of the priorities that are assigned to the executed tasks. Hence, task priorities are not important for our analysis (evaluation of processor loads). For this reason, we assume that all schedulable resources have the same priority in our example (this assumption is used only for this analysis).

For each schedulable resource a transaction is defined in MAST. For each operation, we assigned the execution time determined for the corresponding functional block. To be able to calculate processor loads, we also need to specify the triggering period of each transaction. With the help of cruise control application experts at continental, we assigned 10ms as the period for both input acquisition/interpretation and failure management, and 40ms as the period for both speed setpoint calculation and control. Based on this information and the allocation scenario chosen, the tool calculated an utilization of 0.8 % of the body controller ECU and 1.8 % of the engine management ECU by the cruise control functions. These results meet the requirements listed above (other tested allocation scenarios did not meet these requirements). We therefore kept this allocation scenario as the best scenario. This is the scenario to be satisfied when refining the cruise control architecture at the implementation stage, especially when describing the mapping of runnable entities to OS tasks. Let's note that to calculate the utilization of each ECU by the cruise control functions, we do not model the other vehicle functions executed on these ECU. That is why, in our model (figure 31), we represent only the allocation of the cruise control functions to these ECUs (without considering other vehicle functions executed on the same ECUs).

2.1.4. Implementation Phase

In this phase, the cruise control functional architecture is refined and transformed into AUTOSAR architecture described using software components and runnable entities. To develop the analyzable model, we used the CESSAR-CT tool, this is an AUTOSAR workbench developed by the Continental Engineering Services. This tool is based on the ARTOP (AUTOSAR Tool Platform) framework [43], an implementation of common base functionality for AUTOSAR development tools.

2.1.4.1. Analyzable Model

Cruise Control Application View: We developed this view by transforming the cruise control design functional view into (an) AUTOSAR model. Figure 35 shows an overview of the AUTOSAR application view that we developed (As the tool used does not offer graphical views of the model developed, the figures of this section present a simplified overview of the models, for the clarity of the figure we do not show all the data exchanged between the software components). Each functional block is transformed into an application software component for which we describe the behaviour by specifying the runnable entities and their triggering events. The communication between these software components is modelled through AUTOSAR ports called *"PPortProtyotype"* for provided data and *"RPortPrototype"* for required data.

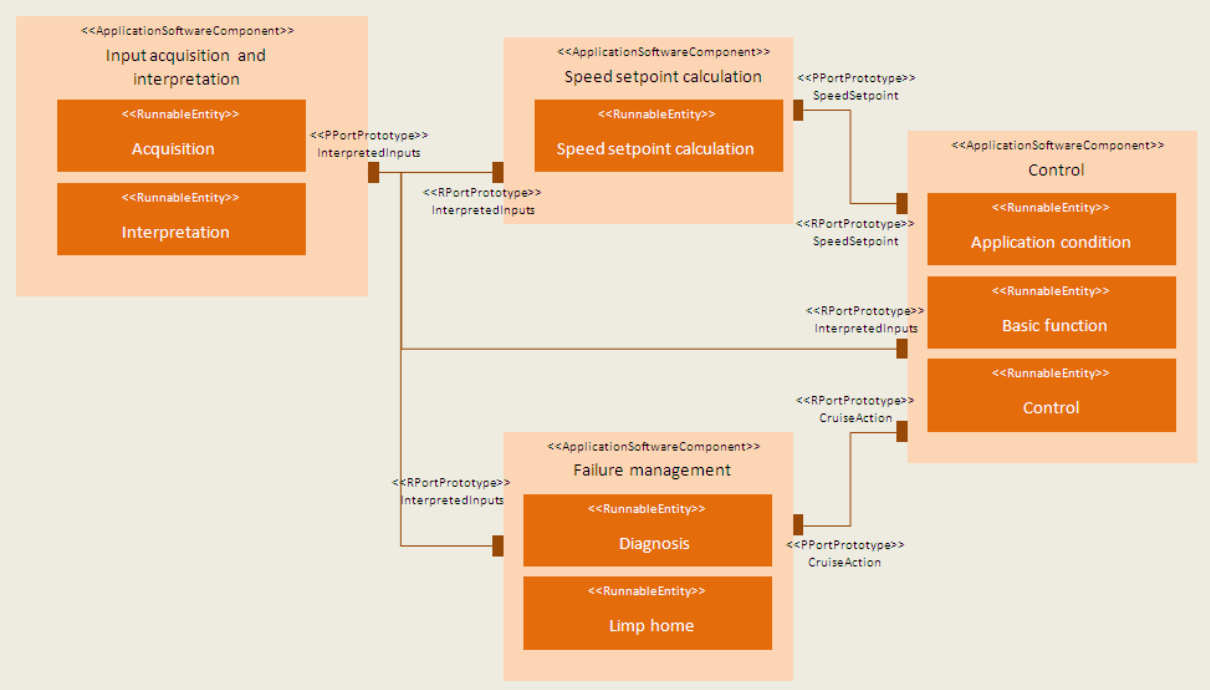

Figure 35 Simplified overview of the cruise control application view

Software component input acquisition and interpretation involves two runnable entities – acquisition and interpretation. Speed setpoint calculation requires only one runnable for the calculation of the speed setpoint. Failure management involves two runnable entities, the first to perform diagnosis of the inputs and the second to decide what action to take in case of error. The control software component is made up of three runnable entities: its application condition and basic function, which calculate the cruise control states and transitions to decide whether to carry out specific cruise control activities; and its controller, which is a PI controller that maintains vehicle speed.

For each runnable entity, we defined an *"RTEEvent"* that defines the triggering of the runnable. For example, as we chose to execute failure management each 10ms, we specified, for diagnosis and limp home runnable entities, an *"RTEEvent"* with 10ms as its period.

For each software component, we also specified a software component implementation that allows us to set the execution time of each runnable entity. This is done using the AUTOSAR concept *"resource consumption"*, which describes the necessary resource in terms of execution time for each runnable entity. Table 16 shows the runnable entity execution times that we determined taking into account the execution times determined previously for the functional blocks and with help of cruise control function experts.

Table 16 Determination of Cruise Control runnable execution times

Functional block	**Execution time (µs)**	**Constraints & Comments** (based on discussion with cruise control experts)	**Runnable Entities**	**Execution time (µs)**
Input acquisition and interpretation	80	Input interpretation needs more time to execute than input acquisition	Acquisition	30
			Interpretation	50
Failure management	100	diagnosis needs more time to execute than limp home	Diagnosis	60
			Limp home	40
Speed setpoint calculation	120	This functional block is transformed to a software component with only one runnable then the runnable has the same execution time as the functional block	Speed setpoint calculation	120

Control	200	the basic function runnable entity should have the largest execution time, application condition runnable entity has the smallest execution time	Application condition	40
			Basic function	100
			Controller	60

As shown in table 16, based on discussion with cruise control function experts, we determined the constraints that should be respected when assigning the execution times for the defined runnable entities. Based on these constraints, we determined an execution time value to each runnable entity.

- **Cruise Control Timing Behaviour View**: In this view, we modelled the timing behaviour of cruise control by means of events and event chains from the AUTOSAR timing extensions. For each timing constraint determined at the design stage (DConst1, DConst2, DConst3 and DConst4), we created an event chain for which we specified a latency constraint. As an example, for DConst1 (figure 36), we created an event chain having the activation event of the runnable entity acquisition as a stimulus and the termination event of the runnable entity interpretation as a response. For this event chain, we specified a latency constraint with 30ms as maximum value. For DConst4, we created an event chain formed by the runnable entity speed setpoint calculation. We specified for this event chain a latency constraint with 90ms as maximum value.

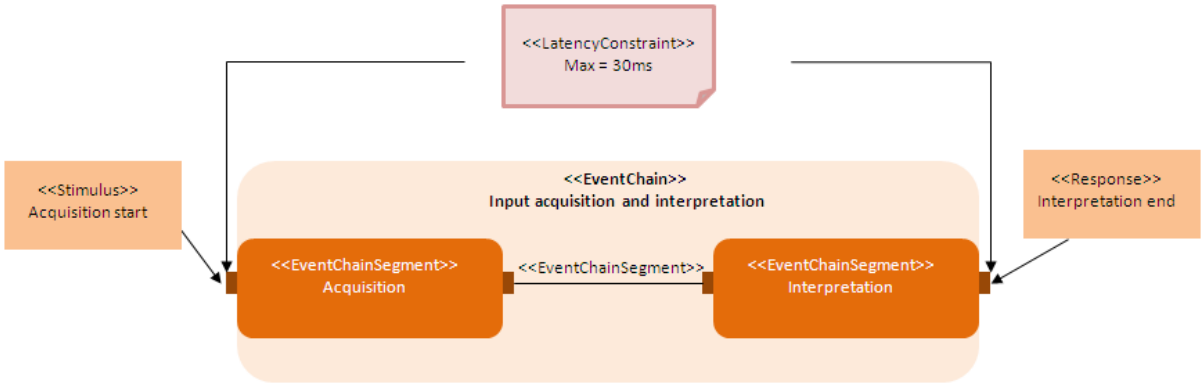

Figure 36 Representation of the modelling of the acquisition event chain

- **Resource Platform View**: In this view, we model the hardware and software resources used by the sub-system. Each ECU is modelled as AUTOSAR *"ECU instance"*, the CAN bus is modelled as *"CanPhysicalChannel"*, which represents a CAN communication medium. As presented in the previous chapter, we intend to perform scheduling analysis after the integration of the cruise control software with the software of other sub-systems executing in the same ECUs. In the implementation phase of the cruise control, we should describe, hence, only the OS tasks that allocate the cruise control runnable in each ECU. Then, during the integration phase, the software platform models from different sub-systems are integrated and scheduling analysis can be performed. Figure 37 shows an overview of the software and hardware platform used by the cruise control.

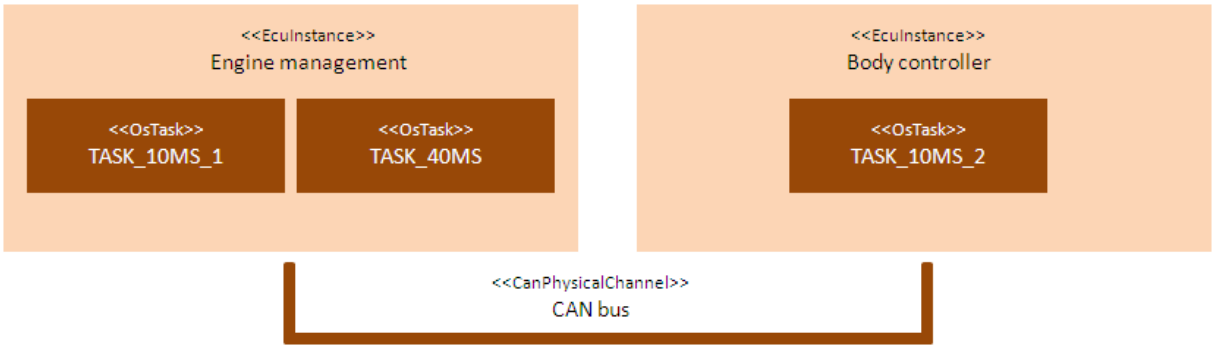

Figure 37 Resource platform used by the cruise control

The description of the OS tasks is produced during description of the OS configuration for each ECU. With this configuration, it is possible to define the OS tasks and their scheduling properties (e.g. priorities). AUTOSAR concepts are used here for OS configuration. After performing scheduling analysis, we can assess the improvements that are necessary for the chosen software resource platform and the mapping scheme selected.

- **Mapping View**: To perform scheduling analysis, we have to describe the mapping of OS tasks to processing units. In AUTOSAR, this can take place when the OS configuration is defined for each ECU. In this view, we also described mapping of the cruise control runnable entities to the selected OS tasks. To do so, we described the mapping of runnable triggering events to the OS tasks. Figure 38 gives an overview of the mapping view developed during this phase. To decide on a scheme for mapping runnable entities to OS Tasks, we should comply with the allocation scenario selected at the design level. This means that the runnable entities of the input acquisition/interpretation function should be mapped to a task allocated to the body controller ECU and the other runnables to tasks allocated to the engine management ECU. Based on the available

software platform, we decided to allocate the runnable entities of failure management to TASK_ 10MS_1 that is allocated to the engine management ECU. The runnable entities of control and speed setpoint calculation are allocated to TASK_40MS. In a similar way, the runnable entities of acquisition and interpretation are allocated to TASK_10MS_2 that is hosted by the body controller ECU. To describe such mapping, we used AUTOSAR concepts for RTE configuration. For example, mapping of the diagnosis entity to the TASK_10MS_1 is described using AUTOSAR concepts to define the link between the triggering event of the entity, and the OS task.

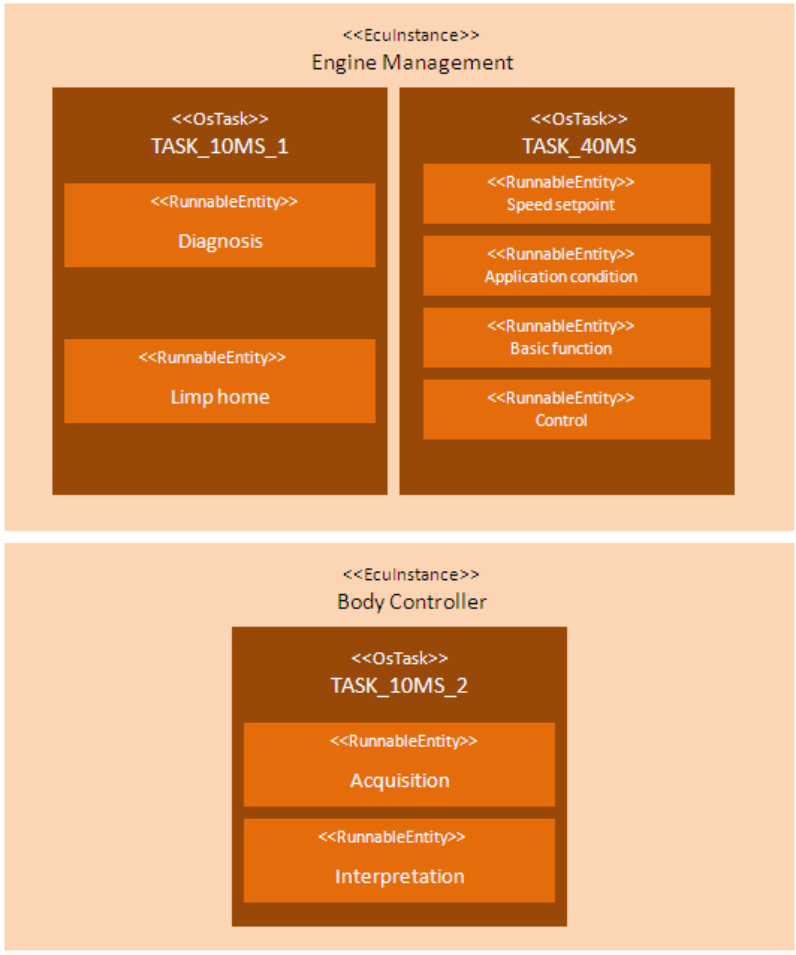

Figure 38 Cruise control mapping view

2.1.5. Scheduling Analysis

We performed scheduling analysis after integrating the cruise control sub-system with other sub-systems executing on the engine management and body controller ECUs (for simplification reasons, the development of other sub-systems is not presented here, but it should be done also by application of the methodology either to develop them from scratch

or by reuse). We used the SymTA/S tool to perform scheduling analysis. To do so, we transformed the AUTOSAR model of the integrated system into a SymTA/S model (cf. section 2.2.3 of part I). We used SymTA/S concepts to describe the OS tasks and the hardware resources (ECUs and CAN bus) included in the resource platform view. Each event chain described in the timing behaviour view is transformed into a "path" formed by runnable entities in the SymTA/S model. The latency constraint for each event chain is specified as a path maximum response time in SymTA/S.

Scheduling analysis results:

Table 17 shows a part of the analysis results (only results related to the cruise control sub-system). It gives the response times obtained for each event chain described in the model. All the response times obtained for all the sub-systems are less than the specified time constraints (deadlines), which means that the system is schedulable.

Table17 Response Times for Cruise Control

Event Chain	Response times (ms)	Deadlines (ms)
Input acquisition and interpretation	15.2	30
Failure management	10	20
Speed setpoint calculation	40.1	90
Control	17.3	50

The tool calculated an overall load of 60% for the engine management ECU with 1.8% requested by the cruise control functions. The overall load of the Body Controller ECU is 75% with 0.8% requested by the cruise control functions. Based on these results, we can validate the architecture designed (application, mapping, software & hardware resource platform).

2.2. Development by Reuse: Knock

In this chapter, we present an example of the application of the methodology to develop the software of a knock sub-system by reusing and adapting a previous version of it. We deal

with the case of medium reuse and particularly we focus on the scenario of adding a new software module to the knock previous software version.

2.2.1. Use Case Presentation

The knock sub-system is used to detect engine knocking during engine combustion and adjust the ignition accordingly to prevent the engine from "knocking". In gasoline internal combustion engines with spark ignition, an undesired effect may occur when the fuel mixture partially and spontaneously ignites as a result of the compression in the combustion chamber. The knock sub-system is developed to avoid such phenomenon.

The knock control is based on the acquisition of the engine noise signal during a crankshaft angular window. This window is set around the ignition operation (we call it main window). Based on the acquired noise signal, a detection phase is performed to determine if knock exists or not. In case of knock detected, a correction is performed by calculating an angular retard to be applied to ignition instant to compensate knock phenomenon.

In turbo-compressed engines, an undesired pre-ignition phenomenon which is similar to the knock phenomenon can occur before the ignition (during another timing window that we call pre-window). This phenomenon may be very harmful and increases considerably the emission of pollutant gases. Thus, it is necessary to control it.

In this chapter, we show how to apply our methodology to develop a new version of the knock control sub-system that allows controlling both knock and pre-ignition phenomena. This is done based on previous version of this sub-system that allows only detecting and controlling the knock phenomenon. As shown in the previous chapter, to develop the new version of the knock sub-system, we use the information represented in the XD model (cf. table 15) of the previous software version and extend it to obtain the needed new software version. Figure 39 shows an overview of the previous software architecture as organized in the XD tool. All the knock software modules are executed on the engine management ECU. The noise acquisition software module contains only one operation that allows the filtering and the integration of the engine noise raw signal. The acquisition of the engine noise signal is performed during a window whose begin instant and duration are calculated by the window parameter calculation operation of the detection software module. The threshold calculation operation, allows, based on the filtered engine noise signal, calculating a threshold value. Based on this threshold value, the knock energy calculation operation determines the knock energy which reflects the knock intensity. If this energy exceeds a

specific limit, this means that a knock phenomenon is taking place. In case of knock detected, the correction loop operation of the control software module calculates an angle retard to be applied to the ignition instant to compensate the knock. The ignition angle retard is communicated to the ignition setpoint sub-system that controls the ignition instant.

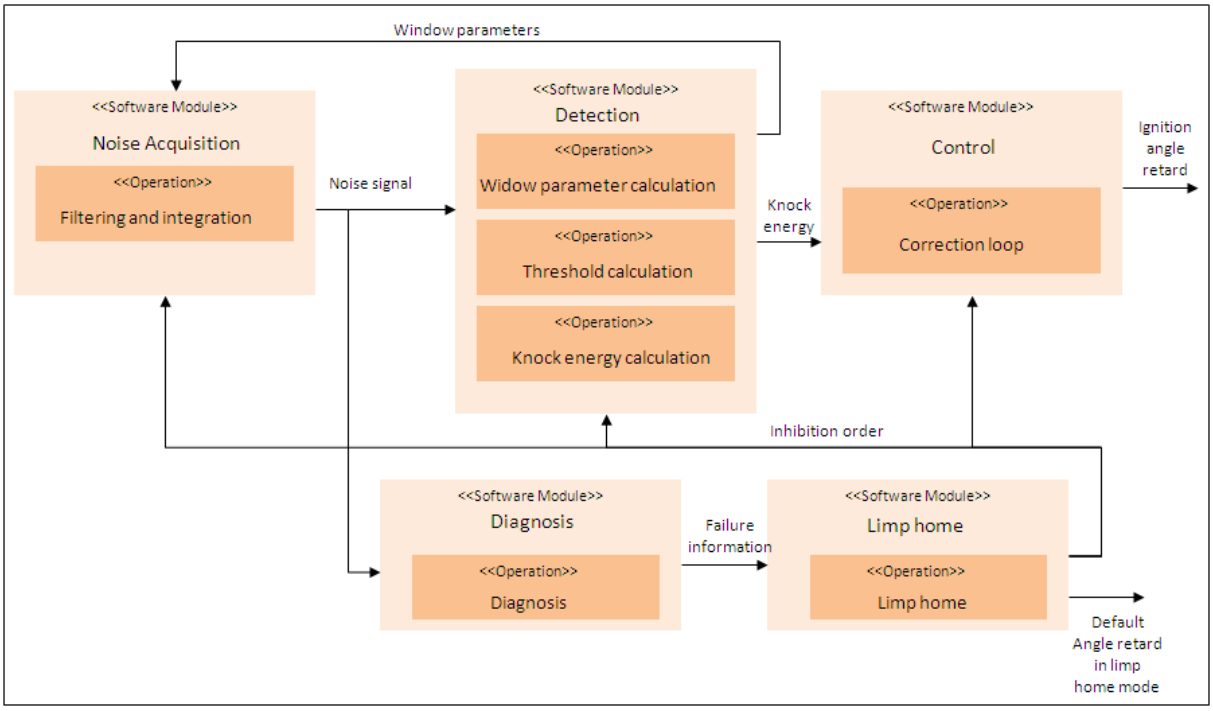

Figure 39 Simplified overview of previous knock software architecture

Based on this architecture, we propose to develop a new version of this sub-system that allows controlling also the pre-ignition phenomenon. We call this new version knock/pre-ignition sub-system rather than knock sub-system.

The control of the pre-ignition phenomenon by the knock/pre-ignition sub-system consists of detecting the pre-ignition phenomenon during the pre-window and then sending a request to the ignition realization sub-system to stop the ignition in the corresponding cylinder if pre-ignition is detected.

2.2.2. Analysis phase

In this phase, we determined the EMS end-to-end requirements that the new version of the knock/pre-ignition sub-system should meet. We identified two EMS end-to-end requirements:

- **EMS_REQ1**: In case of knock detected, the knock should be compensated within 700ms
- **EMS_REQ2**: In case of pre-ignition detected, ignition stopping order should be delivered within 900ms

Based on these end-to-end requirements, we should determine the time budgets to be allocated to the knock/pre-ignition sub-system. Before this, we should develop the analyzable model.

2.2.2.1. Analyzable Model

- **Knock/pre-ignition Analysis Functional View:** Figure 40 shows the functional view developed during this phase for the knock/pre-ignition sub-system. The figure shows the interaction of the knock/pre-ignition sub-system with other sub-systems within the EMS. As presented during the methodology description, EAST-ADL constructs for functional modeling are used to develop this view.

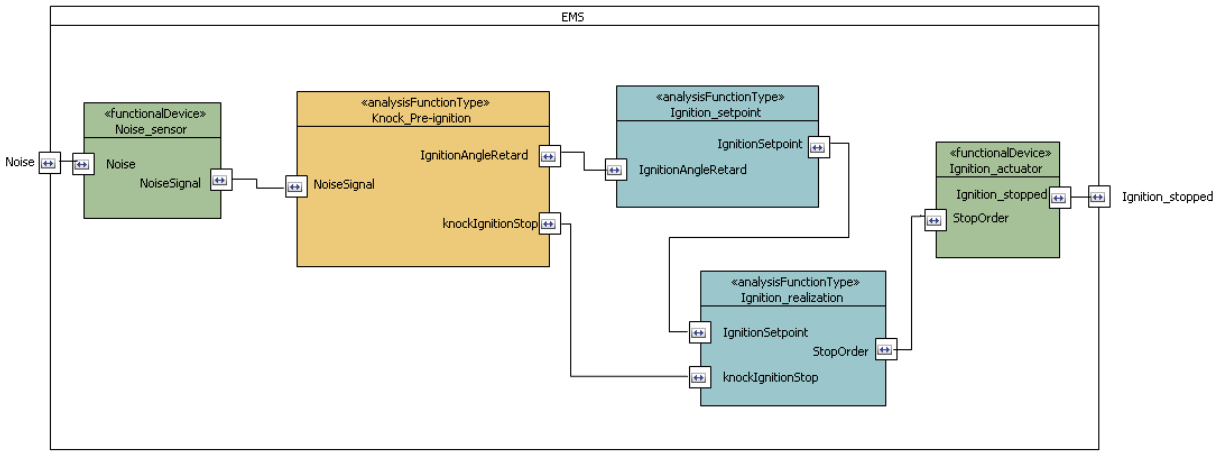

Figure 40 Knock/Pre-ignition analysis functional view

- **Knock/pre-ignition Analysis Timing View**: Figures 41 and 42 show the timing view developed to represent the above-mentioned end-to-end requirements. EMS_REQ1 means that since the acquisition of the noise signal by the sensor and the detection of the knock phenomenon by the knock/pre-ignition sub-system until the new ignition setpoint is calculated by the ignition setpoint sub-system (based on the ignition angle retard information), the duration should not exceed 700ms. EMS_REQ2 means that since the acquisition of the noise signal and the detection of the pre-ignition phenomenon by the knock/pre-ignition sub-system until the ignition realization delivers an order to stop the ignition operation, the duration should not exceed 900ms. The two following figures

represent respectively the timing views corresponding to the requirements EMS_REQ1 and EMS_REQ2.

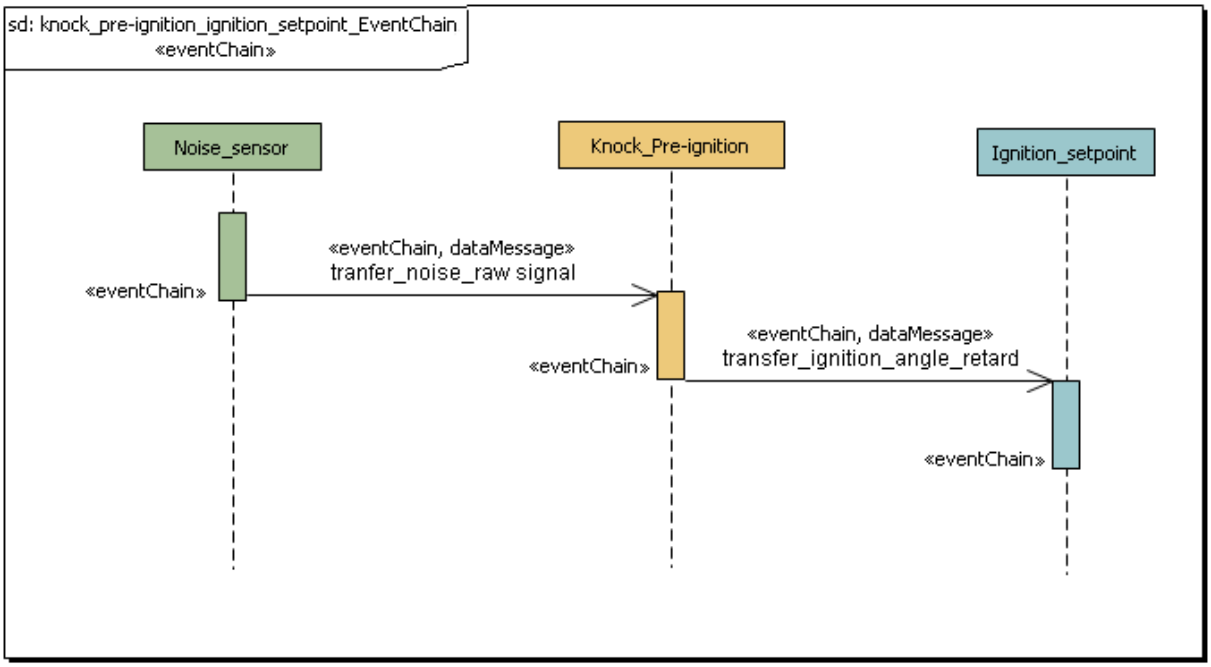

Figure 41 Knock/Pre-ignition-to-ignition setpoint event chain

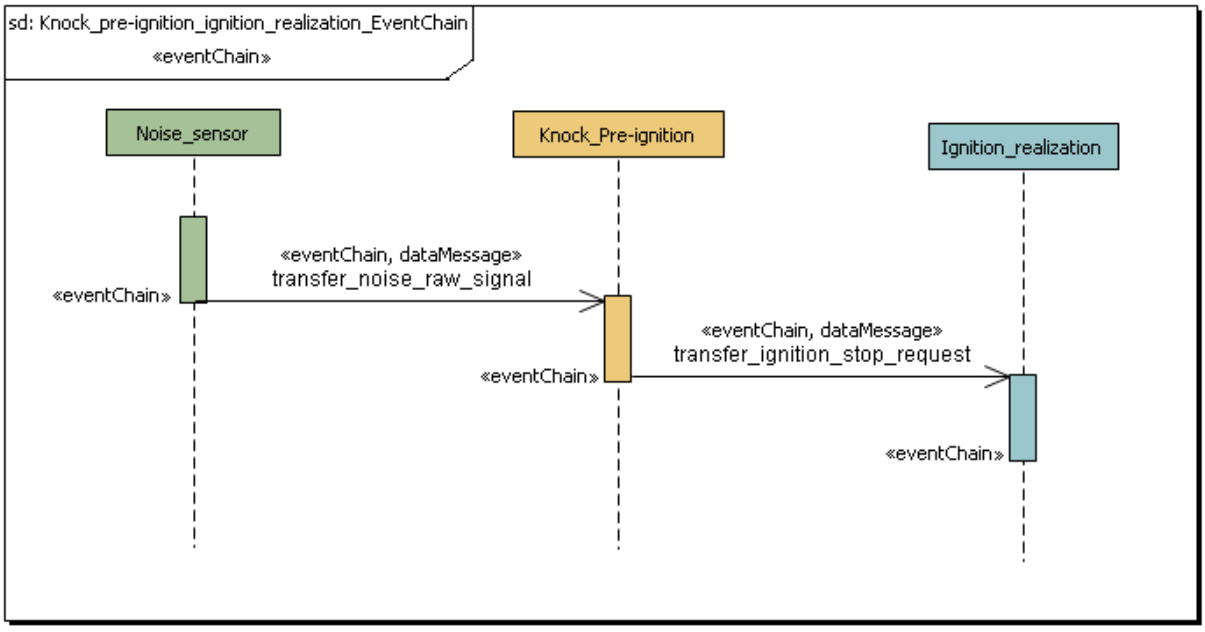

Figure 42 Knock/Pre-ignition-to-ignition realization event chain

2.2.2.2. Determination of Knock/Pre-ignition Time Budgets

Based on our knowledge of the time budgets of previous versions of the ignition setpoint and ignition realization sub-systems, we determined the following time budgets to be allocated

to the knock/pre-ignition sub-system; to comply with the EMS-REQ1 the knock detection and correction should be performed within 300ms. To comply with EMS_REQ2, from the detection of the pre-ignition until an ignition stop request is delivered the duration should not exceed 250ms. Hence, we have the following two constraints to be satisfied when refining the knock/pre-ignition functional architecture during the design phase:

- **AConst1**: The detection and the correction of the knock should be performed within 400ms
- **AConst2**: From the detection of the pre-ignition until an ignition stop request is delivered, the duration should not exceed 250ms.

2.2.3. Design phase

It is during this phase, that we start reusing the information available from the previous version of the knock sub-system software.

To detect the pre-ignition phenomenon, it is possible to use exactly the same software modules as the previous version without adding new functional blocks/software modules. In this case; we just need to adapt the software module called "detection" to enable both the detection of the knock and pre-ignition phenomena. However, the pre-ignition phenomenon occurs only in turbo-compressed engines. So, for the other engine kinds, only the previous knock software version is needed. To facilitate the reuse of both the previous and new versions of the sub-system, we decided, hence, to add a new function that allows detecting separately the pre-ignition phenomenon during the pre-window.

2.2.3.1. Analyzable Model

- **Knock/pre-ignition Design Functional View**: We develop this view based on the already existing software modules. As described in the previous chapter, we transform each software module from the previous software version to a functional block. Then, we add a new functional block for the detection of the pre-ignition phenomenon. Figure 43 shows an overview of the knock/pre-ignition functional architecture developed during this phase.

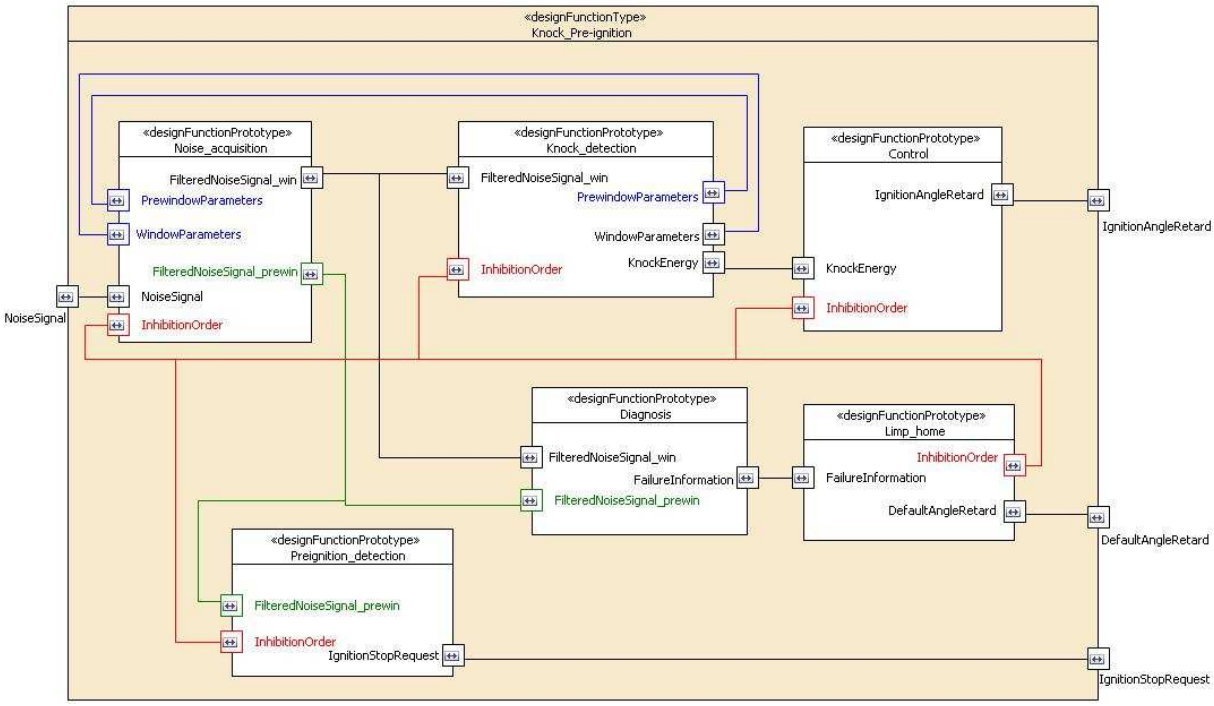

Figure 43 Knock/pre-ignition design functional view

As figure 43 shows, the software modules noise acquisition, detection, control, diagnosis and limp home are transformed respectively into functional blocks called noise acquisition, knock detection, control, diagnosis and limp home. We introduced the new functional block pre-ignition detection for the detection of the pre-ignition phenomenon. This detection is done based on the noise signal acquired during the pre-window. As presented in the methodology description, EAST-ADL constructs for functional modeling are used to develop this view. Note that during this phase, we perform just a direct transformation of each software module from the previous software version to a functional block without focusing on the software module internal implementation (i.e., the operations/runnables that it contains). Such implementation will be described during the implementation phase based on the new functional architecture chosen and the timing analysis results obtained during the design phase. Nevertheless, during this phase (design), we decide about the role of each functional block of the new configuration. For example, the noise acquisition functional block will perform not only the acquisition of the noise signal during the main window (as it was the case for the previous software version) but also the acquisition of the noise signal during the pre-window. As for the previous software version, the knock detection functional block is responsible for the detection of the knock phenomenon and the parameter calculation for the main window.

For the new version of the knock/pre-ignition sub-system, this functional block will also

perform the parameter calculation of the pre-window. The control functional block will perform the same task as for the previous software version. The diagnosis and limp home functional blocks will process the noise signals acquired during both the main window and the pre-window.

Based on this functional configuration and the time budgets determined for the knock/pre-ignition sub-system, we determine the time budget to allocate to each functional block. To do so, we need to develop first the timing view of this phase.

- **Knock/pre-ignition Design Timing View**: Figure 44, 45 and 46 show the timing view of the design phase. The constraint AConst1 (figure 44) means that from the start of the noise acquisition functional block until the knock detection and then the control are finished, the duration should not exceed 300ms. AConst2 (figure 45) means that since the start of the noise acquisition until the pre-ignition detection is finished, the duration should not exceed 250ms. From the previous version of the knock sub-system, we introduce, during this phase, a new constraint AConst3 (figure 46) that requires that, from the start of the diagnosis until the end of the limp home, the duration should not exceed 200ms.

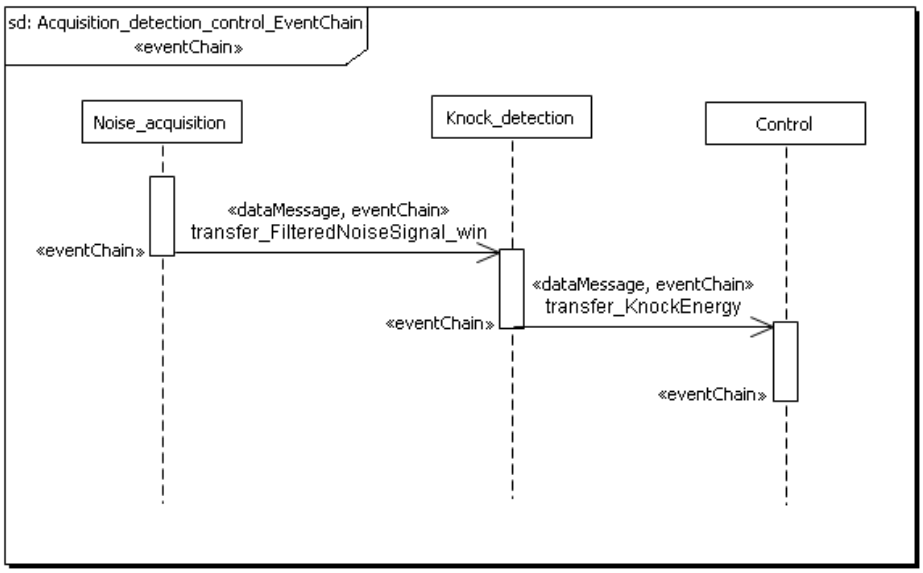

Figure 44 From acquisition to control EventChain

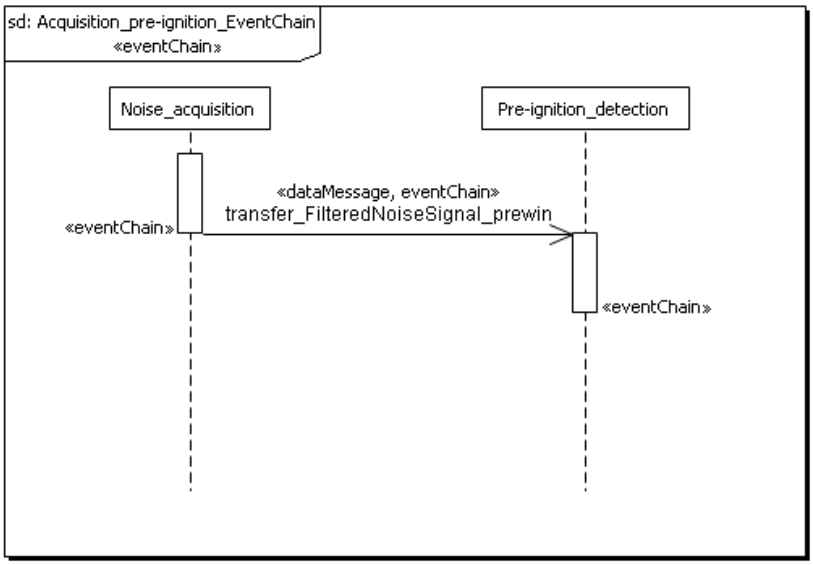

Figure 45 From acquisition to pre-ignition detection EventChain

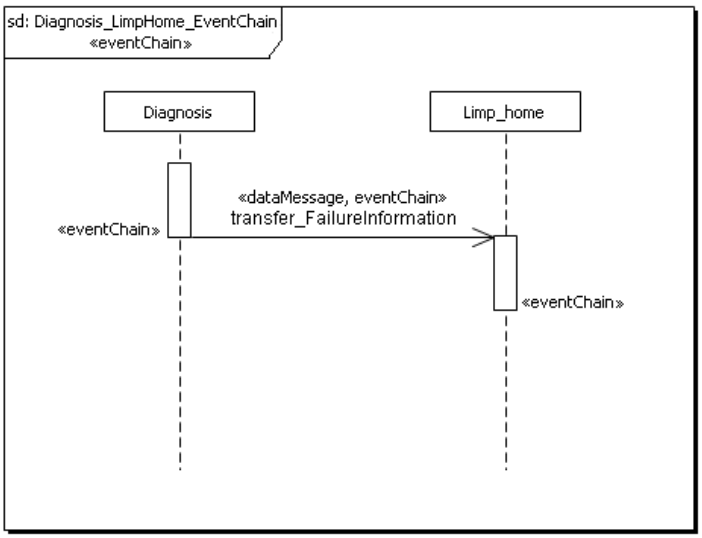

Figure 46 From diagnosis to limp home EventChain

Based on the above mentioned constraints and with the help of knock sub-system specialists we determined the following time budgets to allocate to the functional blocks of the new knock/pre-ignition version (table 18).

Table 18 Knock/pre-ignition functional block time budgets

Functional block	Time budget (ms)
Noise acquisition	100
Knock detection	200
Control	100
Pre-ignition detection	150
Diagnosis	100
Limp home	100

These time budgets mean that:

- **DConst1**: The noise acquisition should be performed within 100ms
- **DConst2**: The knock detection should take place within 200ms
- **DConst3**: The knock control should be performed during 100ms
- **DConst4**: The pre-ignition detection should take place within 150ms
- **DConst5**: The diagnosis should be performed within 100ms
- **Dconst6**: The limp home should take place within 100ms

These constraints should be respected when creating the software architecture of the knock/pre-ignition during the implementation phase

2.2.3.2. Hardware Architecture Exploration:

As mentioned previously, for the previous version of the knock sub-system, all the software modules execute on the engine management ECU. In our case, the new version of the sub-system should use the same resource. Hence in this phase, we perform ECU load estimation not to determine the best functional block-to-ECU allocation scenario but just to verify that the CPU load budget decided for the new version is respected with the new configuration (introduction of pre-ignition detection). With the help of EMS designer and knock sub-

system specialists, we identified that the load requested by the knock/pre-ignition should not exceed 5% of the global load of the engine management ECU. To verify if this load could be respected or not, we should determine the ECU load requested by the new software version based on the functional architecture described previously and the execution times estimated for the functional blocks.

Estimation of functional block execution times:

The execution time of each functional block is determined based on the previous software version. In fact, for this previous version, we have a data base describing the execution time of each operation (these operations are described in figure 39). Table 19 shows the execution times of the previous software version operations. These execution times will allow us having an estimation of the execution time of the functional blocks.

Table 19 Operation execution times from previous knock software version

Operation	**Execution Time (µs)**
Filtering and integration	7
Diagnosis	7.5
Limp home	400
Window parameter setting	6
Threshold calculation	9
Knock energy calculation	9
Correction loop	27

In the previous software version, the operation filtering and integration has a worst case execution time of 7 µs. For the new version of the sub-system, we know that this operation should be performed twice (first on the signal acquired during main window and then on the signal acquired during pre-window). Thus, for the functional block noise acquisition, we can estimate a bound of 14 µs for its execution time (the double of the filtering and integration operation execution time). As the control functional block role remains unchanged, we can assign to it the same execution time as the previous version.

For the knock detection functional block, in addition to old treatment (main window parameter calculation, threshold calculation, energy calculation), the new version should calculate also the parameters of the pre-window. Thus, we can estimate for it an execution time of 30 µs (the sum of the previous execution times augmented by an extra time for the calculation of pre-window parameter. This augmentation should be nearly equal to the execution time of the main window parameter calculation operation). The bound to be estimated for the execution time of the diagnosis functional block is 15µs as the treatment of this functional block should be performed for both the noise signal acquired during the main window and the signal acquired during the pre-window. For the pre-ignition functional block, we estimate an execution time of 20µs (the detection of the pre-ignition phenomenon will be nearly similar to the knock detection based on the calculation of a pre-ignition threshold)

To be able to estimate the load requested by knock/pre-ignition functional block, we also need to know the period of each treatment. Table 20 describes the activation of each operation from the previous software version.

Table 20 periods of previous software version operations

Operation	**Activation**
Filtering and integration	Window end event
Diagnosis	SEG_event
Limp home	100ms
Window parameter setting	SEG_event
Threshold calculation	SEG_event
Knock energy calculation	SEG_event
Correction loop	SEG_event

The window end event and the SEG event are engine-synchronous events (the periods of these events depend on the engine speed). In our case, to simplify the understanding of the illustration, we choose to perform timing analysis for a six cylinder engine running at 6000 rpm. At this engine speed, the window end event has a period of 2ms. The SEG event has a period of 3ms.

Based on these values, we can assign the following periods to our functional blocks (table 21). We choose to assign a period of 3ms for the pre-ignition detection functional block (similar to the period of knock detection functional block). Table 21 shows also the functional block execution times.

Table 21 Knock/pre-ignition functional block execution times and periods

Functional block	**Execution time (µs)**	**Period (ms)**
Noise acquisition	14	2
Knock detection	30	3
Control	27	3
Diagnosis	15	3
Limp home	400	100
Pre-ignition detection	30	3

Based on these values, the global load requested by the knock/pre-ignition functional blocks is 4.1%. This value is less than the load budget authorized for the knock/pre-ignition sub-system. This means that the architecture conceived can be validated and it is possible to move to the next development phase, the implementation.

2.2.4. Implementation Phase

During this phase, we transform the knock/pre-ignition functional architecture described during the design phase into a software architecture using AUTOSAR concepts. As for the cruise control use case, we used the Cessart-CT tool to create the models of this phase.

- **Knock/pre-ignition Application View**: In this view, we chose to transform each functional block from the design phase to an AUTOSAR software component. Figure 47 shows a simplified overview of the software components and their communication (for the clarity of the figures, we do not show all the data exchanged between the software components). Figure 48 shows the runnable entities created for each software component.

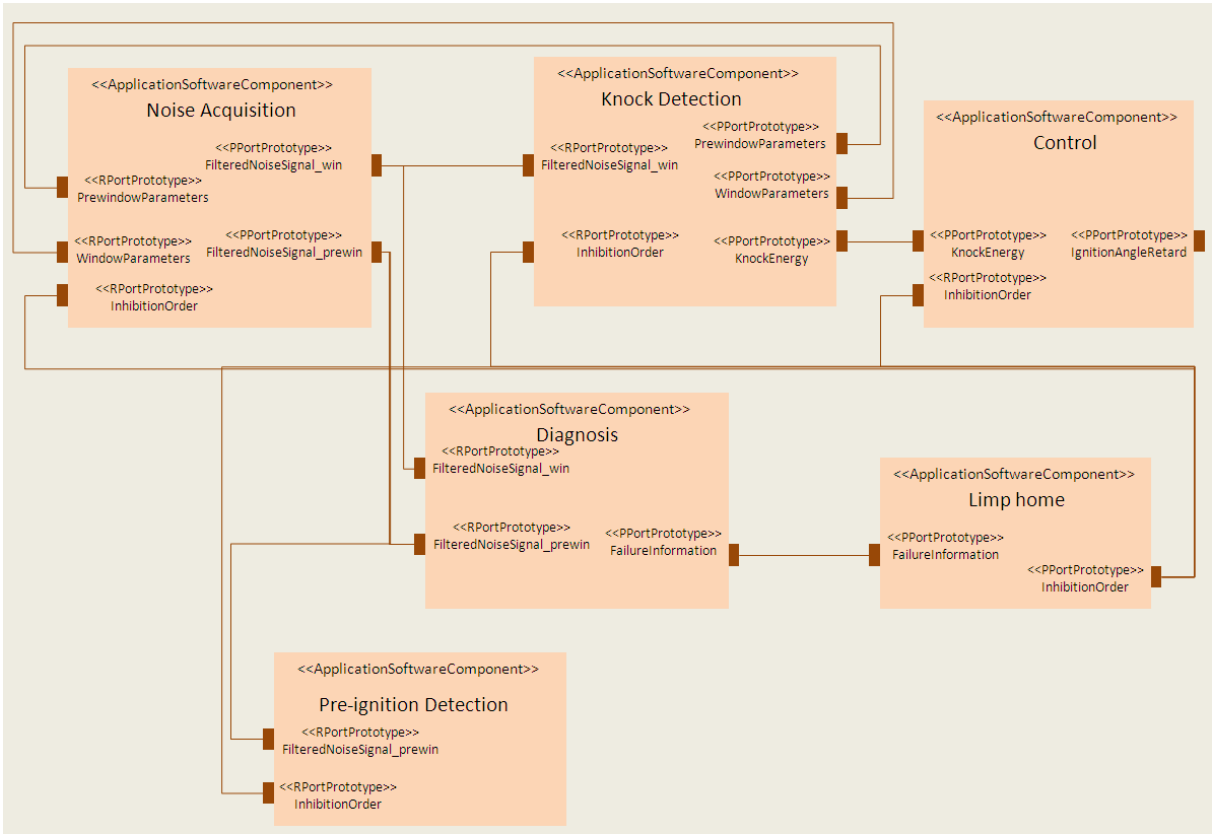

Figure 47 simplified overview of Knock/pre-ignition AUTOSAR software architecture

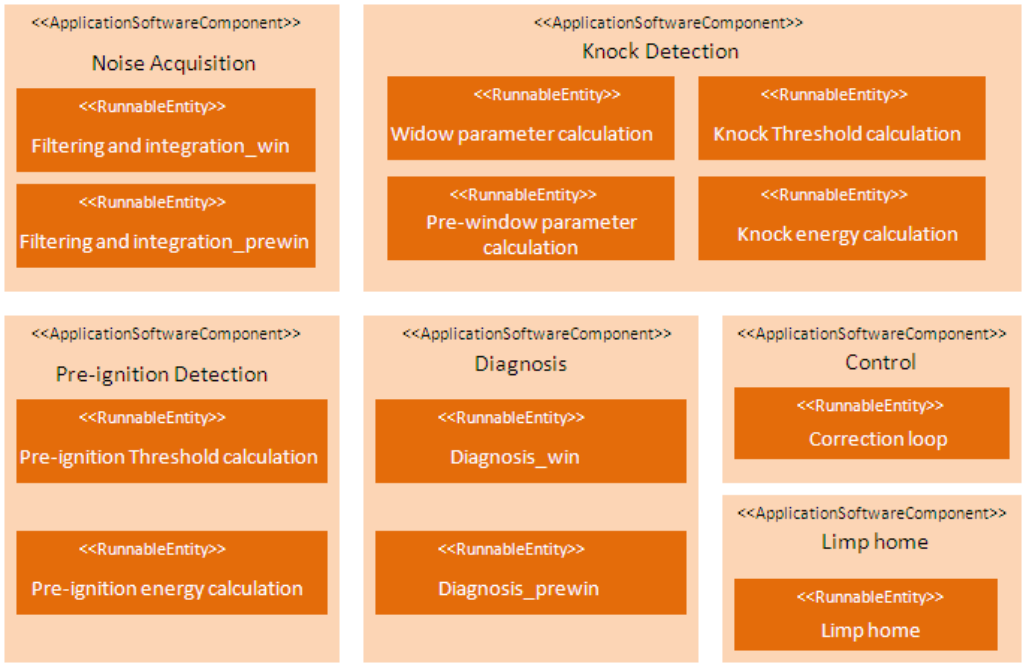

Figure 48 Knock/pre-ignition software components and runnable entities

As figure 48 shows, compared with the previous software version, the internal behavior (the defined runnable entities) of each software component is adapted to respect the new

configuration. The software component noise acquisition contains two runnable entities. The first one performs the filtering and integration of the noise signal acquired during the main window and the second one processes the signal acquired during the pre-window. For the software component knock detection, we added a runnable entity for the calculation of the pre-window parameters (begin instant and duration). The diagnosis software component contains two runnable entities that perform the diagnosis of the noise signal acquired during the main window and the one acquired during the pre-window. The pre-ignition software component contains two runnable entities. The first runnable entity calculates a pre-ignition threshold. Based on this threshold, the second runnable entity calculates the pre-ignition energy. If this energy exceeds a fixed limit, this means that pre-ignition is occurring.

- **Knock/pre-ignition Timing Behavior View**: In this view, we modeled the timing constraints determined during the design phase by means of AUTOSAR events and event chains. For example, for the constraint DConst3, we modeled an event chain having as stimulus the activation event of the runnable entity correction loop and as response the termination event of the same runnable. For this event chain, we specified a latency constraint having 100ms as a maximum value. For Dconst4, we modeled an event chain having as stimulus the activation event of the runnable pre-ignition threshold calculation and as response the termination event of the runnable pre-ignition energy calculation. For this event chain, we specified a latency constraint of 150ms. Figure 49 shows a representation of the event chain related to the constraint Dconst4 (pre-ignition event chain).

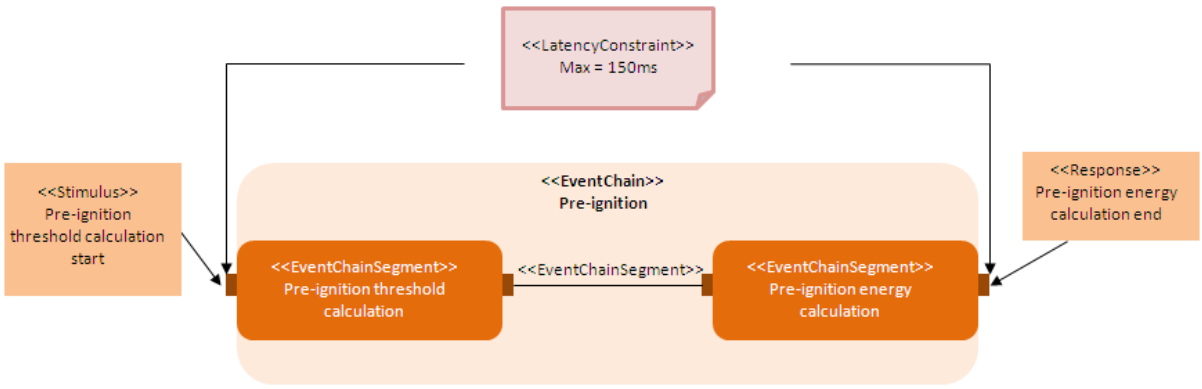

Figure 49 Representation of the modeling of the pre-ignition event chain

- **Resource Platform View**: In this view, we model the hardware and software resources used by the knock/pre-ignition sub-system. Figure 50 shows an overview of the resource platform used by this sub-system. As mentioned previously, all the knock/pre-ignition software components will run on the engine management ECU. Task_ENG is an engine-synchronous task that is triggered by the SEG event presented in table 20. Task_WinEnd is triggered by the window end event.

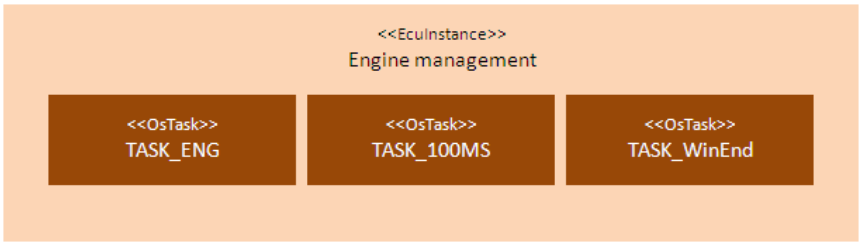

Figure 50 Representation of the resource platform view

- **Mapping View**: Figure 51 shows a representation of the mapping view, it presents the distribution of the runnable entities between OS tasks. To develop this view, we used the information of the mapping of the previous software version operations to OS tasks. For the new software architecture, we chose to map the runnable entities of the pre-ignition detection software component to the TASK_ENG.

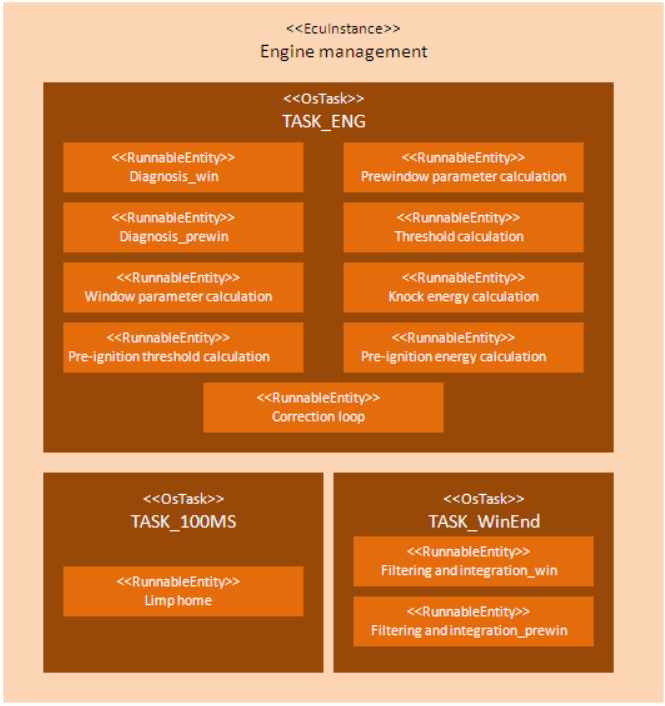

Figure 51 Representation of the mapping view

2.2.5. Scheduling Analysis

We performed scheduling analysis using the SymTA/S tool. The scheduling analysis was performed taking into account all other EMS sub-systems and the new knock/ pre-ignition constraints. Table 22 shows the response times related to the knock/pre-ignition sub-system. All the response times are less than the deadlines specified (the same for the constraints considered for the other sub-systems) which means that the system is schedulable.

Table22: Response times for the knock/pre-ignition event chains

Event Chain	**Deadline (ms)**	**Response Time (ms)**
Noise acquisition	100	25
Knock control	100	15
Pre-ignition detection	150	82.5
Knock detection	200	110
diagnosis	100	30.5
Limp home	100	40.2

3. About the Methodology Acceptability

In this chapter, we propose to measure the acceptability of our methodology and its potential to be adopted by Continental engineers. Let's note that the results of this thesis work has contributed among many other research works carried out at Continental to the decision of Continental to migrate to a new EMS software architecture based on AUTOSAR concepts and adapted to EMS characteristics. As a part of this migration process, a tool allowing the description of EMS software architecture with AUTOSAR concepts is being developed. This tool gives the possibility to transform previous software version described with XD models into AUTOSAR software architecture. It is also intended to allow the integration of AUTOSAR models from different parts of the EMS software.

3.1. Tasks, Roles, Skills

To measure the acceptability of our methodology, we propose to measure the gap between our proposed process and the current EMS development process (described in section 1.2 of this part) in terms of current vs. new tasks and skills in order to determine the training needs and to evaluate their availability potential. Table 23 gives a comparison between the tasks performed currently by the different roles involved in the EMS development process and the new tasks required by our methodology. The new tasks required by the methodology should be weaved into the tasks performed currently during the EMS development process.

Table 23 Current vs. new tasks

Role	Current task	New task
EMS designer	- EMS requirement analysis (functional and performance requirements): elicitationand integration in Doors data base - EMS partitioning	- EMS end-to-end timing requirement determination and analysis - Analysis functional views modeling - Analysis timing views modeling - Sub-system time and CPU load budgets determination
Function developer	- Sub-system requirement analysis (requirement elicitation and integration in Doors data base) - Function and algorithms description (manually or through Simulink modeling)	- Sub-system design functional view modeling - Sub-system timing view modeling - Functional block time budgets determination - Functional block execution times estimation - Modeling of abstract architecture of the hardware platform to be used by the sub-system. - Allocation of functional blocks to hardware resources modeling - Processor load estimation
Software developer	- Sub-system algorithms implementation (manual C coding or automatic C code generation from Simulink models)	- Sub-system software architecture description using AUTOSAR - Runnable entity timing information determination (timing constraints & execution times)
Software integrator	- C code and algorithms integration from different sub-systems	- AUTOSAR models from different sub-systems integration - Scheduling analysis performance for integrated EMS

As the table shows, the new tasks required by the methodology are centered on modeling and timing analysis. To be capable to perform these tasks, some skills need to be acquired by the different roles involved in the development process. Table 24, 25, 26 and 27 give a comparison between the current skills and the new required skills for each role involved in the development process.

Table 24. Current vs. new skills and training needs for EMS designer

Role	Current skills	New skills			Training needs
		Required skill	Rational	Availability risk	
EMS designer	- Use of Doors tool - Domain knowledge: engine control knowledge - CPU load budgeting	Real time skills: determination of EMS end-to-end requirements and sub-system time budgeting	EMS designer should have these skills to be able to determine the EMS end-to-end requirements and, based on this, to determine the time budget to assign to each sub-system	No availability risk as EMS designer can be supported by the function and software developer of each sub-system who have better knowledge about the time budget that can be acceptable for the considered sub-system	No training is needed Need for support from sub-system function and software developer
		EAST-ADL functional modeling	To be capable to develop functional views during the analysis phase	Medium availability risk as EAST-ADL is not very well known currently and hence there is a risk of lack of EAST-ADL training	EAST-ADL training
		UML diagrams use	Need for use of UML composite structure diagrams for functional views and UML sequence diagrams for timing views	No risk of availability as UML is a standard and training for UML concepts and tools can be provided easily	UML basic training
		UML editor use	Needed to develop the models during the analysis phase		UML editor training (e.g., ARTISAN studio, papyrus MDT)
		TADL modeling	Need to be familiar with TADL concepts to develop the timing views that represent the EMS end-to-end requirements during the analysis phase	No availability risk as TADL notions are integrated in AUTOSAR timing extensions for which training can be provided at Continental	TADL training
		Eclipse use	This skill is needed as most of available UML editors are eclipse based	No availability risk as EMS designer can be supported by software developers who have already eclipse use skills	Eclipse basic training

As the table shows, most of the new required skills for EMS developer are centered on modeling skills and the use of modeling languages and tools. Such skills can be acquired by EMS designer through training.

Table 25. Current vs. new skills and training needs for function developer

Role	Current skills	New skills			Training needs
		Required skill	Rational	Availability risk	
Function developer	- Use of Doors tool - sub-system functional design description as Word specifications - Auto coding-aware Simulink modeling	Real time skills: capability to determine time budgets and estimate execution times	To be capable to determine functional block time budgets and estimate functional block execution times	No availability risk as function developer can be supported by Continental dynamic architecture specialists who have strong real time skills	Support from EMS dynamic behavior experts Information exchange with software developer
		EAST-ADL functional modeling	To be capable to develop functional views during the design phase	Medium availability risk as EAST-ADL is not very well known currently and hence there is a risk of lack of EAST-ADL training	EAST-ADL training
		UML diagrams use	Need for use of UML composite structure diagrams for functional views and UML sequence diagrams for timing views	No risk of availability as UML is a standard and training for UML concepts and tools can be provided easily	UML basic training
		UML editor use	Needed to develop models during design phase		UML editor training (e.g., ARTISAN studio, papyrus MDT)
		TADL modeling	Need to be familiar with TADL concepts to model the timing views during the design phase	No availability risk as TADL concepts are integrated in AUTOSAR timing extensions for which training can be provided at Continental	TADL training
		Eclipse use	This skill is needed as most of available UML editors are eclipse based	No availability risk as function developer can be supported by software developers at Continental who have already eclipse use skills	Eclipse basic training

Table 26. Current vs. new skills and training needs for software developer

Role	Current skills	New skills			Training needs
		Required skill	Rational	Skill availability risk	
Software developer	- MISRA C coding - Real time skills: capability to decide about recurrences and deadlines to assign to executing operations - Work within eclipse environment	AUTOSAR modeling: application modeling, timing modeling, platform modeling, mapping modeling	Software developer should master using AUTOSAR concepts to develop sub-system software architecture at implementation level	No availability risk as AUTOSAR trainings can be provided at Continental	AUTOSAR training
		AUTOSAR editor use	Needed to develop AUTOSAR models	No availability risk as an AUTOSAR editor adapted to describe EMS architecture is being developed at continental with the intention to train software developers to it.	AUTOSAR editor training

Table 27. Current vs. new skills and training needs for software integrator

Role	Current skills	New skills			Training needs
		Required skill	Rational	Availability risk	
Software integrator	-Software integration tool use (internal tools) - Capability to analyze the static and timing behavior (execution time and response time measurement) of the integrated software	AUTOSAR modeling	This skill is needed to enable a correct integration of the AUTOSAR models from different sub-systems	No availability risk as AUTOSAR trainings can be provided at Continental	AUTOSAR training
		AUTOSAR editor use	Needed to integrate AUTOSAR models	No availability risk as an AUTOSAR editor developed within Continental and adapted to EMS architecture is being developed with the intention to train software integrators to it.	AUTOSAR editor training
		Capability to work within Eclipse environment	The available AUTOSAR editors are eclipse-based.	No availability risk	Eclipse basic training
		Scheduling analysis tool use	Software integrator need to be capable to use scheduling analysis tools to perform scheduling analysis for the integrated system	No availability risk as for commercial tools such as SymTA/S, the provider is ready to train continental engineers to it.	Scheduling analysis tool training (e.g., SymTA/S, Chronval)
		Basic notions of scheduling theory	The software integrator should have some notions of scheduling theory (preemption, cooperation, blocking, offsets, etc) to be capable to interpret scheduling analysis results.	Real time properties of Continental tasks (recurrences, deadlines, preemptivity) are already known by software integrators. For scheduling theory notions, there is no availability risk as these notions can be acquired during trainings to scheduling analysis tools.	No specific training is needed, this knowledge can be acquired as part of training to scheduling analysis tools

Based on the comparison presented in the previous tables in terms of needed tasks and skills, we can conclude that there is a good potential for our methodology to be adopted by

Continental engineers. In fact some of the skills required by the methodology are already available in continental. For the skills that are not available yet, there is no risk to acquire them as trainings can be provided at Continental. However, as EAST-ADL is a new formalism, EAST-ADL trainings can be not available in the short term. The early development phases based on EAST-ADL modeling can be adopted as an enhancement of the future Continental development process that is intended to be based on AUTOSAR software architecture.

3.2. Tool Support

In this section, we propose to measure the acceptability of the proposed methodology in terms of tool support. To this end, we compare the current tool chain used at Continental to develop engine management systems and the tool chain required by our methodology. Figure 52 shows an overview of the current EMS development tool chain.

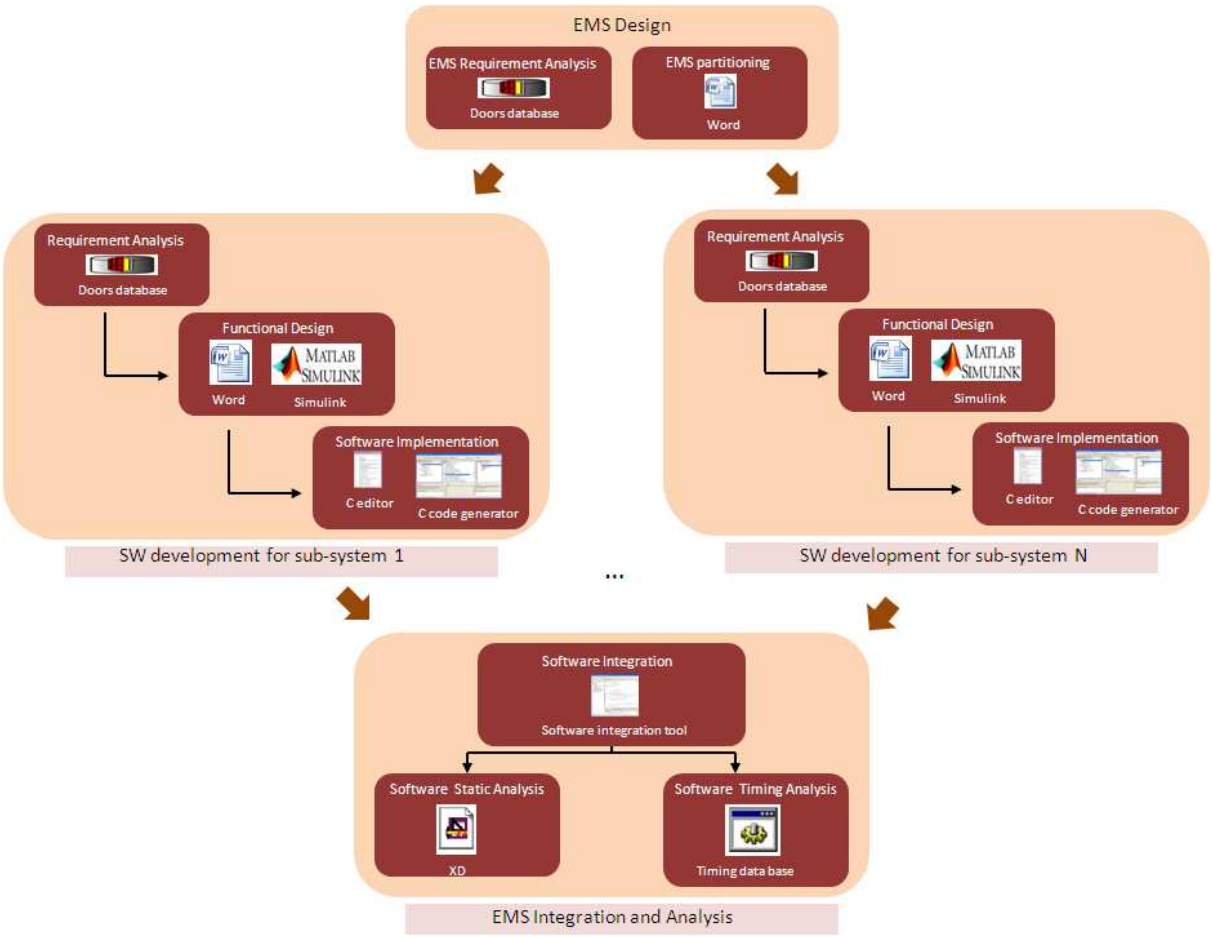

Figure 52 Current tool chain used to develop Engine management systems

As the figure shows, requirement analysis activities are performed by the EMS designer and function developer based on the Doors data base. During the software development of each sub-system, the functional design is described either as Word specifications in the code-centric approach or through using the Simulink tool to model the functions and their associated algorithms in the model-based approach. In the same way, the software implementation is either done manually using a C code editor or by generating the C code automatically using a C code generator. During the software integration, an internal software integration tool is used. Then, from this tool, an XD model is generated to enable the analysis of the static architecture of the software using the XD tool. In parallel, an internal tool (timing data base) allows performing the timing analysis of the integrated software by measuring the operation execution times, the OS task response times and the CPU load values.

Figure 53 shows the new EMS development tool chain underlying our methodology.

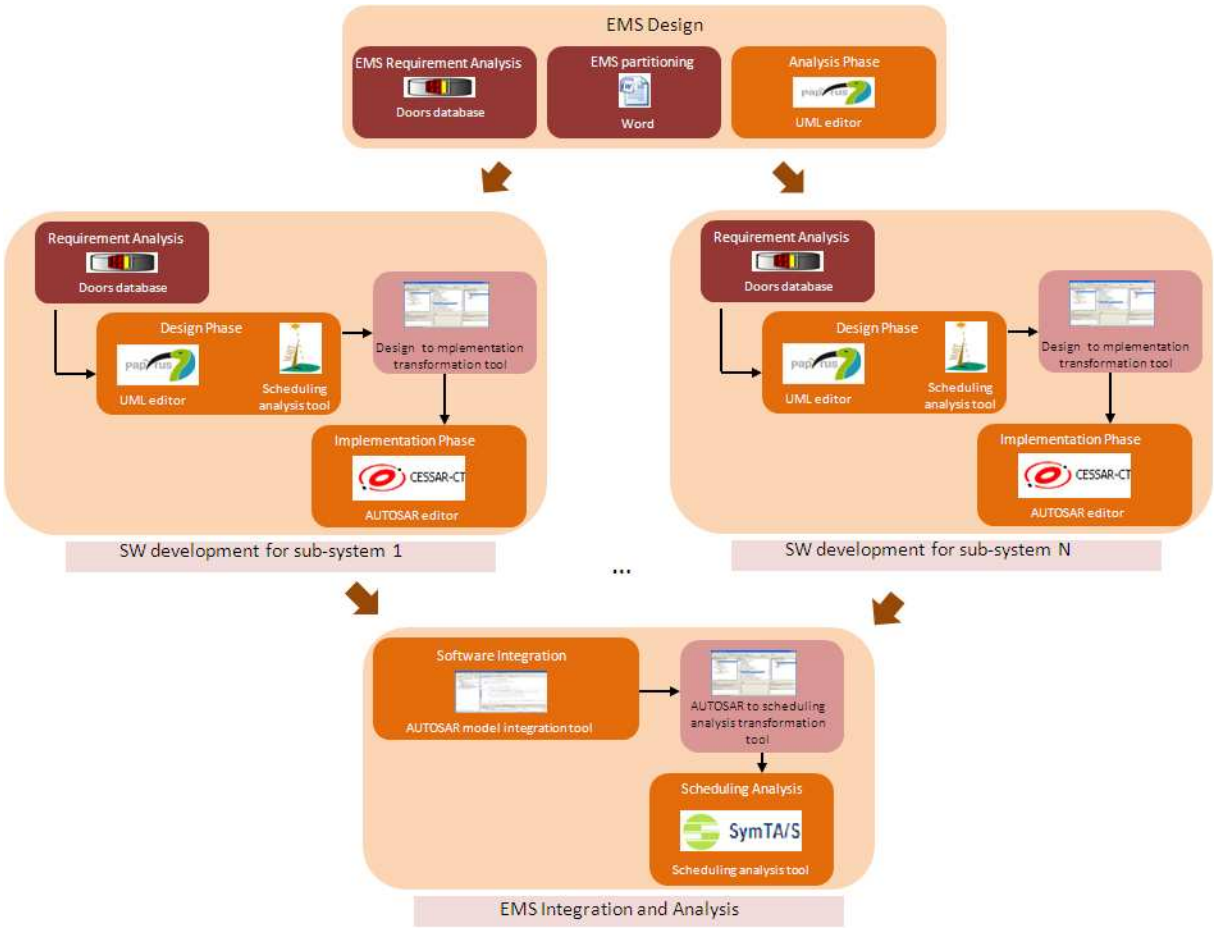

Figure 53 Methodology development tool chain

As figure 53 shows, during the analysis and design phases, a UML editor is needed in order to enable the development of functional and timing views during these phases. During the implementation phase of each sub-system, an AUTOSAR editor is needed to describe the software architecture using AUTOSAR constructs. Such editor is also needed to perform the integration of AUTOSAR models from different sub-systems. Based on the functional architecture modeled during the design phase for each sub-system, a large part of the AUTOSAR software architecture can be generated automatically using a transformation tool. In the same way, to perform scheduling analysis on the integrated architecture, a transformation tool is needed to transform AUTOSAR models to a model understandable by a scheduling analysis tool.

Table 28, 29 and 30 give a comparison between the current and the new tool chains during respectively EMS design phase, software development for each sub-system and software integration and analysis. The new tool chain required by the methodology should be weaved into the tool chain that is currently used during the EMS development process.

Table 28 Current vs., new tool chain used during EMS design phase

EMS Development phase	**Current tools**	**New tools**			
		Tool	**Rational**	**Example**	**Availability risk**
EMS design	- Doors data base - Word	UML editor	To develop EAST-ADL and TADL models during analysis phase	- Papyrus MDT - Artisan Studio	No availability risk as these tools are already developed. Artisan studio is already used by Continental engineers to develop basic software parts for each sub-system

Table 29 Current vs., new tool chain used during software development of each sub-system

EMS Development phase	Current tools	New tools			
		Tool	**Rational**	**Example**	**Availability risk**
Software development for each sub-system	- Doors - Word - Simulink - C editor - C code generator	UML editor	To develop EAST-ADL and TADL models during design phase	- Papyrus - Artisan Studio	No availability risk as these tools are already developed
		Scheduling analysis tool	To calculate the processor loads based on functional block-to-ECU allocation scenario	- MAST - Cheddar - SymTA/S	No availability risk as these tools are already available
		Design to implementation transformation tool	This tool is needed to generate automatically a part of the AUTOSAR software architecture from the functional EAST-ADL architecture and hence accelerate the development	Optimum [60]	A tool called Optimum that allows generating automatically AUTOSAR software components from EAST-ADL functional models exists already. However, this tool is not mature enough. The non-availability of such tool is not a blocking point for the adoption of the methodology but it may prevent from saving more development time compared with current status of the methodology
		AUTOSAR editor	This tool is needed to model the software architecture of each sub-system using AUTOSAR concepts	Cessar-CT	Cessar-CT is not mature enough. Nevertheless, there is no availability risk for an AUTOSAR editor as a tool is already being developed in Continental for a new EMS architecture based on AUTOSAR.

Table 30 Current vs., new tool chain used during software integration and in case of software reuse

EMS Development phase	Current tools	New tools			
		Tool	**Rational**	**Example**	**Availability risk**
Software integration and analysis	- Software integration tool - XD tool - Timing data base	AUTOSAR models integration tool (AUTOSAR editor)	This tool is needed to enable the integration of the AUTOSAR models from different sub-systems	-	No availability risk because the AUTOSAR editor developed for the new EMS AUTOSAR architecture is intended to enable the integration of AUTOSR parts from different sub-systems
		AUTOSAR to scheduling analysis transformation tool	This tool is needed to generate automatically from AUTOSAR models a model understandable by a scheduling analysis tool	AUTOSAR-to-SymTA/S transformation tool	This kind of transformation tool is being currently developed by Symtavision, the SymTA/S provider
		Scheduling analysis tool	This tool is needed to perform scheduling analysis on the integrated AUTOSAR models	SymTA/S	No availability risk as SymTA/S is already available and it satisfies all the scheduling analysis needs for automotive systems. Let's note that the license of this tool costs between 30000 and 40000 Euros.
Development by reuse with the methodology	-	XD to AUTOSAR editor transformation tool	This tool is needed to speed the transformation of the old software architecture represented in XD to an AUTOSAR architecture in case of reuse of software modules.	-	No availability risk as the AUTOSAR editor developed for the new EMS AUTOSAR architecture is intended to allow importing models represented in XD.

As the tables show, several new tools that are not currently used by Continental engineers are required by our methodology. However, these tools are already available and mature enough and engineers can be trained to them. Some tools such as the tool to transform EAST-ADL functional architecture to an AUTOSAR architecture are available but need to be improved. Nevertheless, the evaluation of this tool in Continental showed that it can be improved easily for an accurate integration in the development process.

3.3. Methodology Tooling

In this section, we present the tools that have been developed to facilitate the use of our methodology by Continental engineers. To model our methodology, the Eclipse Process Framework (EPF) [63] is used to represent the phases and activities of our methodology. To guide Continental engineers through the use of our methodology, we implemented also Eclipse cheat sheets that engineers can follow when using the methodology. Finally, a set of model checking rules have been implemented to ensure the development of consistent models during the analysis and design phases using the papyrus tool.

3.3.1. Methodology EPF Model

Figure 54 shows a simplified overview of the EPF model developed for the methodology. The figure shows the modeling of the different activities that should be performed during each phase. In addition, the figure details the tasks that should be performed to develop the analyzable models during the analysis and implementation phases.

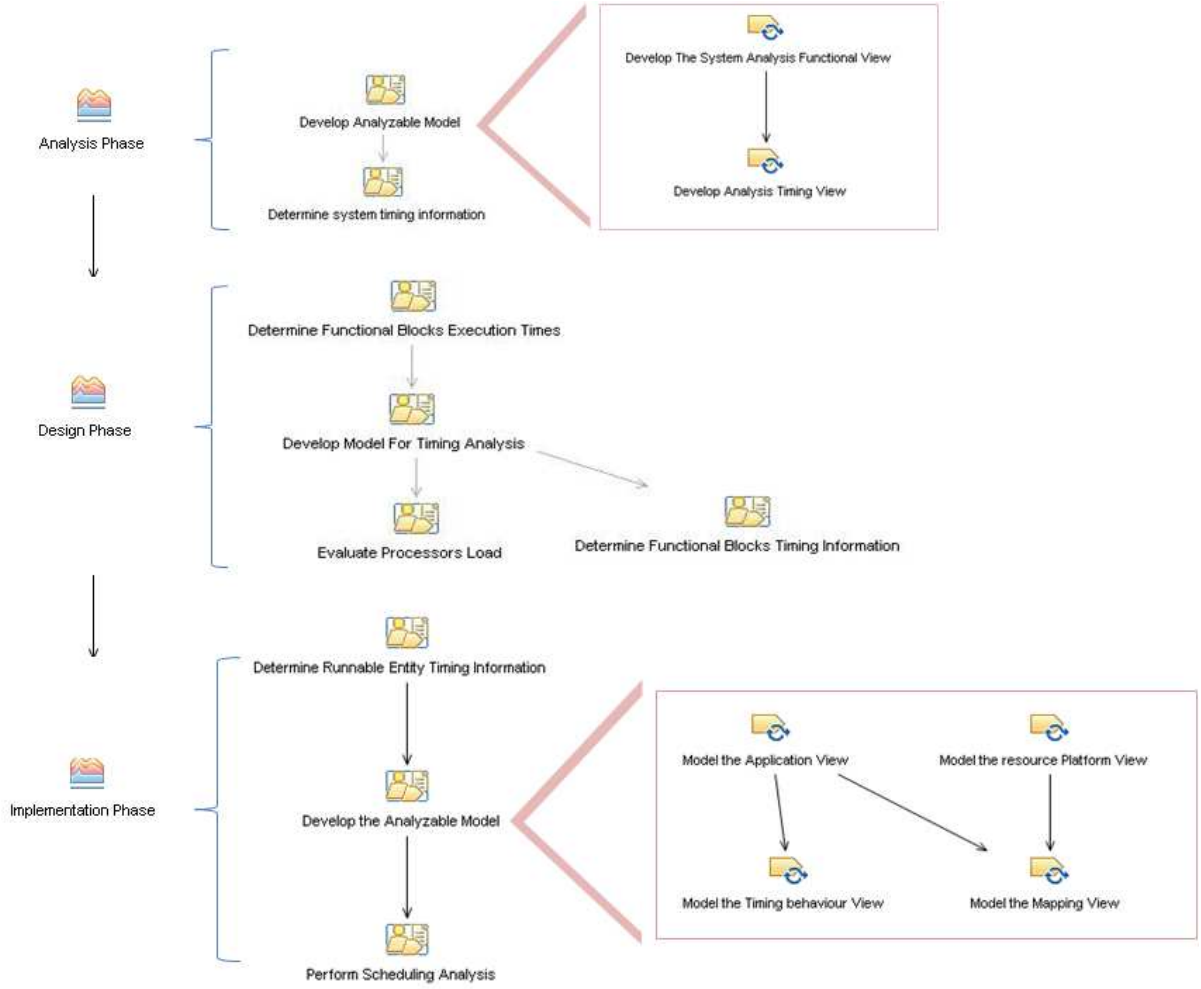

Figure 54 Simplified EPF model of the methodology

3.3.2. Cheat Sheet Guides

To guide engineers through the use of our methodology, we implemented a set of eclipse cheat sheets that describe in details the steps to follow to develop the models needed during each phase of the development. Figure 55 shows an example of a cheat sheet that describes the modeling steps that should be followed to develop the timing view during the design phase. These cheat sheets can be added as a plug-in to the Eclipse platform and can be used by Continental engineers as a part of the Eclipse help.

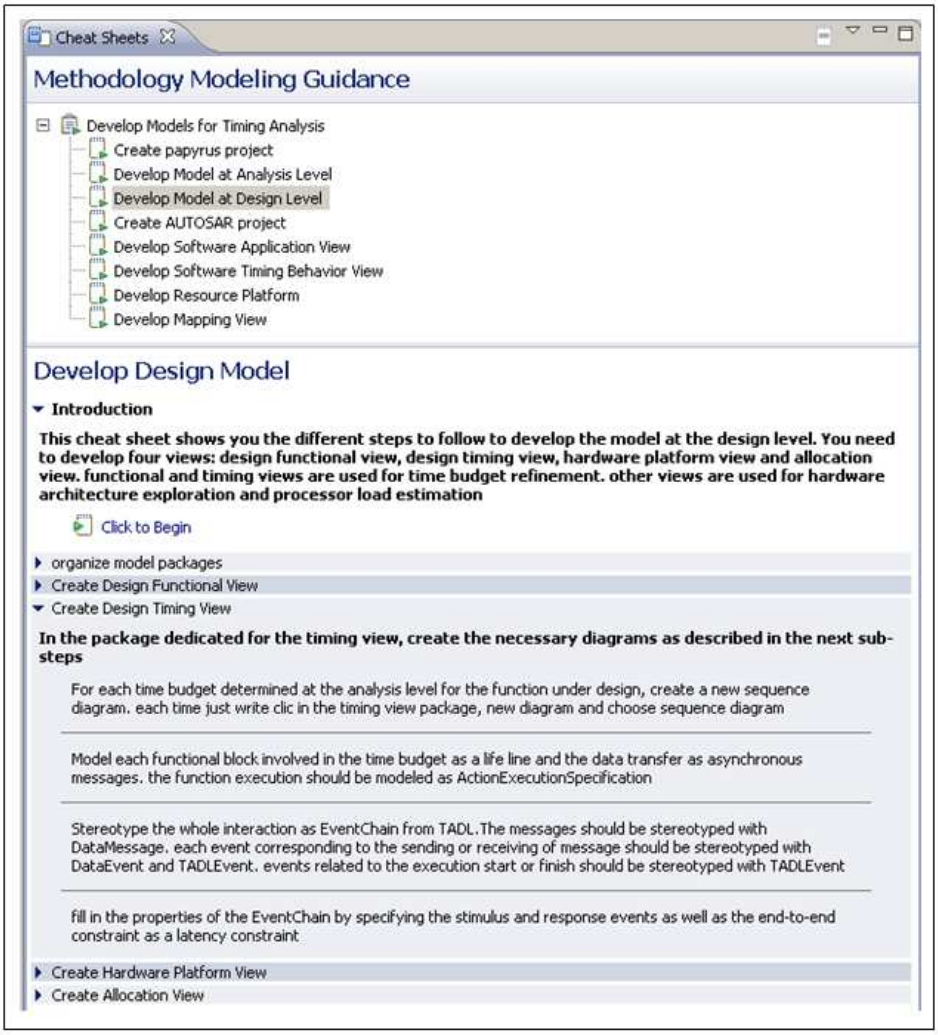

Figure 55 Example of an implemented Eclipse sheet cheat

3.3.3. Model Validation Rules.

To ensure the development of consistent models, we used the EMF (Eclipse Modeling Framework) [64] Validation mechanisms to implement modeling rules against which Continental engineers can verify the correctness of their Papyrus models during analysis and design phases. These rules are developed as constraints implemented in Java. Figure 56 shows an example of the validation of a timing view developed using Papyrus MDT. To validate this view, we implemented a Java constraint telling that messages should be stereotyped with the stereotype "DataMassage" as described during the methodology presentation. As the figure shows, once the validation is launched, the tool detects that this constraint is not respected and an error message notifies the designer about the problem.

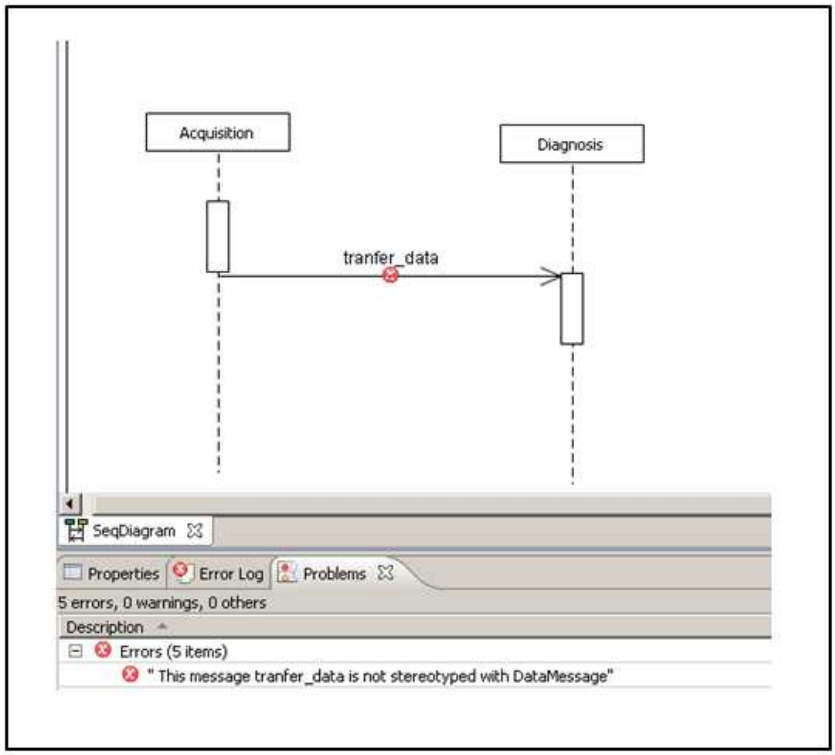

Figure 56 Example of modeling rule validation in Papyrus

4. Methodology General Validation

This section aims at highlighting the added value of the proposed methodology through showing at which extent it allows satisfying the automotive needs presented in the first part of this manuscript. To validate our methodology we select hence a set of validation criteria that reflect these needs. As denoted in the first part of this manuscript, the approach should allow:

- Reducing development time and cost
- Mastering system complexity
- Providing a seamless development process based on a seamless tool chain
- Ensuring system dependability, especially timing correctness through verification and validation.

Table 31 summarizes the capabilities of the methodology against these needs.

4.1. System Complexity Mastering

In our process, a top-down approach is followed, whereby system architecture is detailed and refined from one phase to another. During the early development phase (analysis), for example, the focus is only on system functional architecture, thus abstracting away the complexity that is potentially inherent in hardware or implementation details. This architecture is further refined during the design phase, and the general features of its hardware platform are described. Finally, the software and hardware architectures are supplemented with all related implementation details. In this way, the complexity of the architecture described increases gradually from one phase to another, allowing engineers to focus during each on particular views of the system. The complexity of timing analysis also increases gradually from phase to phase. During analysis phase, the focus is made only on timing validation of the architecture. During design phase, this validation continues and is enhanced by validation of the hardware platform. Finally, after the integration stage, a complete scheduling analysis is performed to validate both the timing and performance constraints.

4.2. Development Time and Cost Reduction

Development time and cost reduction is enabled by our methodology basically through the early detection of time-related failure. In fact, if we consider the current EMS development

process at Continental, timing verification is performed currently very late after the integration of the EMS and is based on tests and measurements (measurement of task response times based on the C code of the integrated system). In case of failure detected during this phase, the correction of such failure is very time-consuming. In fact, the knowledge of the failure source is very difficult (which sub-system is involved? at which stage a design mistake was made? etc…). With our methodology, timing analysis starts since the very early phase of the EMS design. By identifying EMS end-to-end requirements and assigning time budgets to the different sub-systems based on these requirements, we ensure that these requirements remain respected when developing the software of each sub-system. In addition, during the software development process for each sub-system, timing analysis is performed in each phase to ensure the correctness of the architecture designed and hence the possibility to move to the next phase. Hence, if a failure is detected by the scheduling analysis performed on the integrated system, we do not need to spend more time to go back to the early design phases (as the architecture designed during these phases is already validated).

Furthermore, the current timing analysis performed on the EMS (after the system integration) is more time-consuming than the scheduling analysis activity that we propose in our methodology. In fact, for an EMS configuration containing almost 20 OS tasks, the measurement of OS task response times (using C code) at a fixed engine speed takes nearly four days as it requires modifying the C code of the integrated system to get analyzable C code. In addition the tool used currently to measure these response times takes nearly two days to analyze the code. For the same EMS configuration, performing scheduling analysis using the SymTA/S tool takes only one day knowing that the SymTA/S model for the EMS architecture was described manually. This duration will be greatly reduced when the SymTA/S model is generated automatically form, e.g., AUTOSAR models.

In addition, unlike our approach, the current approach does not allow measuring the response times of engine-synchronous tasks; it gives results only for timing tasks. Hence, the results obtained do not reflect at all the real timing behavior of the system. Due to this, some real time failures may be detected only during the final tests on the vehicle itself which introduces extra time and cost to correct them.

To conclude, our process proposes to start timing analysis early. This allows engineers to also detect errors early and thus adapt the already developed architecture using models only.

It also saves any time that would otherwise have been lost, for instance, in correcting code to account for late error detection. Then again, since our development process is model-based, automatic transformations can be used to generate either models for the next phase (e.g. by transforming a model developed at the design stage into AUTOSAR software architecture) or the final code from the AUTOSAR implementation model. It also serves to automatically generate input models for the analysis tools. All of these features represent a huge reduction in development time and hence cost. In the cruise control use case presented previously, development of models in the different phases, together with timing analysis, took only three days, which is much less than the time usually required to develop software based on classical approaches (code-centric approaches).

4.3. Seamless Development Process

Our methodology gives guidance for a seamless development and timing analysis process. In fact, unlike existing approaches that we described in the first part of this manuscript, our methodology gives guidance for model refinement from a phase to another. In addition, it describes how analysis results of each phase should be used for architecture refinement during the next phase.

Moreover, the methodology describes the tool chain that should be used during the development process for both the modeling and timing analysis activity. Based on the acceptability study of the methodology, there is a good potential for an easy adoption of this tool chain at Continental.

4.4. Enabling Timing Verification

The first objective of our proposed process was to enable the integration of timing analysis along the development process. Compared with available approaches presented in the first part of this manuscript, our methodology gives detailed guidance allowing performing timing analysis and verification from early design phases until implementation and integration. Furthermore, compared with current EMS development process where timing analysis is performed only at the integration stage, our process enables starting timing analysis since the very early design phases. In fact during the analysis phase, sub-system time budgets are determined to ensure compliance with vehicle end-to-end requirements. Then, these budgets are refined during the design phase to determine the functional block time budgets. These latter time budgets represent the constraints that are verified during the implementation phase through performing scheduling analysis. In addition, the

validation of the hardware platform is started during the design phase, based only on allocation of functional blocks to hardware resources. This model is refined during the implementation phase by adding the software resources and the mapping of the runnable entities to these resources. This way, our methodology enables during the early phases (analysis and design) a sort of "preparatory analysis" that paves the ground for the scheduling analysis activity performed after EMS integration.

In addition, in this work, we showed how to move from modeling and design activities to timing analysis activities by presenting guidance for model development and refinement.

Table 31 Methodology capabilities

Software development needs	Methodology capabilities
Master system complexity	Development through abstraction levels (From abstract functional description to detailed implementation). The complexity of the designed architecture increases gradually from a phase to another Enable the designer to focus on different aspect at different design phases The complexity of the timing analysis increases gradually (evaluation of time budgets, then evaluation of hardware resources utilization and finally complete scheduling analysis)
Reduce development time and cost	Early detection of design mistakes, Reduce time and cost due to correction of last-minute detected errors The scheduling analysis proposed by the methodology is less time consuming than the approach used currently in continental Automatic transformation of models can be used to accelerate the development and the timing analysis
Define seamless development activity	Gives guidance for model refinement and transformation (from analysis to design phase, from design to implementation phase, from modeling to analysis tools) A tool chain for modeling and timing analysis is defined to cover the whole development process
Enable timing verification	Detailed guidance for integration of timing analysis in the development process Enables starting timing analysis during early design phases Guidance for development of analyzable models during each development phase

Conclusion and Perspectives

In this thesis work, we presented a methodology for a model-based timing analysis process. This work has been done to make up for the lacks of some existing approaches that attempted to provide solutions for automotive software development needs.

Today, four major challenges are to be met in automotive software development domain: 1) Reduce software development time and cost, 2) master system complexity during development, 3) provide a seamless development process based on a seamless tool chain and 4) ensure system correctness through enabling early validation and verification. Among the important aspects to be verified for automotive software is the correctness of its timing behavior.

In order to provide solutions to some of these needs, many model-based development approaches and methodologies have been defined. Some of these approaches are automotive domain specific such as the approaches defining the EAST-ADL, TADL and AUTOSAR modeling languages. Other approaches are dedicated to real-time systems in general like the modeling language MARTE. These approaches give modeling means and concepts that allow describing several aspects of the developed system (application, platform, timing, allocation, etc). However, although these approaches give some solutions for the above-mentioned automotive needs, they remain incomplete in term of enabling timing verification along the development process.

To make up for this lack, we propose in this thesis work a methodology that allows integrating timing analysis, mainly scheduling analysis, in a model-based development process that we defined based on the existing approaches.

First, we studied the feasibility of our approach which combines model-based development for automotive applications and scheduling analysis. On one hand, this feasibility study is based on the evaluation of the expressivity of the available modeling languages for enabling scheduling analysis. On the other hand, the study is based on the evaluation of the usability of scheduling analysis to enable timing verification for automotive systems. This is done through evaluating the capabilities of available scheduling analysis tools to satisfy scheduling analysis needs for automotive applications.

Our approach is based on the definition of a model-based timing analysis process. This process is composed of three development phases; analysis, design and implementation phase. During the early design phases, analysis and design, analyzable models are developed using the EAST-ADL constructs for functional modeling, TADL means for timing modeling and MARTE concepts to model allocation. Based on the developed analyzable model, timing analysis is performed to determine time budgets to allocate either to the developed sub-system itself or to its functional blocks. The time budgets determined during each phase ensure respecting the end-to-end timing requirements of that phase. During design phase, a hardware architecture exploration is also performed to determine the best functional block-to-ECU allocation scenario based on the evaluation of ECU loads. During the implementation phase, the complete software architecture is described and scheduling analysis is performed to verify whether the system respects the timing constraints determined by the timing analysis carried out during previous phases.

In this thesis work, we presented also an approach describing how to apply our methodology for the development of Engine Management Systems (EMS) at Continental. First, we studied the EMS current development process at Continental. Then, an approach describing the application of our methodology in the context of this development process is defined. This approach focuses on two development scenarios; software development from scratch and software development by reuse. Based on the above-mentioned application approach, we studied also the acceptability of our methodology by measuring the gap between this methodology and the current EMS development process in terms of tasks, skills and tool chain. This acceptability study reveals a strong potential of our methodology to be adopted by Continental engineers especially that, as a result of this thesis work, the AUTOSAR formalism is being currently deployed for new EMS architecture at Continental.

The most important added value of our methodology is enabling early detection of timing errors during the development process. This allows avoiding last-minutes detected mistakes and hence saving time and cost required for correcting the software implemented.

Our methodology gives also a seamless development and timing analysis process that is based on seamless tool chain for architecture modeling and timing analysis. The different development phases defined allows describing the system architecture in a progressive way from abstract functional description until detailed implementation. This allows, hence designers mastering the complexity of the designed architecture and give them the

possibility to focus each time only on particular aspect of the architecture (functional description, hardware, timing, etc).

Although our methodology gives several solutions to meet the automotive software development challenges at Continental, some points should be improved in further works:

In our methodology, we suggest to perform scheduling analysis based on a self-contained AUTOSAR software architecture. To perform scheduling analysis, one needs to specify task or function execution times. However, in our methodology this is done based only on estimation and designer expertise without giving any formal approach describing how these execution times can be determined. In case of development by reuse, these execution times can be determined based the execution times measured from the C code of previous software version. In the case of the cruise control use case (development from scratch), the execution times have been determined based on the application expert knowledge. However, this remains insufficient and there is a need to define a formal approach allowing the determination of such execution times.

In addition, from a practical point of view, we presented an approach to apply our methodology for EMS development at Continental. However, for the software description, we do not describe in detail how constructs used currently to describe EMS software architecture can be mapped to AUTOSAR concepts. This work is being carried out by another team at Continental. It aims at adapting AUTOSAR concepts and means for EMS software architecture specificities.

A further topic that is not presented in this work is the design of an AUTOSAR software platform (OS tasks) that ensures the timing correctness of the designed system. In fact, an approach should be developed to describe how, based on the timing properties of AUTOSAR runnable entities (deadlines, end-to-end constraints, periods, etc), a task model respecting these properties should be designed. For example the following questions should be answered:

- How to define OS task deadlines
- How to define task priorities
- How to define task periods/ activation patterns

Annex 1 shows an example of a work in progress that is performed in this thesis work to solve this problem. The annex shows mainly an approach to define OS task deadlines based on the deadlines and end-to-end constraints imposed on the AUTOSAR runnable entities.

Annex1: Definition of an AUTOSAR OS Task Model

At the implementation level (based on AUTOSAR software architecture), to enable scheduling analysis, the designer should define the OS tasks that constitute the software resource platform. To define a complete task model, the designer should make some choices to answer the following questions:

1. How to define the OS tasks of the system
2. How to assign the priorities to these tasks
3. How to determine the deadlines for these tasks based on the runnable deadlines and end-to-end constraints
4. What are the activation patterns and the recurrences of these tasks.
5. How to define the "preemptivity" kind of each task (which tasks are preemptive/cooperative)

1. OS tasks choice: generalities

When choosing the OS tasks, the designer should take into account the characteristics of the runnable entities to be mapped to these tasks. In fact, the designer has as input a set of runnable entities submitted to a number of constraints such as deadlines or end-to-end constraints and characterized by recurrences and execution times. Based on this information, the designer should decide about the properties to assign to each chosen task (priority, deadline, etc). Of course, the choice of the task model should be done in an accurate and optimized way. For example, to optimize the CPU load resulting from task switch overheads, the designer should try to minimize at maximum the number of chosen tasks while keeping, at the same time, an efficient task model.

2. Task priorities

Here, to comply with AUTOSAR OS, we consider a fixed priority task model, i.e. task priorities are fixed before system execution and do not change at runtime. When assigning priorities to chosen tasks, the designer should consider both the timing constraints of the runnable entities mapped to these tasks (deadlines and end-to-end constraints) and their execution times. Runnable entities having small deadlines (i.e.

representing urgent treatments) should be mapped to tasks for which the designer should assign high priorities (deadline monotonic way). Execution times of mapped runnables should also be considered in order to prevent tasks from being delayed by higher priority task having a large execution time. In such task case (tasks with large execution time), the designer should assign a low priority to these tasks and allow them to have pre-emption points (schedule points) in order to give the possibility to higher priority tasks to execute without waiting the termination of these task. Moreover, when assigning priorities to tasks, one should consider its deadline value but also the criticality of the treatment associated.

3. Task deadlines

Task deadlines should be determined based on the deadlines and end-to-end constraints of the mapped runnable entities.

A. Case1: System with only deadlines on runnables (no defined end-to-end constraints on flows of runnables)

In this case, the designer has as input a set of runnable entities, each runnable has got a deadline, a recurrence and an execution time. Of course, it is not optimal to create a task for each runnable and assign the runnable deadline to this task. So the designer should find a solution to map many runnable entities to the same task for which he chooses a deadline that ensures respecting all the deadlines of the mapped runnables. To do so, the designer determines, first based on his expertise, groups of runnables to be mapped to the same task (these groups are formed by runnables with deadlines that are close to each other). For each group of runnables we define a "deadline class". This represents the smallest runnable deadline in the group. The task to which we map the runnables of this group will have as deadline this deadline class. To avoid a very pessimistic design the designer should adapt the definition of the groups and the repartition of the runnables based on the following constraint: for each runnable entity, to belong to a group, the difference between the deadline of this runnable and the deadline class of the group should be smaller than a certain value that we denote X. This value is chosen by the designer based on his expertise

Formulation

Let's consider a system defined by R = {re_1, re_2,….., re_n}, R is a set of runnable entities re_i (i ϵ{1. ..n}). Each runnable entity re_i is defined by (p_i, d_i, ex_i), p_i is the runnable recurrence, d_i is its deadline and ex_i is its execution time. The designer determines a set of groups of runnable entities G= {g_1….g_s}. Each group g_m is defined as follows: g_m = {re_j…re_k} (m ϵ {1...s} and j, k ϵ {1...n}). For each group g_m we define a deadline class d_{gm} = min d_r, r ϵ {j…k}

Contraints:

1. A runnable entity re_i belongs to a group g_m if and only if $|(d_i - d_{gm})| \leq X$

Example:

Let's consider the runnable entities of table 1:

Table 1 Example of runnable entities and their deadlines

Runnable	Deadline
RE0	200 µs
RE1	1 ms
RE2	2 ms
RE3	3 ms
RE4	10 ms
RE5	12 ms
RE6	20 ms
RE7	25 ms
RE8	100 ms
RE9	101 ms
RE10	26 ms

Based on his expertise, the designer will determine a first repartition of these runnables into groups (runnables with deadlines that are close to each other will belong to the same group). Table 2 gives a repartition of runnable entities into groups

Table 2 Example of a repartition of runnable entities to groups

Group	Runnables	Deadline class
g1	RE0	200 µs
g2	RE1, RE2, RE3	1 ms
g3	RE4, RE5	10 ms
g4	RE6, RE7, RE10	20 ms
g5	RE8, RE9	100 ms

This is a first repartition of the runnable entities based on the designer expertise

Now suppose that the bound X is equal to 1ms (i.e. *A runnable entity* re_i *belongs to a group* g_m *if and only if* $| (d_i - d_{gm}) | \leq 1$ *ms)*

In this case, RE3 can not belong anymore to the group g1, so we should put it in new group: g23 = {RE3} having a deadline class equal to 3ms. It is also the case for the RE5 that can't belong anymore to g3 so we assign it to a new group g34 = {RE5} with 12ms as a deadline class, this is also true for the runnable RE7 that can not belong to g4, so we create a group g45= {RE7} with 25ms as a deadline class. RE10 should also be removed from g4. As the difference between the deadline of RE10 and the deadline class of g45 is less or equal to 1ms, we should put RE10 in the group g45

Finally we end up with the repartition of table 3:

Table 3 Repartition obtained

Group	Runnables	Deadline class
g1	RE0	200 µs
g2	RE1, RE2	1 ms
g23	RE3	3 ms
g3	RE4	10 ms
g34	RE5	12 ms
g4	RE6	20 ms
g45	RE7, RE10	25 ms
g5	RE8, RE9	100 ms

So, to ensure the respect of the deadlines of these runnables preventing a very pessimistic design, we define the following tasks presented in table 4:

Table 4 Defined tasks

Task	Task deadline	Mapped runnables
T1	200 µs	RE0
T2	1 ms	RE1, RE2
T3	3 ms	RE3
T4	10 ms	RE4
T5	12 ms	RE5
T6	20 ms	RE6
T7	25 ms	RE7, RE10
T8	100 ms	RE8, RE9

B. Case 2: System with runnables deadlines and end-to-end constraints

In this case, there are two possible configurations: either end-to-end constraints are imposed on independent end-to-end flows of runnables (i.e. constrained end-to-flows have no common runnable entities, figure 1) or these end-to-end flows have common runnables (figure 2)

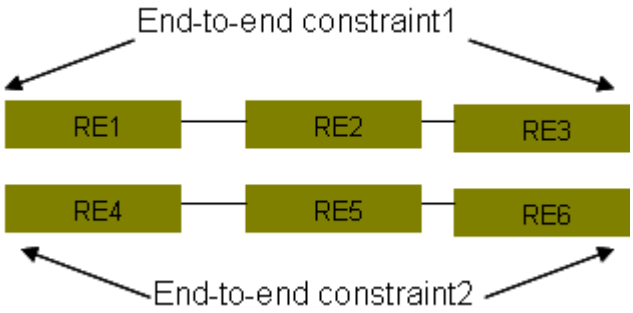

Figure 1 Example of independent end-to-end flows

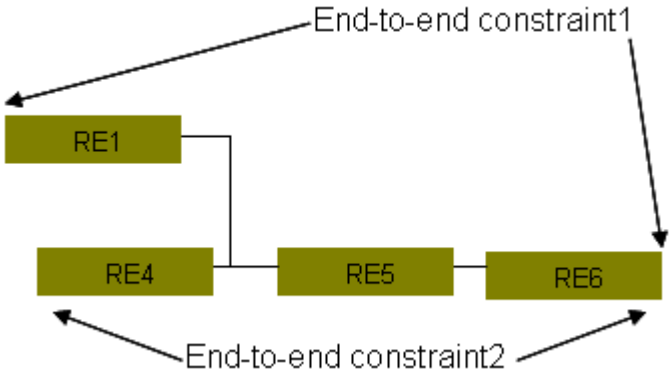

Figure 2 Example of dependent end-to-end flows

B.1: system with independent end-to-end flows:

In this case, the designer considers each constrained end-to-end flow as a unique runnable entity formed by the succession of the runnables of this end-to-end flow and having as deadline the end-to-end constraint imposed on this end-to-end flow (figure 3)

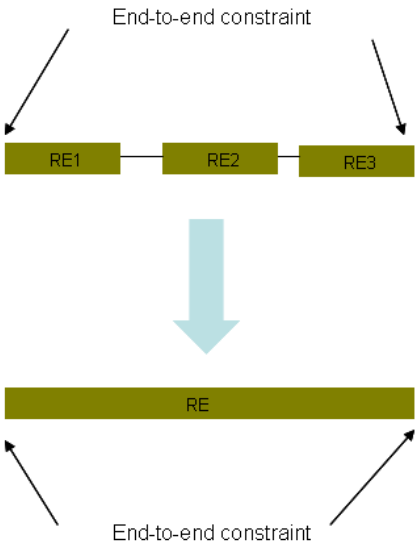

Figure 3 End-to-end flow transformation

This means that all the runnables belonging to a constrained end-to-end flow will be mapped to the same task. The designer performs then the same work described above by defining groups of runnables with deadline classes.

B.2: system with dependent end-to-end flows:

Here, we have also two cases: either we are allowed to map a runnable to more than one task (i.e. a runnable can be called by more than one task) or each runnable should be mapped to exactly one task. In the first case, the work is easy and is the same as the work described in B.1: each end-to-end flow is considered as a unique runnable and will be mapped to a task. Runnables belonging to more than one end-to-end flow may be mapped to more than one task.

In the second case (when a runnable cannot be mapped to more than one task) the problem concerns mainly the runnables that belong to more than one end-to-end-flow. The designer separates the runnable entities in two groups: the first group contains the runnable that does not belong to any constrained end-to-end flow or to only one end-to-end flow. The second group contains the runnables that belong to more than one constrained end-to-end flow. He performs then the same work described in A for the runnables of the first group. Then based on his expertise, and the formed groups, he assigns the remaining runnables (i.e. the runnables belonging to more than one end-to-

end flow) to the formed group in a way that the global end-to-end constraint will be respected

4. Task recurrences

The problem of choosing task recurrences is similar to the problem of choosing task deadlines. The designer has as input a set of runnable entities having recurrences and should be mapped to tasks for which we assign recurrences that should respect the runnable recurrences. The designer should make a trade-off between the choice of task recurrences and the choice of task deadlines. How this trade-off should be made?

5. Task preemptivity

Choosing the preemptivity kind for a task means to choose between three categories of tasks: either a task is fully preemptive, fully non-preemptive or cooperative. What is the criterion on which the preemptivity kind is chosen?

References

[1] B. Selic, "A Generic Framework for Modeling Resources with UML", IEEE Computer vol. 33 no.6, pp.64-69, June 2000.

[2] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Gonzalez Harbour, "A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems", Kluwer Academic Publishers, 1993.

[3] J. J. P. Tsai, S. J. Yang and Y.-H. Chang, "Timing constraint Petri Nets and their Application to Schedulability Analysis of Real-time System Specification", IEEE Transactions on Software Engineering, vol. 21, n° 1, pp. 32-49, 1995.

[4] D.C. Petriu, C. Shousha, A. Jalnapurkar, "Architecture-Based Performance Analysis Applied to a Telecommunication System", I.E.E.E. Transactions on Software Eng, Vol.26, No.11, pp.1049-1065, Nov. 2000.

[5] R. Alur and D. L. Dill, "A Theory of Timed Automata," Theoretical Computer Science, vol. 126, pp. 183–235, 1994.

[6] A. Pretschner, M. Broy, I. H. Kruger, T. Stauner, "Software Engineering for Automotive Systems: A Roadmap", 29th International Conference on Software Engineering (ICSE 2007), Minneapolis MN, USA, May 27th , 2007.

[7] Object Management Group: Unified Modeling Language -- Superstructure Version 2.1.1 formal/2007-02-03

[8] I. T. Union, "Specification and Description Language", International Telecommunication Union Recommendation Z.100, 1992.

[9] Avionics Architecture Description Language Standards Document (AADL), http://www.aadl.info.

[10] MARTE website. www.omgmarte.org

[11] TIMMO website. www.timmo.org

[12] S. Gérard, "Modélisation UML Exécutable pour les Systems Embarqués de l'automobile", PhD Thesis.

[13] H. Espinoza, "An Integrated Model-Driven Framework for Specifying and Analyzing Non-Functional Properties of Real Time Systems", PhD Thesis.

[14] L. Fuentes, A. Vallecillo: "An Introduction to UML Profiles", UPGRADE, The European Journal for the Informatics Professional, 5(2):5-13, April 2004, ISSN: 1684-5285.

[15] Object Management Group: Object Constraint Language (OCL). OMG Available Specification. Version 2.0 (2006)

[16] M. Fowler, "Domain Specific Languages" (Book), ISBN: 0321712943 9780321712943

[17] EAST-ADL website. www.atesst.org

[18] AUTOSAR Partnership. www.autosar.org

[19] J. L. Peterson, "Petri Net Theory and the Modeling of Systems", Prentice Hall, 1981

[20] P. L. Guernic, T. Gautier, M. L. Borgne and C. L. Maire, "Programming Real Time Applications with SIGNAL", INRIA-RENNE, report N 1446, 1991

[21] Liu, C. L. and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment". Journal of the ACM (Association for Computing Machinery), Vol. 20 n°1, Jan.1973

[22] Lehoczky, J. P., L. Sha, and D. Y. Ding: 1989, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", 10th IEEE Real-Time Systems Symposium (RTSS1989), Santa Monica CA, USA, December 5 – 7th, 1989.

[23] Joseph, M. and P. Pandya: 1986, "Finding Response Times in a real-time system. BCS Computer Journal, Vol. 29, n°5, 390-395.

[24] N. C. Audslay, A. Burns, M. F. Richardson, and A. J. Wellings: "Hard real-time Scheduling: The Deadline Monotonic Approach", 8th IEEE Workshop on Real-Time Operating Systems and Software. Atlanta, GA, USA,

[25] Leung, J. Y. T. and J. Whitehead: 1982, On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation (Netherlands)*

[26] Lehoczky, J. P.: 1990, Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proc. 11th IEEE Real-Time Systems Symposium.

[27] Tindell, K., A. Burns, and A. J. Wellings: 1994a, An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems*

[28] K. Tindell, Adding Time-Offsets to Schedulability Analysis, Technical Report YCS 221, Dept. of Computer Science, University of York, England, January 1994.

[29] J.C. Palencia Gutiérrez and M. González Harbour, Schedulability Analysis for Tasks with Static and Dynamic Offsets. Proceedings of the 18th. IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998.

[30] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99), 1999

[31] MAST website (Mast.unican.es).

[32] Cheddar website (http://beru.univ-brest.fr)

[33] Rapid-RMA website (http://www.tripac.com/rapid-rma)

[34] L. Sha, T. Abdelzaher, K. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A. Mok, Real Time Scheduling Theory: A Historical Perspective, Real-Time Systems Journal, November-December 2004.

[35] Chronval website (http://www.inchron.com/chronval.html)

[36] SymTA/S website (http://www.symtavision.com/symtas.html)

[37] M. Traub, V. Lauer, J. Becker, M. Jersak, K. Richter and M. Kuhl: Using timing analysis for evaluating communication behaviour network topologies in an early design phase of automotive electric/electronic architectures. SAE World Congress, Detriot, MI, USA, April 2009

[38] OSEK group website (http://www.osek-vdx.org)

[39] K. Tindell and J. Clark: Holistic Schedulability Analysis for Distributed Real-time Systems. Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems), 40:117–134, 1994

[40] F. Singhoff, J. Legrand, L. Nana and L. Marcé: Cheddar: a Flexible Real Time Scheduling Framework. ACM SIGAda Ada Letters, volume 24, number 4, pages 1-8. Edited by ACM Press, New York, USA. December 2004, ISSN: 1094-3641.

[41] J. M. Drake, M. G. Harbour, J. J. Gutiérrez, P. L. Martinez, J. L. Medina, J. C. Palencia : Modelling and Analysis Suite for Real Time Applications (MAST 1.3.7), Description of the MAST Model. Report, Universidad De Cantabria, SPAIN, 2008.

[42] Papyrus website (www.papyrusuml.org)

[43] Webpage of the ARTOP User Group (www.artop.org)

[44] Simulink website (www.mathworks.com/products/simulink)

[45] Saoussen Anssi, Sara Tucci-pergiovanni, Chokri Mraidha, Arnaud Albinet, François Terrier, Sébastien Gérard, "Completing EAST-ADL with MARTE for Enabling Scheduling Analysis for Automotive Applications", Embedded Real Time Software and Systems (ETS²2010), Toulouse, France, May 19th - 21st, 2010

[46] P. Cuenot, P. Frey, R. Johansson, H. Lönn, M. O. Reiser, D. Srevat, R. Tavakoli Kolagari, D. J. Chen, "Developing Automotive Products using the EAST-ADL2, an AUTOSAR Compliant Architecture description Language", Embedded Real Time Software (ERTS2008), Toulouse, France, January 29- 31st, February 1st, 2008.

[47] EAST-ADL Specification, Version 2.1, 2010-06-02.

[48] H. Blom, R. Johansson, H. Lönn, "Annotation with Timing Constraints in the Context of EAST-ADL2 and AUTOSAR, the Timing Augmented Description Language", Workshop on the Definition, evaluation and exploitation of modeling and computing standards for Real Time Embedded Systems (STANDRTS'09), Dublin, Ireland, June 30th, 2009.

[49] TADL: Timing Augmented Description Language Specification, version 2, 2009-10-05.

[50] AUTOSAR Methodology Specification, version 1.2.2, release 4.0, 2008-08-15.

[51] AUTOSAR Software Component Template, version 4.0.0, release 4.0, 2009-09-15.

[52] AUTOSAR Basic Software Module description Template, version 2.0.0, release 4.0, 2009-11-13

[53] AUTOSAR Specification of Operating System, version 4.0.0, release 4.0, 2009-11-30.

[54] AUTOSAR System Template, version 4.0.0, release 4.0, 2009-12-04.

[55] AUTOSAR Specification of Timing Extension, version 1.0.0, release 4.0, 2009-11-30.

[56] UML profile for MARTE (specification), version 1.0, November 2009.

[57] K. Albers, "Approximative Real Time Analysis", PhD Thesis, 2010

[58] AUTOSAR Specification of RTE, version 3.0.0, release 4.0, 2009-12-18.

[59] SymTA/S 1.4 Intro and Theory Manual, report, version 1.4.2, 2009.

[60] C. Mraidha, S. Tucci-Piergiovanni, S. Gérard. Optimum: A MARTE-based Methodology for Schedulability Analysis at Early Design Stages. Third IEEE International workshop UML and Formal Methods. November 2010, Shangai, China.

[61] E. Wozniak, C. Mraidha, S. Gerard and F. Terrier: "A Guidance Framework for the Generation of Implementation Models in the Automotive Domain", 2nd international workshop DANCE, (Distributed Architecture modeling for Novel Component based Embedded systems), held in conjunction with SEAA 2011 the 37th Euromicro conference, 2011.

[62] S. Gérard, D. Servat: "Proposal for an EAST-ADL2 Annex to MARTE", (report), 2008.

[63] Eclipse Process Framework website (http://www.eclipse.org/epf/)

[64] Eclipse Modelling Framework website (http://www.eclipse.org/modeling/emf/)