THÈSE

pour l'obtention du Grade de

Docteur de l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique

(Diplôme National - Arrêté du 25 mai 2016)

Ecole Doctorale : Sciences et Ingéniérie des Systèmes, Mathématiques, Informatique
Secteur de Recherche : Informatique et Applications

Présentée par :

Anh Toan Bui Long

****

Contributions à la conception à base de modèles des systèmes temps réel en vue de leur analyse de performance temporelle

****

Directeur de thèse : **Emmanuel Grolleau**
Co-encadrant de thèse : **Yassine Ouhammou**

****

Soutenue le 20 décembre 2018
devant la Commission d'Examen

****

JURY

Président :

Yamine AÏT-AMEUR	Professeur, INP Toulouse - ENSEEIHT, Toulouse

Rapporteurs :

Christian ATTIOGBÉ	Professeur, Université de Nantes, Nantes
Jérôme HUGUES	Professeur, ISAE - SUPAERO, Toulouse

Membres du jury :

Liliana CUCU-GROSJEAN	Chargée de recherche (HDR), INRIA, Paris
Emmanuel GROLLEAU	Professeur, ISAE - ENSMA, Poitiers
Yassine OUHAMMOU	Maître de Conférences, ISAE - ENSMA, Poitiers

Intelligence is the ability of a system to adjust appropriately to a changing world.

— Christopher Evans, *The Micro Millennium*, p.157, 1979

Remerciements

Pour tout travail accompli, des remerciements s'imposent et des personnes incontournables sont venus me supporter dans ma thèse durant leur temps libre (ou du moins leur temps moins occupé). Ces remerciements risquent d'être plutôt une liste que des remerciements à proprement parler. Le dernier va vous surprendre.

Avant tout, mes premiers remerciements vont à Emmanuel Grolleau, mon directeur de thèse, ainsi qu'à Yassine Ouhammou, mon encadrant, pour m'avoir guidé tout au long de cette thèse, pour leur disponibilité, leurs conseils et pour les discussions très élaborées.

Reste ensuite dans les permanents de l'ENSMA ou du LIAS que j'ai côtoyé pendant mes années estudiantines : Alexandre G., pour sa franchise durant ma présence à L'ENSMA, Fred C. pour ses conseils avisés , Brice C. pour les longues soirées de jeu de plateau, le grand sportif Laurent G. que j' ai eu la chance de rencontrer, Claudine R., Bénédicte B., Bénédicte C., Corinne, Lydie et Isabelle pour leur aide inestimable, Vincent A. pour son humour et humeur distingué, Olivier G. pour son talent de rassemblement dans le journal, Fabrice P. pour m'avoir fait voyager dans toute la France, Rodolph pour son talent audiovisuel.

Avec leur lot de bonne humeur, j'ai eu la chance de rencontrer d'innombrables et d'inestimables personnes dans ma thèse (et souvent sous-cotées) à commencer par les doctorants du Lias, Guillaume P., un roi encore plus sous-côté, Thomas L. qui m'a fait découvrir le Lindy, Yves M., un grand génie incomparable, Anaïs C. qui ne me fait pas perdre mon côté geek, Ibrahim D. dit le français, Géraud F., le chocolatier en chef, Cyrille P., chocolatier désigné, Jorge G., chocolatier actuel, Olga G., Abdelkarim A., passés dans les bureaux furtivement mais de façon remarquable, Sana N., remarquable par elle-même. Je n'oublie pas cependant, les stagiaires passés par le laboratoire comme Pierre-Emmanuel de R., Adnene B., Mehdi A., Walid K., Lotfi T..

Sans transition, un grand merci également à tous ceux qui m'ont soutenu à l'extérieur de la thèse de près ou de loin, qui ont fait de moi ce que je suis aujourd'hui, avec leur humeur, qu'elle soit bonne ou non. Je les remercie tout simplement de faire partie de ma vie et réciproquement faire partie de leur vie. Dans cette catégorie, je peux remercier les thésards, Florian G., pour sa présence (in)défectible, Aurélien L., Caroline G., Armande H., Pierre V., pour leur débordante énergie, Sylvain H., Caroline R., pour leurs jeux de mots débridés, François B.-S., pour ses conseils avisés, mais aussi les doctorants de l'ADIIS, Mathieu N., Hermine C., Julien C., Pascal L., Lydie R. Angélique P., pour leur volonté indéfectible et dans les amis proches Bastien P., restaurateur en chef, Camille B., commis de cuisine, Romane V., Bastien N., des (presques) légendes, Éric C., master de son domaine, Stéphanie D., Adeline S., Kyung Y., toujours là quand on en a (pas) besoin. Je n'oublie pas les danseurs Carine, Nanou, Mary, Dounia, Roderick avec qui j'ai pu éviter d'acheter des pantalons plus grands durant cette thèse. Cette longue liste comporte forcément des oublis (volontaires ou non).

Sans aucun doute, je remercie également Arthur, Florent, 'Ny, Nicolas, Jean, Anaïs, Gader, Romain, ainsi que Max, Alex, Robin, Bertrand pour les (longues) soirées d'été et d'hiver. #GP Ils se reconnaitront.

Outre toutes ces personnes, un remerciement plus que spécial à Clément A. qui m'a soutenu, encouragé, et supporté sans faillir pendant l'écriture de cette thèse ainsi que la préparation de la soutenance.

Un thésard sans la famille n'est rien : elle est également mise à l'honneur ici, que ce soit mes frères, mes neveux ou ma mère qui ont soutenu dans cette démarche.

L'exception de ces remerciements : je ne remercie ni ma console de jeux ni ma propension à boire des bières, ni ma bonne humeur car sans eux, la thèse serait certainement terminée quelques mois plus tôt.

Et enfin, le meilleur pour la fin dit-on, je remercie moi-même, qui m'a soutenu sans faillir pendant près de 27 années à ce jour, qui m'a toujours aidé dans les moments difficiles et qui fut le moteur principal de ma vie.

Table des matières

Liste des figures

Liste des tableaux

Chapitre 1

Introduction générale

> À défaut d'un Tout inaccessible, ces divers *Quelque Chose* valaient sans doute mieux que *Rien*.
>
> — Maurice Agulhon

Sommaire

1.1 Contexte et motivation

Les systèmes embarqués, en particulier temps réel, sont de plus en plus présents dans la vie quotidienne, dans les systèmes avioniques, les drones, les automobiles, le multimédia, les robots autonomes, etc. Les systèmes temps-réel évoluent dans un environnement nécessitant l'adaptation du système à des stimuli externes et internes. La défaillance de tels systèmes peut être dangereuse pour les personnes ou pour le matériel, ce qui en fait des systèmes critiques. La correction de ces systèmes est déterminée, dans la partie logicielle, en partie par le résultat que les tâches rendent, domaine du génie logiciel, et en partie par le temps de réponse de ces tâches, domaine du temps-réel.

Le coût de développement des systèmes critiques augmente de plus en plus avec le nombre croissant de fonctionnalités gérées par les systèmes informatiques. A titre d'exemple dans le domaine aéronautique, le Boeing 747-400 mis en service en 1989, comptait 400 000 lignes de code contre près de quatre millions de lignes de codes pour le Boeing 777 mis en service en 1995 [DK04] voire plus de cent million de lignes de code pour l'Airbus A380 [WDD+12]. Une erreur lors de la conception de tels systèmes impliquerait une augmentation significative du coût de développement. Les erreurs de dimensionnement logiciel/matériel font partie des défauts qui doivent être détectés le plus tôt possible pour réduire les risques d'avoir à opérer une correction tardive lors de la conception du système.

Le dimensionnement logiciel/matériel est lié au problème d'ordonnançabilité, qui adresse la façon dont les ressources matérielles (calculateurs, bus) sont choisies et arbitrées pour répondre aux besoins d'exécution des fonctions et de communication entre celles-ci. Pour déterminer l'ordonnancabilité d'un système de tâches, des recherches ont été menées pour déterminer, de la manière la plus fiable possible, si le système est temporellement valide, c'est-à-dire si toutes les fonctions s'exécutent suffisamment rapidement par rapport à la dynamique du système et de son environnement. Cette notion de "suffisamment rapide" est généralement traduite sous formes d'exigences non fonctionnelles, dérivées sous formes de contraintes temporelles, typiquement des échéances. Les analyses d'ordonnançabilité (aussi appelées analyses de performances) sont multiples et fortement dépendantes du contexte logiciel et matériel du système. Cette multiplicité des analyses requiert de la part de l'ingénieur temps-réel (l'analyste dans la suite de ce mémoire) une grande connaissance du domaine temps-réel. Ceci lui permet de vérifier comment l'arbitrage des ressources associées à l'exécution des fonctions permet à ces fonctions de satisfaire leurs contraintes de temps, en utilisant des analyses offertes par la littérature [SAÅ+04, ZA04, DB11b], et implémentées dans des outils d'analyse temporelle. L'analyste doit, pour ce faire, correctement modéliser les fonctions et le matériel choisis par l'architecte système impliquant ainsi un temps supplémentaire pour l'évaluation du système. Réciproquement, l'architecte système doit s'approprier les analyses d'ordonnançabilité afin d'évaluer son système. En particulier, il est possible que l'analyse conduite par cet ingénieur ne soit pas exempte d'erreurs, pouvant potentiellement occasionner une augmentation significative des coûts de développement. Le rôle de l'architecte système et celui de l'analyste sont liés pour la détermination de la correction du système, cependant ces rôles représentent plusieurs métiers nécessitant des expertises différentes, de plus comme nous le verrons dans ce manuscrit, ces rôles sont très éloignés dans le cycle de vie en V classique utilisé majoritairement pour la conception de systèmes informatiques critiques. Il existe une lacune dans les outils et techniques qui ne permettent pas aujourd'hui à un architecte système d'avoir accès à l'expertise de l'analyste dès la phase de conception logiciel/matériel du système.

Les travaux de Y. Ouhammou en 2013 [Ouh13] permettent de combler une partie de cette lacune en proposant un langage pivot entre langages de description d'architecture logiciel/matériel et analyse de performance temporelle du système. Ces travaux sont au cœur de la définition d'un projet collaboratif, le projet FUI WARUNA [1], qui vise à définir un langage de modélisation et des

1. https://www.waruna-projet.fr/

outils destinés à l'architecte système lui permettant d'évaluer au plus tôt les performances temps réel de son architecture logiciel/matériel, ceci afin de grandement diminuer les risques d'erreur de conception découverte tardivement dans le cycle de vie logiciel. L'architecte système, alors qu'il définit l'architecture logicielle, est l'acteur, dans le cycle de vie logiciel, dont les choix de conception auront le plus d'importance dans la validité temporelle, ou non, du système. Cependant, aujourd'hui dans l'industrie, la phase de validation temporelle n'arrive qu'en toute fin de cycle de vie, exposant tous les projets de développement de systèmes temps réel à des risques de sur-coût important. Afin de s'en prémunir, il est fréquent que les systèmes à contraintes de temps soient fortement sur-dimensionnés à la conception, ce qui en augmente les coûts de fabrication, et en diminue l'autonomie, les ressources de calcul sur les systèmes embarqués, représentant une partie importante des systèmes temps réel critiques, étant typiquement alimentées sur batteries. Le but du projet est donc de proposer à l'architecte système un outillage lui permettant d'évaluer dès la phase de conception la correction temporelle de son système. Pour cela, il faut considérer en entrée des modèles utilisés par les architectes système, tels que les langages de description d'architecture logiciel/matériel, et proposer une méthode outillée offrant l'expertise de l'analyste, c'est-à-dire capable d'orienter le concepteur vers les outils de validation temporelle adaptés à son modèle d'entrée. Cependant, ces outils reposent sur la théorie de l'ordonnancement temps réel, et chaque test proposé ne fonctionne que dans un contexte très précis. Utiliser un test d'ordonnançabilité hors de son contexte est incorrect. Les travaux précurseurs de [Ouh13] proposent donc un langage pivot orienté analyse temporelle, et un référentiel d'analyse, permettant d'exprimer exactement dans quel contexte une analyse temporelle peut s'appliquer, sur lequel un outillage peut s'appuyer pour guider l'architecte système dans l'analyse temporelle de son système.

Il existe cependant plusieurs lacunes :

— Les modèles étudiés par les méthodes académiques outillées ne considèrent pas toutes les spécificités des modèles industriels, il faudra donc soit proposer de nouvelles méthodes d'analyse spécifique, soit adapter les modèles industriels aux modèles analysables, le tout en étant certain de ne pas faire une analyse optimiste, du modèle industriel original, c'est-à-dire un résultat plus favorable que la réalité. Toute analyse, puisque l'analyse temporelle fait partie de la sûreté de fonctionnement, se doit d'être conservative, c'est-à-dire qu'elle peut être pessimiste (ce qui mène au sur-dimensionnement), mais qu'elle ne peut en aucun cas se montrer optimiste.

— Il existe de plus en plus de systèmes temps réel répartis, cependant, comme nous le verrons par la suite, la plupart des langages de description d'architecture manquent d'expressivité pour exprimer toutes les propriétés nécessaires à une analyse de temps de traversée des réseaux.

— Il existe de nombreux outils, académiques et commerciaux, d'analyse temporelle, cependant, ceux-ci sont trop limités en terme de modèles d'entrée pour analyser des systèmes complexes distribués, pouvant être constitués de parties très différentes, et pas analysables individuellement avec les mêmes méthodes. Il convient donc de définir une méthode qui permettra d'unifier les efforts de plusieurs outils effectuant des analyses locales, afin d'analyser globalement un système complexe, et ceci de manière conservative.

— Le projet FUI WARUNA se veut agnostique par rapport au domaine. Les artefacts de programmation, et donc de conception, varient de façon importante d'un domaine à un autre. Ainsi par exemple, dans l'automobile, il est fréquent que tous les traitements soient strictement périodiques, alors que dans d'autres domaines favorisant la réactivité, on préférera une programmation événementielle des traitements. De même, certains systèmes peuvent explicitement vouloir rester sur des architectures monocœurs, d'autres sur des systèmes répartis, et/ou multicœurs, voire manycœurs. Alors que le langage pivot que devra définir WARUNA se complexifie, un ingénieur d'un domaine, d'une entreprise, d'un service, ne pourra vraisemblablement n'utiliser qu'un sous-ensemble du langage. Il faudrait donc

des méthodes permettant de contraindre l'ingénieur à n'utiliser qu'un sous-ensemble du langage, consistant avec ses contraintes propres.

1.2 Objectifs et approche suivie

Cette thèse a été financée par le biais du Fond Unique Interministériel du Ministère de l'Économie et des Finances, sur le projet Waruna regroupant Artal, Clearsy, l'Inria, le LIAS, Real-Time-at-Work et Thales Research & Technology. Il s'agit de répondre à plusieurs avancées technologiques dans le domaine des systèmes embarqués temps-réel auxquelles la thèse précédente ne répond pas de façon suffisante.

Elle adresse les lacunes énoncées à la fin de la section 1.1 pour effectuer un transfert des travaux académiques réalisés dans [Ouh13] vers l'industrie. Ainsi, le manque d'expressivité des modèles académiques pour traiter les cas posés dans l'industrie a été soulevée par nos partenaires industriels du consortium. Plutôt que de régler le problème de modélisation posée de façon ad-hoc, à savoir créer une transformation à l'intérieur du langage pivot proposé dans le cadre du projet, nous avons préféré proposé un référentiel de transformation, qui pourra être utilisé quel que soit le langage de modélisation, et qui permettra de regrouper toutes les transformations (ou adaptations) de modèles d'entrée industrielles, ainsi que de vérifier si la transformation est licite (i.e., conservative) pour le cas donnée en entrée.

Le consortium étant appelé à définir un langage pivot orienté analyse, qui est nommé Time4Sys, à l'instar du langage pivot MoSaRT qui avait été défini dans [Ouh13], cette thèse devait apporter des nouvelles méthodes comblant les lacunes existantes entre définition de l'architecture logiciel/matériel et analyse temporelle de celle-ci. Cependant, nous ne pouvions pas faire évoluer unilatéralement Time4Sys, qui nécessite pour évoluer une décision des membres du consortium. Nous pouvions cependant modifier MoSaRT de façon unilatérale. Par conséquent, l'approche suivie consiste, pour chaque contribution, à la proposer de façon indépendante au langage de modélisation sous-jacent, puis à l'intégrer dans MoSaRT, qui devra, parfois, être enrichi. MoSaRT nous sert donc de langage d'expérimentation, il est, de plus, à ce jour, le langage pivot orienté analyse le plus riche sémantiquement, ce qui permettra de représenter très finement les concepts liés à l'analyse temporelle. Lorsque MoSaRT nous permet d'obtenir un prototype, démontrant la faisabilité de la contribution, nous proposons alors de l'intégrer à Time4Sys, ce qui pourra être opéré dans les limites des contraintes négociées avec le consortium.

1.3 Organisation de la thèse

Cette thèse s'organise se compose sept chapitres et s'organise comme suit.

— Le chapitre 2 constitue la base de notre étude. Il décrit les principes généraux de l'ordonnancement temps réel, pour des systèmes monoprocesseurs et des systèmes distribués.

— Le chapitre 3 présente les principes de développement dans l'industrie et plus particulièrement dans l'industrie des systèmes critiques. Ce chapitre présente également la nécessité de certifier temporellement les systèmes critiques. Il passera en revue quelques méthodes et approches existantes dans le domaine des systèmes temps réel pouvant influencer de façon plus ou moins intensive le processus de conception dirigée par les modèles.

— La recherche dans le domaine temps réel s'inspire très fortement des technologies développées. Le chapitre 4 présente des difficultés d'analyse temporelle de cas rencontrés dans l'industrie. Dans ces cas de figure, une adaptation du modèle de conception est nécessaire pour pouvoir l'analyser à condition qu'elle soit conservative. Ce chapitre présentera une approche à base de modèles pour capitaliser ce savoir-faire et automatiser ces adaptations. Cette possibilité est ensuite intégrée dans un processus d'analyse utilisé dans le cadre d'un projet collaboratif.

— Les langages de conception temps réel standards présentent peu d'artéfacts spécifiques à la modélisation des réseaux temps réel. Après les avoir passé en revue et les avoir comparés, le chapitre 5 propose des nouveau concepts réseaux permettant une modélisation incrémentale évoluant au cours de conception. Ces concepts ont été intégrés sous forme d'une extension de méta-modèle.

— Une bonne utilisation des langages de modélisation dédiés nécessite à la fois une bonne maîtrise du domaine et une bonne compréhension des concepts proposés. En effet, ceux-ci doivent être adaptables à toute situation rencontrée dans l'industrie et donnent lieu à de nombreux concepts qui peuvent parfois nuire à la cohérence du langage et génère des ambiguïtés sémantiques. Le principe de réduction de modèle et de réduction du langage sera ainsi traité et présenté dans le chapitre 6. Ce principe sera alors adapté pour les systèmes temps réel avec pour finalité, construire un point de vue basé sur les besoins des architectes.

— Le dernier chapitre de cette thèse présente une conclusion générale en résumant toutes les contributions proposées par rapport aux constats établis. Il présente des perspectives dans le but d'améliorer ou étendre chacune des contributions.

La figure 1.1 suggère un ordre de lecture des sections pour cette thèse selon le chapitre voulu.

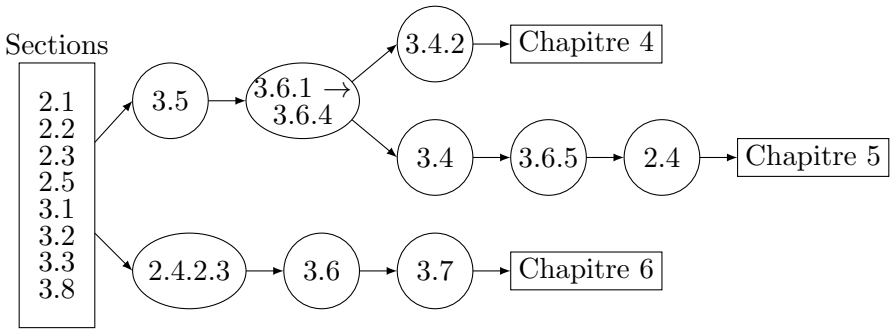


FIGURE 1.1 – Suggestion de lecture des sections de chapitres

1.4 Publications scientifiques

Cette thèse a donné lieu à quatre publications scientifiques :

— Yassine Ouhammou, Anh-Toan Bui Long, Emmanuel Grolleau, Henri Bauer, Frédéric Ridouard, Pascal Richard, Michaël Richard, *A model-based process for the modelling and the analysis of avionic architectures*. International Journal of Intelligent Information and Database Systems, IJIIDS 10(1/2) : 117-144 (2017)

— Anh Toan BUI LONG, Yassine OUHAMMOU, Emmanuel GROLLEAU,*Binding Design and Analysis by Viewpoints for Heterogenous Real-Time systems*, Abstract soumis à Heterogeneous Architectures and Real-Time Systems Seminar (HARTS2017), Université Libre de Bruxelles, 2017

— Anh Toan Bui Long, Yassine Ouhammou, Emmanuel Grolleau, Loïc Fejoz, Laurent Rioux, *Bridging the gap between practical cases and temporal performance analysis : a models repository-based approach*, 25th International Conference of Real-Time Networks and Systems (RTNS 2017), édité par ACM, 2017, pp. 178-187

— Anh-Toan Bui Long, Yassine Ouhammou, Emmanuel Grolleau, *Leveraging Real-Time Network Analyses by Extending a Model-Based Framework*, ACS/IEEE International Conference on Computer Systems and Applications, AICCSA 2017 : 871-878

Première partie

Etat de l'Art

Chapitre 2

Ordonnancement temps-réel

Qui perd patience dans une file d'attente,
vient de doubler son temps d'attente.

— Anonyme

Sommaire

Le domaine des systèmes temps-réel est un domaine très vaste en raison de la variété des systèmes existants. Plus le système est complexe et avancé, moins l'ordonnancement est aisée.

Dans ce chapitre, il sera présenté les principales définitions dans le domaine de l'analyse temps-réel ainsi que l'évolution historique des systèmes temps réels. Pour finir, les réseaux principaux ainsi que l'analyse temps réel de ceux-ci seront présentés et comparés.

2.1 Introduction

Les nouvelles technologies s'impliquent de plus en plus dans des applications critiques tels en aéronautique ou en automobile. La certification de tels systèmes est donc très rigoureuse et est régie par de multiples normes. La section 2.2 définit les termes qui seront utilisés dans ce mémoire pour les systèmes temps réel. Ensuite, la section 2.3 présente l'histoire de l'ordonnancement dans le domaine monoprocesseur et multiprocesseur et comment les systèmes de tâches sont validés temporellement. Ces avancées sont intimement liées aux technologies : les réseaux embarqués nécessitent également une validation temporelle. Les réseaux ainsi que les analyses associées à chaque réseau sont décrites dans la section 2.4.

2.2 Systèmes temps réel

Cette section est dévouée à présenter les principes généraux des systèmes temps réel. Les définitions utilisés par la suite sont également présentées.

2.2.1 Définitions

Un système de contrôle/commande est constitué d'un ensemble d'éléments interagissant entre eux et/ou avec leur environnement pour exécuter certaines fonctionnalités. Ces fonctionnalités sont exécutées par des fonctions informatiques et électroniques. Les fonctions informatiques sont elles-mêmes le plus souvent exécutées dans des tâches logicielles, exécutées par un ou des calculateurs.

De tels systèmes sont utilisés dans la vie de tous les jours tels les robots d'usine, les téléphones portables, les voitures, les avions, les drones, le métro automatique, etc.

Ces systèmes reçoivent de la part des capteurs, sous forme électrique ou électromagnétique, des signaux représentant l'environnement, ainsi que des consignes de l'utilisateur comme présenté sur la figure 2.1. Selon les informations perçues via les capteurs, et les entrées utilisateur, le système de contrôle prend une décision se traduisant par la commande des actionneurs pour agir sur le système et son environnement.

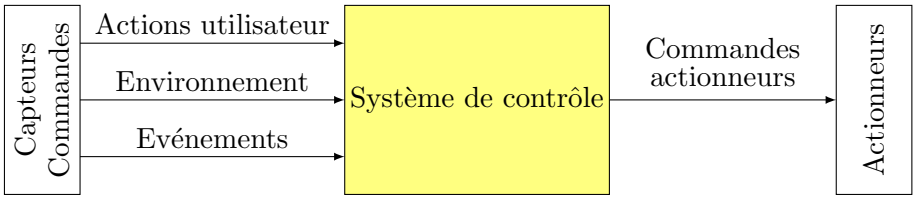


FIGURE 2.1 – Entrées et sorties d'un système temps réel

Un système embarqué est un ensemble logiciel et matériel autonome dédié à accomplir un ensemble de missions spécifiques. Très souvent, les ressources sont limitées sur ce type de système : l'énergie est limitée à des batteries embarquées– potentiellement ré-alimentées en consommant du carburant, embarqué lui-aussi –, l'encombrement – physique ou mémoire – doit être minimal, le coût des systèmes doit être au plus près des besoins, ou au moins rationnel (selon le domaine d'application), etc.

La criticité des systèmes embarqués influe sur le temps de développement des systèmes : plus le système est critique – ou dangereux – , plus la phase de validation durera longtemps. La criticité dépend du risque couru par l'environnement (utilisateur inclus) : la criticité est d'autant plus haute si une simple défaillance peut conduire à des dégâts humains et d'autant plus faible si une défaillance n'occasionne que peu de dégâts. Il existe donc des systèmes critiques comme dans

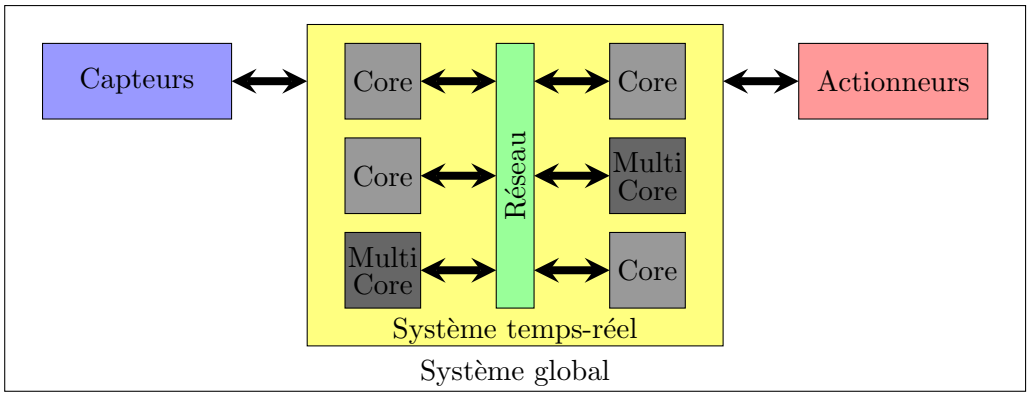


FIGURE 2.2 – Structure interne système

l'aéronautique ou le nucléaire et des systèmes moins critiques comme les téléphones portables ou les micro-contrôleurs Arduino par exemple.

Ce type de système est aussi très souvent soumis à des contraintes de réactivité, réagir du mieux possible pour répondre à la dynamique de l'environnement et aux commandes. Il n'est en général pas admissible que s'écoule un temps important entre la commande et la réaction du système : un temps de réaction trop long du système par rapport à sa dynamique peut engendrer une défaillance. Il est évident, par exemple, que le système chargé de piloter le freinage d'une automobile doit réagir "suffisamment vite" aux sollicitations du conducteur. Cette notion de "suffisamment vite" se traduit par des exigences non fonctionnelles sur les délais, traduites ensuite sous forme d'échéances temporelles sur les tâches informatiques. Le respect des échéances temporelles dans les systèmes embarqués est donc une exigence pour les systèmes critiques afin d'éviter des comportements dangereux.

2.2.2 Généralités

Les systèmes embarqués temps réel sont des systèmes pour lesquels le temps est un élément primordial pour le bon fonctionnement du système.

Prenons l'exemple d'un système automobile *Anti-lock Breaking System* (ABS). Le système électronique ABS évite le blocage des roues en cas de freinage d'urgence. Cela permet un freinage plus efficace et évite la perte de contrôle du véhicule. Le système récupère les éléments de l'environnement comme, entre autres, le taux de rotation des roues, et l'enfoncement de la pédale de freinage. Le système doit déterminer si les roues dérapent pour relâcher les freins jusqu'à reprise d'adhérence des roues. Le temps dans lequel se déroule ces évènements est très court : la réponse des tâches de calcul doit être appropriée. De la même manière, le calculateur du système doit tenir compte d'un éventuel déclenchement des coussins gonflables de sécurité dans un temps raisonnable.

La variété des systèmes temps réel ne permet pas de faire une généralité, cependant, la Figure 2.2 présente un exemple de système embarqué, récupérant des informations des capteurs, et d'après ces informations déterminant les actions à mener. Entre le moment où la commande est donnée et le moment où l'actionneur agit, il s'écoule un certain temps où il faut potentiellement exécuter plusieurs tâches, s'exécutant sur différents calculateurs, transmettre le message sur un réseau, tout en respectant les contraintes de précédence entre les tâches.

2.2.3 Temps réel critique : définition

Il existe plusieurs types de domaines dans le temps réel :

— systèmes temps réel **durs** : où le système n'autorise pas de dépassement d'échéance. Un tel évènement causerait une catastrophe. Cela est représenté par les pilotes automatiques de vol ou la gestion des aiguillages d'un réseau de trains.

— systèmes temps réel **fermes** ou mixte : le système peut autoriser quelques échéances non respectées pour certaines tâches. Une grande fréquence d'échéances non respectées pourrait engendrer une défaillance.

— systèmes temps réel **souple** : le système est résilient aux échéances non respectées mais dégrade les performances de celui-ci. C'est le cas par exemple des systèmes multimédia de diffusion de vidéos ou des téléphones portables.

Dans le domaine des systèmes temps réel durs, il est nécessaire de vérifier les échéances des tâches pour éviter les retards donc les incidents système. La plupart des tâches des systèmes de contrôle/commande ont un comportement récurrent. Les tâches expriment un rythme d'activation des fonctions qu'elles exécutent, et chacune de leur activation sont contraintes par une échéance. Il existe plusieurs types d'activation de tâches :
- les tâches **périodiques** pour lesquelles l'activation arrive tous les T_i unités de temps,
- les tâches **sporadiques** pour lesquelles l'activation arrive au plus tôt T_i unités de temps après la dernière activation,
- les tâches **apériodiques** où l'activation survient de façon non spécifiée.

Comme nous le verrons dans la section traitant de l'histoire de l'ordonnancement, il existe dans la littérature des centaines de modèles de tâches, dont nous donnons ici quelques notions communes.

On formalise dans la suite les notions qui peuvent s'avérer nécessaires dans la suite de ce mémoire. Une tâche périodique ou sporadique τ_i est caractérisée par les éléments suivants :
- C_i : le **temps d'exécution**, souvent représenté par le *Worst Case Execution Time* (WCET), le pire temps d'exécution pour représenter le pire cas du système.
- r_i : la **première date d'activation** lorsqu'elle est connue (ce qui est généralement vrai dans le cas périodique, et faux dans le cas sporadique). Si toutes les tâches sont activées au même instant, les tâches sont dites synchrones,
- T_i : dans le cas sporadique la **période d'activation** minimale entre deux activations successives de la tâche, dans le cas périodique la période exacte d'activation
- D_i : l'**échéance relative**, le délai maximal donné à une tâche pour s'exécuter après son réveil, en d'autres termes, le délai entre réveil d'une tâche et son échéance.
- chaque réveil d'une tâche τ_i donne naissance à un travail, ou *job* $\tau_{i,j}$, dont l'échéance $d_{i,j}$ est égale à sa date d'activation plus son délai critique D_i.

À partir de ces éléments, on peut en définir :
- R_i : le **pire temps de réponse** de la tâche est la valeur maximal du temps de réponse de ses jobs, sur une durée infinie. Si celui-ci est supérieur à D_i, l'échéance n'est pas respectée,
- U_i : l'**utilisation processeur** ou la **charge processeur** (définie par $\frac{C_i}{T_i}$),
- B_i : le **temps de blocage** : le temps pendant lequel la tâche est bloquée par des tâches moins prioritaires à cause d'une ressource critique prise par une autre tâche,
- $L_i(t)$: la laxité dynamique, le temps pendant lequel un job peut être retardé sans dépasser son échéance à un temps t.
La Figure 2.3 présente ces caractéristiques lors d'une exécution d'une tâche périodique.

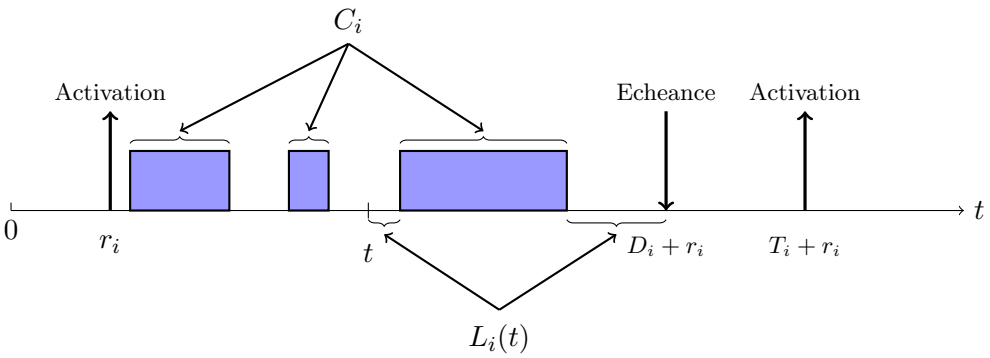


FIGURE 2.3 – Principaux paramètres temps réel

2.3 Histoire de l'ordonnancement processeur

Les définitions de la section précédente sont apparues au cours de l'histoire de l'ordonnancement processeur. Dans la suite, l'évolution de l'analyse temps réel avec le temps depuis les premiers travaux à aujourd'hui est présenté de façon non exhaustive. En effet, les travaux sur le temps réel se sont multipliés et parfois se recoupent.

2.3.1 Ordonnancement monoprocesseur

Les premiers systèmes temps réel multi-tâches sont apparus dans les années 1960 avec le programme Apollo entre autres mais ce n'est qu'en 1973 qu'est apparu la première étude par Liu et Layland [LL73]. À l'époque, les systèmes temps réel dans le programme Apollo, par exemple, fonctionnaient avec de l'ordonnancement hors-ligne. L'ordonnancement hors-ligne consistait à préparer une liste statique d'appel des tâches, liste qui sera appelée lors du fonctionnement en-ligne. Le travail de Liu et Layland fut fondateur de l'ordonnancement en-ligne en proposant pour des tâches périodiques, indépendantes, pré-emptives, sur un processeur avec un instant critique (instant auquel tous les réveils surviennent simultanément), une condition suffisante d'ordonancabilité :

$$U \leq n(2^{1/n} - 1) \xrightarrow[n\to\infty]{} ln(2) \backsimeq .693 \tag{2.1}$$

La condition suffisante d'ordonancabilité 2.1 était valable pour un ordonnancement *Rate Monotonic* (RM) où la plus grande priorité est donnée à la tâche dont la période est la plus petite, ce qui en fait un algorithme optimal pour les conditions énoncées précédemment. Cette condition est dite suffisante : il suffit de vérifier cette condition pour assurer l'ordonnancabilité du système de tâches. En revanche, il est possible que des ensembles de tâches ne respectant pas cette condition soient ordonnançables.

Dans le même temps, l'ordonnancement sous *Earliest Deadline First* (EDF) fut proposé par Liu & Layland [LL73], repris ensuite par Horn [Hor74] et Dertouzos [Der74], proposant ainsi avec des priorités, qualifiées à l'époque de dynamiques, de donner la plus grande priorité à la tâche dont l'échéance est la plus proche. Notons que, depuis, cet algorithme a été requalifié : de dynamique, il est passé à priorités fixes aux travaux, car chaque job, à son réveil, se voit doté d'une priorité fixe. Il est à noter que, comme cela est souvent le cas, cet algorithme avait déjà été étudié sur des jobs en ordonnancement classique sous le nom d'Earliest Due Data [Jac57].

De façon générale, on peut remarquer que le domaine de l'ordonnancement classique qui étudie l'ordonnancement de travaux non récurrents dont on veut minimiser le temps d'execution, ont souvent inspirés l'ordonnancement temps réel où on a des travaux récurrents dont on veut

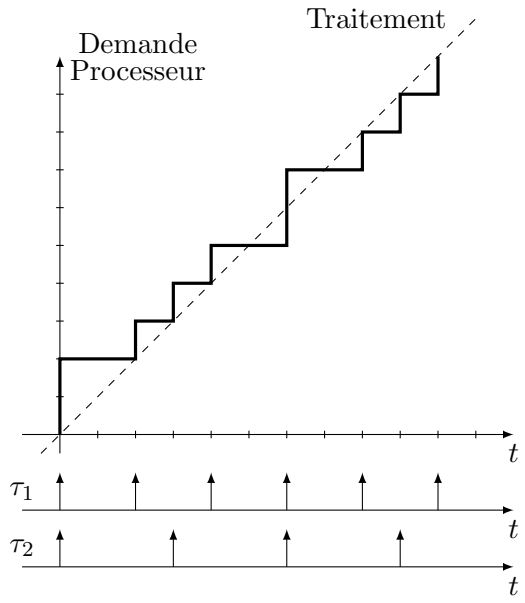


FIGURE 2.4 – Exemple de fonction RBF quelconque

minimiser le retard. EDF permet ainsi d'ordonancer plus de tâches dans un même système car la condition nécessaire et suffisante d'ordonançabilité est [LL73] :

$$U = \sum_i \frac{C_i}{T_i} \leq 1 \tag{2.2}$$

Ceci est valable pour des tâches périodiques [LL73] ou sporadiques [BMR90] indépendantes, à échéances sur requête. Cette approche de l'ordonnancement était inflexible car les systèmes requéraient des tâches indépendantes, ce qui n'est jamais le cas dans les systèmes réels. Néanmoins, cela a conduit à la construction des fonctions *Request Bound Function* (RBF) et *Demand Bound Function* (DBF) qui, respectivement, représentent l'interférence de la demande processeur ainsi que la demande processeur restant à être satisfaite. Ces fonctions se traduisent, par une fonction escalier toujours croissante.

Prenons l'exemple de deux tâches τ_1 et τ_2 activées respectivement toutes les 2 et 3 unités de temps et avec un pire temps d'exécution de 1 unité de temps. L'activation de ces tâches est représentée sur la figure 2.4. Cette figure montre également en trait épais la RBF pour deux tâches τ_1 et τ_2 ainsi que la droite en pointillés représentant le traitement processeur.

Les systèmes temps réel devenant de plus en plus complexes, Leung [LM80] propose d'abord avec Merrill un ordonnancement pour des tâches qui ne sont pas activée par un instant critique. Ensuite, Leung [LW82] propose plus tard l'existence de tâches dont l'échéance peut être plus petite que la période pour certaines applications.

Mok [Mok83] a proposé d'améliorer les algorithmes en mettant en place le principe de priorités dynamiques, *Least Laxity First* (LLF), où la tâche dont la laxité est la plus faible est la plus prioritaire. LLF générant de nombreuses préemptions, il n'a cependant jamais eu d'implémentation industrielle.

Harter [Har84, Har87] propose pour la première fois une méthode exacte de test d'ordonnancement : l'analyse du temps de réponse. Il s'agit de déterminer de façon analytique, mathématique, le temps de réponse en fonction du temps d'exécution des tâches de priorité supérieure. De façon indépendente, Joseph, Pandya et Audsley [JP86, ABRW91] ont également développé des analyses équivalentes résumées dans l'équation 2.3 pour des tâches préemptives. L'équation 2.3

reprend les notations précédentes, et considère une tâche τ_i en supposant que les indices des tâches sont ordonnés suivant les priorités décroissantes (i.e., le plus petit indice correspond à la tâche la plus prioritaire). Cette analyse des temps de réponses fut ensuite améliorée plus tard par Lehoczky en 1990 pour des tâches à échéances arbitraires [Leh90].

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.3}$$

Les temps d'exécutions utilisés dans cette analyse étaient sur-évalués et ne commencent à être évalués plus précisément qu'à partir des études de Kligerman et Stoyenko [KS86] en 1986. Néanmoins c'est en 1989 que Mok *et al.* [MACT89], Puschner et Koza [PK89], et Shaw [Sha89] proposent d'évaluer le WCET et ce pour affiner les analyses des systèmes temps réel.

Sha [SLR86] introduit des tâches apériodiques dans un ensemble de tâches périodiques avec la qualité de service comme critère principal. Les tâches apériodiques deviennent ici des tâches à haute priorité. Par contre, lorsqu'une vague de tâches apériodiques survient, certaines d'entre elles doivent attendre : un algorithme (Deferrable Server) est proposé dans Lehoczky [LSS87, Str88, SLS88, SSL89, SLS95]. Celui-ci donne la plus haute priorité à un maximum de tâches apériodiques tout en garantissant le respect des échéances des tâches récurrentes.

Avec la création des premiers réseaux de terrain dans les années 80, Stankovic et Ramamritham [RS84, SRC85, ZR85, RSZ89] se penchent sur l'ordonnancement dynamique sur les systèmes distribués. Lehoczky et al. [LS86] étudient l'affectation de priorités sur les réseaux distribués. Des réseaux de capteurs commencent à voir le jour : Zhao [ZR87] propose un protocole *Carrier Sense Multiple Access - Collision Detection* (CSMA-CD) pour optimiser ces réseaux et prend en compte l'échéance des messages pour l'ordonnancement d'une application non critique. C'est à partir d'un protocole équivalent, le *Carrier Sense Multiple Access - Collision Avoidance* (CSMA-CA), que le bus *Controler Area Network* (CAN) est inventé à la fin des années 80 [Bos91].

Malgré des systèmes réels utilisant des ressources critiques, ce n'est qu'à la fin des années 80 que l'on s'intéresse aux tâches non indépendantes. C'est alors que Mok [MAC$^+$87] introduit pour la première fois le délai de bout en bout : les tâches ne sont pas indépendantes et c'est une série de tâches liées entre elles par des buffers limités en taille qui peuvent influencer le temps de traitement global – c'est-à-dire depuis l'activation de la tâche jusqu'au résultat final. Le principe de transaction, ou chaîne fonctionnelle, consistant à mettre à la suite les tâches fut une première fois défini dans le système d'exploitation MARS [DRSK89] pour être formellement défini par Tindell quelques années plus tard [Tin94]. Il faut noter qu'auparavant, Harbour *et al.* [HHHKaL91] ont considéré le cas des chaînes de tâches sans nommer le principe de transactions. Dans une transaction, il s'agit, pour une tâche τ_i, de récupérer le temps de réponse de la tâche la précédant et de s'en servir comme gigue ou latence d'activation de τ_i.

Sha [SLR87, SRL90] et auparavant Kaiser [Kai82], décrivent le problème d'inversion de priorités sur un processeur où une tâche de priorité intermédiaire s'exécute avant une tâche de plus haute priorité car une tâche de basse priorité a pris un sémaphore protégeant une ressource critique, attendue par la tâche de haute priorité. Tout se passe comme si, pendant son attente de ressource, la tâche de haute priorité avait la priorité de la tâche de basse priorité détenant la ressource. Ils proposent les protocoles à priorités plafond et à priorités héritées permettant de borner l'interférence des tâches moins prioritaires sur les plus prioritaires. Ce principe est étendu ensuite par Rajkumar [RSL88] pour les systèmes multiprocesseurs. Des variantes ont également été développées par d'autres chercheurs [DM89, Raj90, Mar91, Mue99]. Malgré ces

études, cela n'a pas pu empêcher ce problème d'inversion de priorités de se produire sur le rover Mars Pathfinder lancé en 1997, problème réglé par un patch logiciel.

Pour affiner les analyses sur les systèmes monoprocesseurs, Lehoczky [LSD89] démontra que l'instant critique impliquait pour le cas (et contexte) de RM que l'on ait toujours à chaque instant t :

$$U_i(t) = \frac{\sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil}{t} \leq 1 \tag{2.4}$$

Audsley [Aud91] propose d'ordonnancer de façon optimale, par algorithme d'affectation optimale de priorités ou *Optimal Priority Assignement* (OPA), en évitant de vérifier toutes les affectations possibles. En effet, à partir du moment où le contexte fait que l'ordonnançabilité d'une tâche n'est pas impactée par l'ordre des priorités entre les tâches plus prioritaires entre elles, ou bien par l'ordre des priorités des tâches moins prioritaires entre elles, il suffit de vérifier au plus $\frac{n(n+1)}{2}$ affectations pour n tâches afin de trouver une affectation faisable de priorités si elle existe.

L'ordonnancement hors-ligne est également possible, cela consiste à préparer auparavant une table des tâches avec l'ordre d'exécution. Par exemple, Xu et Parnas [XP90], proposent d'évaluer toutes les combinaisons possibles et de choisir celles possibles.

Pour le réseau CAN, Tindell [THW94, TBW95] calcule les pires temps de réponse dans un réseau CAN, plus tard révisé en 2007 [DBBL07], le calcul original pouvant être optimiste. En parallèle, pour les réseaux embarqués, il développe l'analyse holistique [TC94] consistant a calculer les pire temps de réponses de bout en bout. De façon globale, Garcia étudie l'affectation de priorités pour des systèmes distribués [GH95].

Le principe de tâches sautables pour les tâches faiblement prioritaires fut introduit dans les années 90 [KS95, BB97, CB97] afin d'améliorer le temps de réponse des tâches plus prioritaires.

L'évolution des technologies permet de mettre en place Ethernet dans les années 80 mais ce n'est que dans les années 90 qu'on l'envisage pour des applications temps réel. Par exemple, Lehoczky commence à étudier l'ordonnancement dans les réseaux [Leh98].

Les recherches permettent d'affiner les études sur le temps de réponse : Sjödin et Hansson [SH98] proposent une borne sur le pire temps de réponse pour tester un système de tâches du type de Liu & Layland.

Certains systèmes comme le contrôle des moteurs automobiles nécessitent un ordonnancement spécifique : il faut faire un retour de contrôle comme dans les moteurs automobile où la fréquence de rotation du moteur est variable. Il existe aussi un modèle à périodes et durées variables [LSTS99, SLST99, EHA00, GTPH01].

Néanmoins, ce n'est qu'en 2003 que Bini et al. proposent un rapprochement de la borne supérieure [BBB03] du cas de Liu & Layland démontré comme étant une condition suffisante par Butazzo [But11] :

$$\prod_{i \in 1..n} (U_i + 1) \leq 2 \tag{2.5}$$

Ce test, nommé borne hyperbolique, est non seulement strictement meilleur (i.e., moins pessimiste) que le test de Liu & Layland, mais en plus, les auteurs démontrent qu'aucun test basé uniquement sur la charge, ne peut être meilleur que leur borne hyperbolique.

Dans le même temps, des techniques impliquant des modèles de systèmes temps réel commencent à être mis en place pour l'analyse du comportement temps réel. Par exemple, des

logiciels comme Cheddar [SLNM04] permettent la simulation et l'analyse de temps de réponse de systèmes.

Plus récemment, des études viennent prendre en compte les délais de préemptions lors de l'accès au cache mémoire, *Cache Related Preemption Delay* (CRPD), pour des tâches indépendantes [FW99] et ainsi des algorithmes hors-ligne [Pha16, PRG+18] prennent en compte ce délai permettant ainsi d'affiner les analyses d'ordonnancement, en prenant en compte un WCET moins pessimiste (consistant à prendre en compte l'amélioration apportée par les mémoires cache) tout en payant le délai de rechargement de cache généré par les préemptions.

Pour réduire les coûts dans les systèmes, le principe de criticalité mixte (ou *Mixed criticality*) est introduit sur monoprocesseur [dNLR09]. C'est-à-dire qu'il existe plusieurs criticités sur un même système et qu'il est possible de changer le temps d'exécution ou la période d'activation. Ce modèle est l'un des modèles proposés aujourd'hui pour faire face au problème de calcul de WCET de plus en plus complexe sur les architectures modernes (mémoires cache, pipelines, pré-chargement d'instruction et prédiction des sauts, contention d'accès à la mémoire centrale pour les architectures multicœurs, contention d'accès au bus sur puce (*Network on Chip* (NoC)) dans les systèmes manycœurs, un cas particulier de multicœurs). La criticalité mixte propose une piste consistant à ne pas considérer un WCET réel, qui a tendance à être de plus en plus grand comparé au temps d'exécution moyen, voire au pire temps d'exécution observé, mais à considérer l'occurrence d'un trop grand temps d'exécution comme une faute rare, à laquelle le système devra réagir en se reconfigurant dynamiquement, abandonnant au passage certains jobs de basse criticalité.

Les systèmes de tâches à durées probabilistes représentent une solution pour analyser des architectures embarquées de plus en plus puissantes. En effet, pour certifier certains systèmes comme les systèmes aéronautiques, il est demandé d'éviter les incidents par nombre d'heure de vol. Typiquement, il ne doit pas avoir de défaillance ne doit pas survenir dans, au plus, 10^9 heures de vol selon le niveau de criticalité préconisé par les normes aéronautiques. L'une des premières études probabiliste fut mené par Lehoczky en 1996 [Leh96]. Par la suite, beaucoup d'améliorations ont suivi cette étude : [Gar99, GL99, AB99]. Ces travaux ont proposé des modèles et des analyses de systèmes de tâches à durée d'exécution ou rythme d'activation probabiliste.

Afin de mieux représenter les tâches ainsi que les relations entre elles, le principe de graphes orientés (ou digraphes) permet de généraliser les modèles d'activation de tâches [SEGY11].

Cette section concernait principalement les systèmes monoprocesseurs et les systèmes distribués. Le domaine des systèmes multiprocesseurs est un domaine pour lequel les méthodes d'analyses en monoprocesseur ne peuvent s'appliquer immédiatement. En effet, le fait que les tâches puissent s'exécuter sur plusieurs processeurs rendent l'affectation de tâches plus difficile. Dans la suite, on présentera l'histoire des recherches portant sur les systèmes multiprocesseurs.

2.3.2 Ordonnancement multiprocesseur

Une idée pour s'affranchir de capacités limitées de processeur est de passer au multiprocesseur grâce aux avancées technologiques. Les travaux sur l'ordonnancement multiprocesseur se sont souvent appuyés sur les études menées dans le domaine du monoprocesseur.

À partir des travaux initiaux sur les systèmes monoprocesseurs, il fut donc nécessaire de développer les algorithmes afin d'améliorer et suivre les nouvelles technologies. Dhall et Liu [DL78] proposent une méthode pour déterminer le nombre minimum de processeurs pour un système multiprocesseur. Ces travaux furent fondateurs du temps réel en multi-processeur à l'instar des travaux de Liu & Layland.

Il existe plusieurs types de multiprocesseurs : les multiprocesseurs homogènes apparus dans les années 60 [AHSW62], les multiprocesseurs hétérogènes apparus plus tard pour traiter des problèmes plus complexes dont les premiers sont apparus dans les années 80 [FPE+89] et enfin les multiprocesseurs uniformes dont la structure est identique à celle des multiprocesseurs

homogènes mais dont les capacités de traitement sont différentes Pour affecter les tâches aux processeurs, deux types d'algorithmes d'affectation existent : (1) de façon globale où les tâches peuvent s'executer sur tous les processeurs ou bien (2) de façon partitionnée dans laquelle les tâches (ou les instances de tâches) sont assignées à un processeur. L'affectation optimale des tâches, semblable au problème d'empaquetage (ou en anglais *bin packing*), est connue pour être un problème dont la résolution est non-polynomiale (sauf si P=NP), ou NP-complet [GJ79].

Pour pallier le problème d'affectation, des recherches ont été menées pour trouver des algorithmes d'affectations pour des ordonnancements partionnés à échéance implicite [RSS90, TBW92, HS97]. Par la suite, le principe de synchronisation en monoprocesseur est étendu par Rajkumar [RSL88] pour les systèmes multiprocesseurs. Markatos [Mar91] et Dertouzos et Mok [DM89], entre autres, ont revu les synchronisations avec priorités plafond pour des systèmes multiprocesseurs sans toutefois proposer un algorithme général. Le principe de l'analyse des temps de réponses fut transposé par Andersson et Jonsson [AJ00] mais en donnant une borne supérieure sur le temps de réponse.

Il est possible de trouver des conditions suffisantes d'ordonnancement simples sous Rate Monotonic First-Fit défini par Oh et Baker [OB98]. De multiples conditions suffisantes d'ordonnançabilité furent trouvées dans le cadre de systèmes multiprocesseurs [BF05, BC06, CG07].

Les études pour un multiprocesseur à ordonnancement global n'ont commencé que dans les années 2000. Cela a commencé avec les études de Andersson [ABJ01] pour des tâches à échéances implicites et avec l'algorithme EDF appliqué au multiprocesseur [FGB01]. Le cas de tâches à échéances contraintes ou arbitraires ne fut considéré qu'à partir des travaux de Bertogna [BCL05]. L'algorithme d'affectation optimal OPA d'Audsley [Aud91] fut transposé par Davis plus tard [DB11a]

Ces études ont été rassemblées par Davis et Burns dans [DB11b].

2.3.3 Paramètres de validité d'analyse

La recherche sur les systèmes temps-réel est un domaine très vaste comme l'historique des sections précédentes l'ont montré. Chaque analyse ou recherche dans le domaine s'est accompagnée d'une série d'hypothèses pour lesquelles les analyses sont valides.

Une analyse d'un système de tâches est donc soumise à un certain nombre de contraintes d'application liées au support sur lequel le système est embarqué, ou bien aux caractéristiques des tâches par exemple. Ces contraintes sont très diverses et peuvent être liées à des facteurs aussi divers que le délai critique des tâches par rapport à leur période, la communication entre celles-ci, la préemption des tâches sur les autres, le fait que les tâches se suspendent pour faire des entrées-sorties,etc.

Par la suite, quelques unes de ces différentes contraintes sont présentées.

2.3.3.1 Échéances

Une tâche doit finir de s'exécuter avant son échéance. L'échéance relative à la date d'activation coïncide souvent avec la période, dans ce cas, on dit que l'échéance est **implicite**. L'échéance peut être inférieure ou égale à la période, on parle d'échéance **contrainte**. L'échéance peut être quelconque : l'échéance est alors **arbitraire**.

2.3.3.2 Préemption

Certaines tâches ne peuvent pas être préemptées, d'autres ont des portions de code qui ne peuvent mutuellement se préempter : la section critique. Il arrive que les tâches ne soient pas préemptibles pour d'autres raisons telles l'exécution d'une section critique, ou l'accession à un capteur, par exemple.

2.3.3.3 Temps d'activation

Parfois le système d'activation des tâches est connu et fixé depuis le lancement de l'application. C'est le cas pour les tâches dites périodiques. Si la première date d'activation, et les successives, sont connues, alors les tâches sont dites **concrètes**. Si celles-ci sont inconnues, les tâches sont **non-concrètes**. Inconnues ne veut pas dire non-maitrisées : ainsi, par exemple, en ordonnancement monoprocesseur, lorsque les tâches sont indépendantes et toujours préemptibles, le pire cas se produit lorsqu'un instant critique intervient dans le système de tâches [LL73, Aud91], autrement dit lorsqu'il existe un instant pour lequel toutes les tâches sont activées.

En ce qui concerne les tâches sporadiques, le pire cas se produit lorsque celles-ci se comportent comme des tâches périodiques activées la première fois au même instant. Il est important de souligner ici que les deux hypothèses formulées sont importantes : en effet, cela n'est pas vrai lorsque le système est ordonnancée globalement sur plusieurs processeurs, ni quand les tâches ne sont pas toujours préemptibles, ou lorsque certaines tâches sont soumises à des contraintes, telles que lorsque certaines tâches doivent en précéder d'autres, ou lorsque certaines tâches se suspendent. Ces contraintes exprimées pour énoncer une propriété ("Le pire cas se produit lorsque ...") permettent de définir des "contextes".

2.3.3.4 Durée d'exécution

Afin de conduire des analyses, les durées d'exécution peuvent prendre des formes différentes. Souvent, pour simplifier l'analyse, c'est la pire durée d'exécution qui est considérée. Selon l'analyse choisie, c'est un intervalle de valeurs ou bien des valeurs probabilistes qui seront considérées.

2.3.3.5 Communication et précédence d'exécution

La communication entre les tâches est également un élément important : le cas d'un système de tâches indépendantes simplifie certes l'analyse de systèmes mais ce cas de figure est inexistant dans le domaine industriel. La communication entre les tâches peut s'effectuer de différentes manières. Par exemple, par le biais d'une file d'attente, dans laquelle la première valeur arrivée est la première servie. Les tâches peuvent également communiquer avec une précédence, une tâche doit attendre qu'une autre tâche produise une information pour s'exécuter. Ou encore, une tâche produit une donnée écrasant la dernière donnée, devenue trop obsolète, suivant le principe du tableau noir. Dans ces protocoles, l'analyse sera différente à chaque fois, et impliquera d'autres analyses si des réseaux sont impliqués dans la communication. La section 2.4 présente les particularités associés aux réseaux et systèmes temps réel distribués.

2.3.3.6 Principe de viabilité de l'analyse

L'un des concepts fondamentaux des analyses d'ordonnancement est de façon surprenante, un concept qui a mis quarante ans à être énoncé : la viabilité. Chaque analyse a été développée pour un contexte particulier. Par exemple, l'analyse des temps de réponse, centrale dans l'analyse de l'ordonnancement monoprocesseur à priorités fixes aux tâches, a été proposée pour analyser le pire temps de réponse de tâches périodiques, concrètes, simultanées de durée fixe, indépendantes.

On voit très simplement, sur la figure 2.5 que l'intersection entre RBF et la courbe représentant la puissance processeur permet d'obtenir la date de fin de la tâche étudiée (la moins prioritaire des tâches apparaissant sur le diagramme). Si l'on diminue la durée des tâches, les marches sont moins hautes, l'intersection ne peut que se rapprocher de l'origine des temps. De même, si on augmente une période, on éloigne une marche, à nouveau, il est clair que l'intersection ne peut que se rapprocher de l'origine. Enfin, si on change une date de réveil, on repousse des marches, et là encore, la date de l'intersection entre RBF et puissance processeur ne peut que se rapprocher de l'origine des temps. Cela signifie que l'analyse des temps de réponses est un pire cas pour :

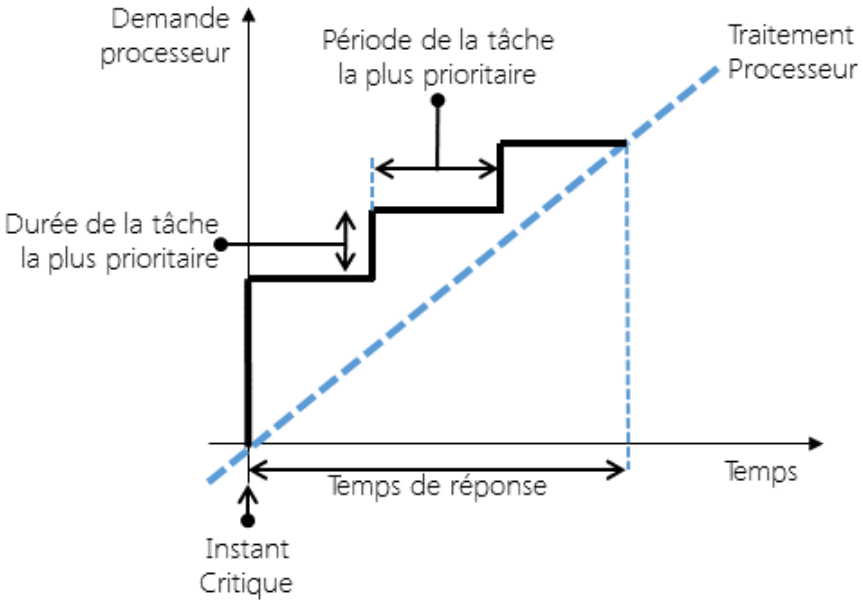


FIGURE 2.5 – RBF sur un processeur et exécution processeur

- les tâches dont la durée d'exécution est inférieure ou égale au WCET
- les tâches sporadiques (i.e. leur période T est le plus petit intervalle inter-activation)
- les tâches différées.

Le test sera alors dit, respectivement, C-viable, T-viable, et r-viable. Comme le temps de réponse est indépendant du délai critique, celui-ci est aussi D-viable.

On parle ainsi d'analyse viable, si, en modifiant les paramètres de telle sorte qu'un système soit moins contraint, le système de tâches reste ordonnancable. Par exemple, lorsque le temps d'exécution d'une tâche diminue, cela ne devrait pas rendre le système non ordonnançable. Auparavant, on parlait beaucoup d'anomalies d'ordonnancement qui peuvent se produire, par exemple, lors de simulations de tâches dépendantes. Il est plus clair de parler de non C-viabilité du test de simulation sur des tâches dépendantes.

En conclusion de cette section, il est important de contextualiser tous les résultats de l'état de l'art utilisés : ceux-ci assument un **contexte**, un ensemble d'hypothèses sur lesquelles les analyses reposent. Il est donc primordial, pour analyser un système avec une méthode, de caractériser le contexte du système, afin de pouvoir choisir l'analyse la plus adaptée.

2.4 Réseaux embarqués

Dans le domaine de l'ordonnancement réparti, les tâches communiquent par le biais de un ou plusieurs réseaux. Ces réseaux s'inspirent de systèmes utilisés actuellement dans la vie courante comme internet ou les bus de terrain. Cependant, nombre de ces systèmes sont basés sur un principe de meilleur effort et non de déterminisme temporel.

Dans cette section, différents types de réseaux compatibles avec la vérification de contraintes temps réel sont présentés, ainsi que quelques analyses disponibles pour ces réseaux.

2.4.1 Introduction aux réseaux embarqués

Malgré une évolution dans la puissance des processeurs, ils restent limités par une certaine puissance de calcul en raison des coûts de production et de fonctionnement. De plus, la répartition des éléments sur de grands systèmes, tels les avions, ne permet pas de mettre facilement en œuvre la centralisation des systèmes sur quelques processeurs. Ainsi, les réseaux facilitent la mise en œuvre de grands systèmes.

Les réseaux sont également devenus nécessaires au fonctionnement de ces systèmes. Pour transmettre des informations, les bus faisaient partie des premiers médium de communication entre processeurs. Dans le domaine du transport automobile, les bus sont un moyen simple pour interconnecter les processeurs.

Dans le domaine aéronautique, les commandes aux actionneurs étaient transmises de façon mécanique : chaque câble de contrôle provenait directement de la cabine de pilotage et circulait sous le plancher de l'avion. Ensuite, la norme ARINC 429 (Aeronautical Radio, INCorporated) [ARI01], publiée dans les années 1970 [Fuc12], permettait la transmission, unidirectionnelle, de consignes ou de valeurs par le biais de câbles directs entre l'émetteur et les récepteurs, permettant de réduire la masse de câbles mécaniques. Des réseaux basés sur le protocole Ethernet et les commutateurs [ARI03] ont été ensuite mis en place dans les avions permettant ainsi de mutualiser les câbles de communication et ainsi augmenter les performances temporelles.

La validation temporelle des réseaux est une méthode requise pour certifier les réseaux dans le domaine aéronautique. Pour ce faire, de multiples analyses ont été mises en place pour faciliter la validation temporelle.

Il est nécessaire de connaître les analyses applicables sur les réseaux pour être capable de modéliser les systèmes distribués de la façon la plus simple mais utilisable pour l'analyse. ainsi par exemple, il n'est vraisemblablement pas utile de spécifier la taille des connecteurs ou la position de vis d'un commutateur en vue d'une analyse de temps de traversée. L'analyse des réseaux se fait de différentes façons selon le réseau utilisé. Pour tous les réseaux, c'est le pire temps de traversée qui est recherché, c'est-à-dire le temps entre l'émission du message par une tâche et la réception par le système d'arrivée. Dans la suite, quelques grands types de réseau embarqués temps réel sont présentés ainsi que les analyses associées.

2.4.2 Bus CAN

2.4.2.1 Présentation

Le bus CAN est l'un des systèmes les plus répandus dans les transports automobiles et aéronautiques. Ce bus, conçu par Bosch pour l'automobile, permet la communication entre plusieurs processeurs ou capteurs [Bos91]. Il fut normalisé sous les normes ISO 11898 [ISO16a] et ISO 11519 [ISO94]. Le bus est utilisé comme médium de communication partagé : tous les composants communicants sont reliés au même bus et l'ordonnancement des messages est géré par le protocole CSMA-CA. Un exemple de réseau est montré sur la figure 2.6.

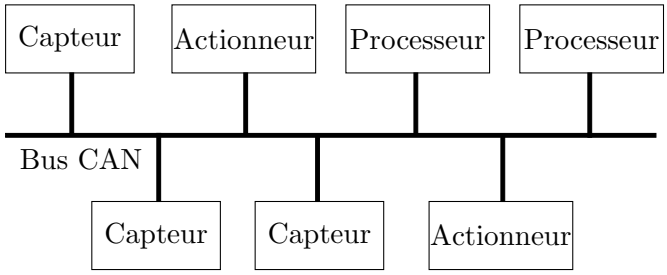


FIGURE 2.6 – Exemple de réseau CAN

Ce protocole se base sur la détection des messages sur le réseau. Chaque bit est émis de façon synchronisée par tous les nœuds du réseau. Lorsqu'une carte réseau en cours d'émission détecte qu'un message de plus haute priorité est envoyé sur le réseau, cette carte arrête d'émettre pour laisser le message de plus haute priorité. Cette distinction de priorité se fait dans l'identifiant du message, plus l'identifiant est petit, plus la trame est prioritaire. Le bus permet aux 0 d'absorber les 1, ce qui fait que lorsqu'un nœud émet un bit à 1 lors de l'émission de l'identifiant de message, il observe le réseau : s'il lit un 0, il en déduit qu'un autre nœud a émis un 0, par conséquent

qu'un message de plus petit identifiant (donc plus prioritaire) est en cours d'émission. On décrit la composition du message dans la figure 2.7.

Une trame CAN comprend :
- SOF : *Start of Frame* ou le bit de début de trame,
- ID : l'identifiant de la trame déterminant la priorité de celle-ci,
- Ctrl : le champ de contrôle, les bits définissant le type de message et le format de la trame,
- Données : la charge utile de la trame,
- CRC : *Cyclic Redunduncy Code*, la somme de contrôle de la trame
- Ack : Acquittement de trame
- EOF : *End Of Frame*, bits de fin de trame

Selon le modèle de réseau CAN, il est possible de transmettre à une vitesse maximale variant entre 125 $kbits/s$ et 1 $Mbits/s$. Cette vitesse relativement lente s'explique par le fait que chaque bit doit avoir le temps de faire une aller-retour sur le réseau avant émission du prochain bit. Plusieurs réseaux CAN peuvent co-exister dans un même système. Par exemple, dans une voiture, il y a entre autres un réseau CAN pour le confort, un autre pour les écrans, et un pour les équipements au cœur du moteur [DBBL07]. C'est un bus de communication encore utilisé et produit à grande échelle dans l'automobile, que l'on trouve aussi dans les cockpits avioniques.

SOF	ID	Crtl	Données	CRC	Ack	EOF
1 bit	12 bits	6 bits	0-64 bits	16 bits	2 bits	7 bits

FIGURE 2.7 – Structure de trame réseau CAN

2.4.2.2 Analyse du temps de réponse dans le réseau CAN

Pour le réseau CAN, le principe de *Response Time Analysis* (RTA) est repris par Tindell [TBW95] et Davis [DBBL07]. Lorsqu'une trame traverse le réseau, un émetteur ne peut pas transmettre son message : la trame est non préemptible. Les analyses de Tindell et Davis utilisent ce principe, en considérant le réseau CAN comme un processeur, ordonnancé de façon non préemptible, sur lequel les trames sont représentées par des tâches de même priorité que les trames, le temps de transmission est représenté par le pire temps d'exécution de la tâche associée. Pour cette analyse [DBBL07], il faut les paramètres suivants :
- La période T_m de chaque message m,
- La priorité de chaque message,
- Le temps de transmission C_m de chaque message m dépendant ainsi de la taille du message,
- La gigue de la file d'attente J_m : le temps correspondant au temps d'attente maximal avant d'être en file d'attente.
- Le temps de transmission d'un bit sur tout le réseau τ_{bit}

Le temps de transmission d'un message m est donné par [TBW95] :

$$R_m = J_m + C_m + w_m \tag{2.6}$$

Avec w_m, le délai de traitement de la file d'attente ou le temps avant que le message ne soit envoyé sur le réseau, déterminé par le point fixe de l'équation suivante [DBBL07] :

$$w_m^{n+1} = \max_{k \in lp(m)}(B_k, C_k) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{2.7}$$

Il est noté $hp(m)$ (resp. $lp(m)$) les indices de messages de plus haute (resp. plus basse) priorité que m, B_m le temps où le message est bloqué par d'autres messages en cours de transmission de

priorité inférieure. Ce test suffisant est considéré pour une transmission parfaite des messages, sans erreur. Il existe des variantes de ce test prenant en compte le fonctionnement interne (gestion du buffer) des cartes d'Entrée-Sortie CAN [KDN11].

2.4.2.3 Analyse holistique

La gigue utilisée dans l'analyse de réseau CAN provient d'une autre analyse, applicable à tout réseau, qui est l'analyse holistique introduite par [TC94, Tin94].

Cette analyse consiste à prendre en compte conjointement l'analyse d'ordonnancement des tâches et celles des messages du réseau. Cela suppose que les tâches sont activées de façon synchrone par l'arrivée des messages : la tâche activée attend le message périodique avant de pouvoir s'exécuter, cependant celui-ci pouvant être retardé, la tâche est soumise à une gigue d'activation. Cette gigue dépend du retard (temps de réponse) qu'a pris la tâche périodique émettrice correspondante à émettre le message, plus le retard qu'a pris le message pour traverser le réseau (temps de traversée). La figure 2.8 représente les valeurs de gigue considérées pour l'analyse holistique, la tâche τ_1 émettant un message reçu par la tâche τ_2. Le message souffre d'une gigue J_m dû au traitement par la tâche τ_1 et la tâche τ_2 d'une gigue composée de la gigue du message et lors du traitement du message, sans compter la gigue due aux tâches localisées sur le processeur.

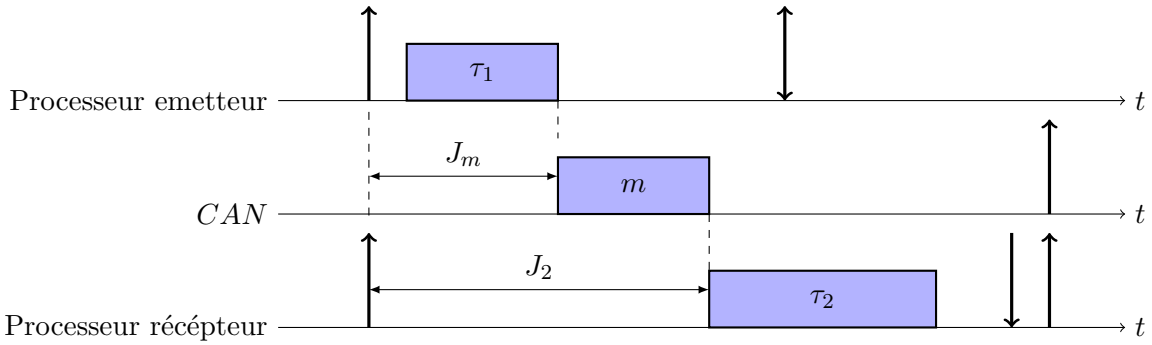


FIGURE 2.8 – Exemple de gigue entre 2 tâches

Au final pour conduire une analyse, en connaissant l'analyse locale d'ordonnancement pour chaque composant du système (processeurs, réseau), on récupère la gigue maximale pour les tâches et les messages. Connaissant ainsi la gigue liée entre les messages et les tâches, il est nécessaire d'actualiser les analyses locales pour déterminer les nouvelles gigues et répéter l'opération. Au final, cela peut aboutir sur un point fixe, i.e. les gigues ainsi que les temps de réponse convergent vers une valeur fixe. Si cela diverge, on ne peut pas conclure. Cela donne une analyse pessimiste [BB98] mais pouvant être utilisée dans le cadre industriel. Le modèle de communication sous-jacent est important : cette analyse n'a de sens que si les tâches sont activées par les messages, mais pas si celles-ci sont activées périodiquement, et considèrent, par exemple, le dernier message reçu, qu'il soit nouveau ou non.

2.4.3 Réseau LIN

Le réseau *Local Interconnect Network* (LIN), ou Réseau local inter-connecté , est également un bus utilisé dans le domaine automobile pour des applications non critiques tels les climatiseurs, fenêtre électrique, chauffage, rétroviseurs électriques, etc... Les bus LIN sont normalisées par la norme ISO 17987 [ISO16b] et peut comprendre jusqu'à 16 nœuds. C'est un bus à diffusion auquel tous les éléments du réseau sont connectés comme CAN. En revanche, la communication est basée sur un nœud maître et des nœuds esclaves : le nœud maître demande au nœud esclave de communiquer, et l'esclave ne peut envoyer de son propre chef des messages. L'ordonnancement des tâches et des messages est donc gérée par le nœud maître : les tâches sur les nœuds

esclaves sont en attente de la demande du noeud maître. L'analyse y est similaire que entre deux processeurs reliés entre eux.

2.4.4 Réseau FlexRay

Le réseau FlexRay est également un bus partagé, normalisé par la norme ISO 17458 [ISO13]. Le réseau peut transmettre des données jusqu'à 10 Mo/s et peut comprendre des flux statiques, composés de messages périodiques, et des flux dynamiques, composés de messages apériodiques. Ensuite, dans chacun de ces flux, les nœuds peuvent chacun leur tour communiquer pendant un certain temps de façon périodique : c'est le principe *Time-Division Multiple Access* (TDMA). Des approches d'analyses d'ordonnancement pour les deux flux de FlexRay ont développées par Pop et al [PPE+08] calculant ainsi le pire temps de réponse par calcul ou par analyse holistique.

2.4.5 Réseau MIL-STD-1553

Dans le même type de communication que le réseau LIN, le protocole MIL-STD-1553 est un protocole utilisé dans l'avionique militaire [DoD78]. Il est basé que le même principe que LIN, un contrôleur (Maître) demande régulièrement à chaque nœud esclave si celui-ci a des informations à transmettre. Le contrôleur interroge tous les nœuds de façon cyclique.

2.4.6 Réseau AFDX

2.4.6.1 Présentation

Le réseau *Avionics Full DupleX* (AFDX), ou ARINC 664 part 7, est un type de réseau basé sur le protocole Ethernet commuté et similaire au réseau ATM, c'est un type de réseau mis en place dans l'aéronautique. Il fut standardisé par la norme ARINC 664 (Aeronautical Radio, INCorporated) [ARI03]. Le réseau repose sur la même infrastructure qu'un réseau Ethernet commuté, comme les câbles réseau et les commutateurs (*switches* en anglais).

La figure 2.9 présente un exemple de réseau AFDX. On appelle terminal ou *End System* (E/S), tout équipement connecté au réseau (en rouge sur la figure) se trouvant à l'origine ou à une terminaison du réseau susceptible d'être émetteur ou récepteur d'un message. Le terminal est ensuite en lien avec le contrôleur pour effectuer l'action liée au messages, l'ensemble s'appelant *Core Processing & I/O Module* (CPIOM). Des commutateurs (ou switches sur la figure) permettent de transmettre entre les terminaux. Selon le matériel, le débit peut atteindre jusqu'à 100 $Mbit/s$ voire 1 $Gbits/s$.

Étant donné que ce protocole est indéterministe, car il implique une résolution des collisions aléatoire, le mode half-duplex n'est pas utilisé dans des applications critiques et notamment en aéronautique.

Les applications utilisent les réseaux AFDX via des liens virtuels, *Virtual Link* (VL). Les VLs sont des canaux virtuels, statiques, unidirectionnel reliant un émetteur et un ou plusieurs destinataires. Un VL permet notamment de remplacer un lien ARINC 429 : une source et une ou des destinations. Cependant, les liens Ethernet et les commutateurs empruntés dans le coeur du réseau sont mutualisés, au contraire de ce qui se passait dans le cas ARINC 429 ou chaque brin était dédié. Un VL se voit réserver une partie de la bande passante du réseau.

On caractérise un VL par :
- ID : l'identifiant de la connexion,
- E/S Emetteur,
- E/S destinataires : les End Systems destinataires avec le chemin pour y accèder depuis le End system émetteur,
- F_{max} : la taille maximale de la charge utile de la trame,
- *Bandwidth Allocation Gap* (BAG) : le temps minimum entre deux trames consécutives émises

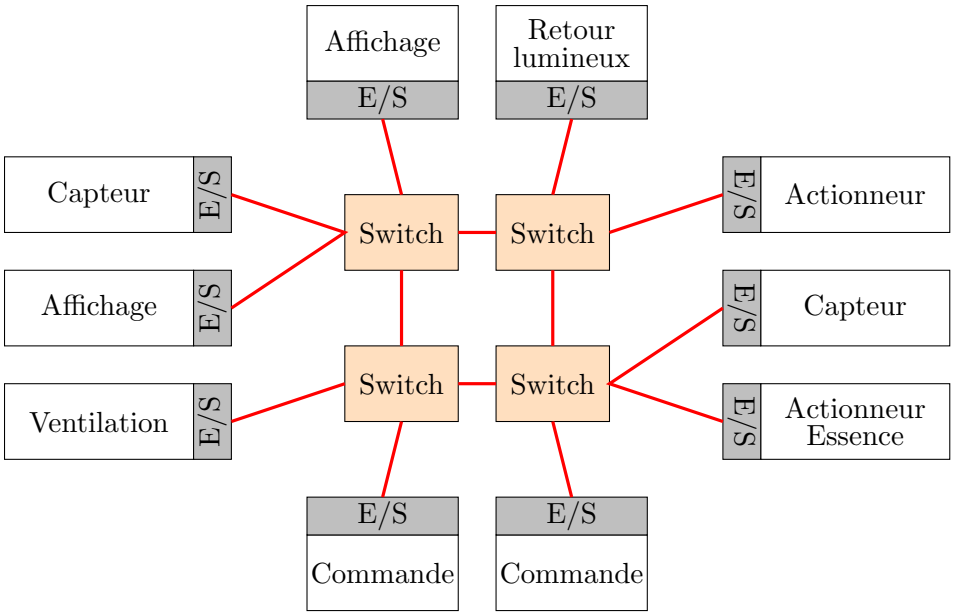


FIGURE 2.9 – Exemple de AFDX

en entrée de ce Virtual Link

Pour le réseau AFDX, il existe plusieurs analyses possibles. Nous n'en détaillerons que quelques unes dans ce qui va suivre.

2.4.6.2 Network Calculus

Cruz a développé l'algèbre Min-plus pour le calcul de temps de transmission sur les réseaux en 1991 [Cru91a, Cru91b]. La méthode calcule une borne supérieure du pire délai de bout en bout sur un réseau commuté. Cette méthode a été adaptée pour les réseaux AFDX par Grieu [Gri04]. C'est, en particulier, la méthode retenue par Airbus pour la certification des réseaux embarqués.

Il s'agit de modéliser les flux par des fonctions affines par morceaux représentant la quantité de données généré par le flux jusqu'au temps t. Ces fonctions sont les courbes d'arrivée. Il faut également calculer, pour chaque port de sortie, la quantité maximale de données qu'il est possible de faire transiter qui est également une fonction affine par morceaux : c'est la courbe de service.

Comme la courbe de service est commune à plusieurs flux, les courbes d'arrivées sont sommées et sont soumises à la capacité du commutateur pour traiter les bits.

Il est nécessaire de connaître les paramètres suivants, pour chaque VL :
- Le BAG : le temps entre deux trames,
- La taille maximale d'une trame,
- La vitesse du réseau,
- Le cheminement des VLs.

Cette méthode permet d'obtenir des résultats conservatifs mais propose encore des valeurs trop grandes par rapport aux valeurs réelles, et ne tolère pas une charge globale instantanée supérieure à 1.

2.4.6.3 Méthode des trajectoires et *Forward Analysis*

Kemayo et al. ont étudié une autre méthode moins pessimiste que le Network Calculus, pour les réseaux AFDX : la méthode des trajectoires [KRBR14]. La principale différence dans la modélisation réside dans les flux : un flux ne concerne qu'un seul destinataire. Donc, un VL comprenant plusieurs flux sera divisé en autant de flux.

La technique consiste à considérer les collisions possibles sur les nœuds (ou *switches*) depuis le destinataire vers l'émetteur. Les collisions pouvant interférer sont prises en compte dans le calcul de pire temps de transmission.

Les mêmes paramètres que dans la méthode du *Network Calculus* sont requis. C'est une méthode qui peut présenter des résultats optimistes [KRBR14] rendant cette méthode peu recommandable pour des applications critiques devant être certifiées.

Afin d'obtenir des résultats conservatifs, la méthode du Forward Analysis a été proposée [KBR+15]. Le principe du *Forward Analysis* (par opposition à *Backwards*) est d'analyser les flux depuis l'émetteur vers le destinataire – contrairement à l'analyse précédente. Cette méthode propose des résultats conservatifs pouvant être moins pessimistes que le *Network calculus* selon le système AFDX utilisé. Les deux méthodes sont incomparables et on trouve des instances validées par l'une mais pas par l'autre ce qui fait qu'on pourrait conserver les résultats d'analyse les moins pessimistes des deux méthodes.

2.4.7 Réseau ATM

Le réseau *Asynchronous Transfert Mode* (ATM) est une technologie conçue pour la transmission de la voix et vidéo pour de la télécommunication [ITU99]. Il peut être utilisé dans le cadre de réseaux privés ou publics. Grâce à ce protocole, la vitesse de transfert peut atteindre plusieurs gigabits par secondes. Il fut normalisé dans les télécommunications en 1991.

Le réseau ATM est basé sur des commutateurs et sur l'asynchronisme des communications. Les commutateurs (ou switch) permettent la centralisation des flux et ainsi réduire le nombre de câbles.

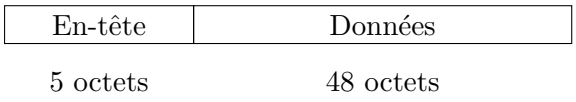


FIGURE 2.10 – Structure de base d'une trame ATM

Les messages transmis sont basés sur une structure de base figurant sur la figure 2.10. L'entête d'une trame permet de décrire l'identifiant de trame, le type de données, la priorité de la trame, etc. Les sorties des commutateurs contiennent des files d'attentes en *First In, First Out* (FIFO) à priorités fixes (ou Premier arrivé, premier servi) : les files d'attentes sont classées par priorité. Pour chaque sortie de commutateur, la file d'attente de plus haute priorité est utilisée en premier et une fois la file d'attente vidée, la file d'attente de priorité inférieure est utilisée. Pour chaque file d'attente, les messages sont envoyées par paquets de façon régulière avec un *cell spacer*. Plusieurs paquets peuvent être requis pour un message, s'il est plus long que 48 octets.

Un ensemble de files d'attentes sont utilisées pour chaque connexion de sortie. La figure 2.11 représente la structure des files d'attentes utilisés par les commutateurs.

Pour résumer, un message est affecté d'une priorité et sera classé avec cette priorité jusqu'à réception du message. Cette priorité définira la file d'attente affecté au message dans chaque commutateur.

Les connexions entre les composants à travers le réseau ATM se font avec des chemins virtuels (ou *Virtual Path*). Le chemin – la succession des commutateurs utilisés – peut être soit

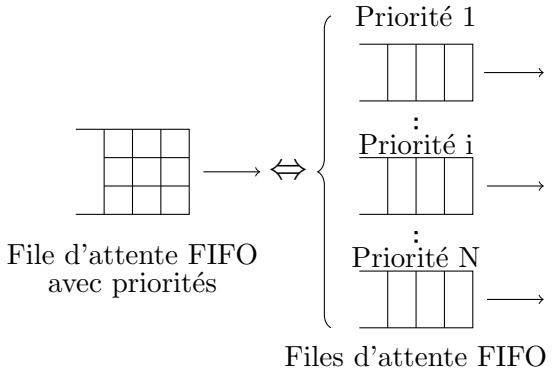


FIGURE 2.11 – Structure des files d'attentes à priorités sur ATM

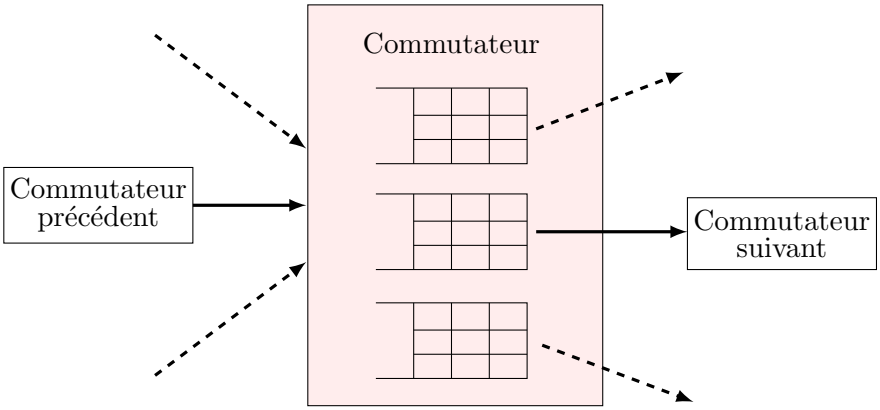


FIGURE 2.12 – Exemple de Switch ATM

dynamique pour éviter un composant défaillant ou soit statique. La figure 2.12 représente un commutateur avec ses files d'attentes ainsi que le chemin suivi par un message.

Des études furent menées car ce réseau fut présenti pour être utilisé dans les applications critiques [EHS97] mais l'industrie aéronautique a préféré s'orienter vers les réseaux de type AFDX, s'inspirant du réseau ATM, mais utilisant des technologies Ethernet. Le protocole est néanmoins utilisé dans les applications de télécommunications.

2.4.7.1 Analyse du temps de réponse dans ATM

Pour le réseau ATM, on considère que les chemins sont fixés afin de simplifier le problème [GRR11]. Chaque chemin possède une priorité et les switches font office d'un espace de collision dans le réseau.

Les paramètres nécessaires sont :
- Le nombre maximal de paquets par message N_i
- La période T_i entre deux messages,
- La gigue J_i possible pour un message,
- La période t_i entre deux paquets,
- La gigue réseau j_i augmentant avec la longueur du chemin

Le pire temps de transmission d'un flux i traversant m switches est dépendant de :
- du temps passé dans le processeur s_0 de départ :
  - du temps dans la file d'attente L_i,
  - du temps pour tous les paquets soient envoyés depuis $Q_i(s_0)$,
- du temps de transmission entre deux nœuds C_0,
- du temps de traitement $Q_i(s_j)$ dans un switch s_j de transition

- du temps de transmission depuis ce switch C_{s_j},

On obtient de façon similaire au réseau CAN une expression permettant de déterminer le temps de transmission.

$$R(flux_i) = L_i + Q_i(s_0) + C_0 + \sum_{p=1..m} (Q_i(s_p) + C_{s_p}) \tag{2.8}$$

On ne détaillera pas les calculs, les résultats et démonstrations se trouvant dans [GRR11].

	CAN	**LIN**	**FlexRay**	**MIL – STD 1553**
Vitesse de communication	1 Mo/s (CAN 2.0 B)	~ 2.5 Ko/s	10 Mo/s	1 Mbps
Norme internationale	ISO 11898 [ISO16a] et ISO 11519 [ISO94]	ISO 17987 [ISO16b]	ISO 17458[ISO13]	MIL-STD 1553
Utilisation	Transports terrestres et Aéronautique	Automobile	Automobile	Aéronautique militaire
Ordonnancement temps-réel	Réseau assimilable à un monoprocesseur à priorités	Communication Maître / Esclave	TDMA	Attente active (*Polling*)
Analyse temps-réel	[DBBL07] [TC94, Tin94]	Analyse transactionnelle [GGH97]	[SS09, Lam77]	[Ray87]

	AFDX	**ATM**
Vitesse de communication	~ 12,5 Mo/s (100 Mbps)	3-100 Mo/s
Norme internationale	Arinc 664 - part 7 [ARI03]	Norme ITU 190 [ITU99]
Utilisation	Aéronautique	Télécommunications
Ordonnancement temps-réel	FIFO des Virtual Link	FIFO à priorités
Analyse temps-réel	Network Calculus [Gri04], Méthode des trajectoires, Forward Analysis [KRBR14]	Analyse du temps de réponse [GRR11]

TABLEAU 2.1 – Comparaison des réseaux

2.5 Conclusion

Les modèles de tâches temps réel ainsi que les modèles d'architectures sous-jacentes ont évolué avec des concepts clés de l'ordonnancement, tels la viabilité, concept émis en 2006 par Baruah et Burns [BB06], avec l'évolution technologique. La figure 2.13 présente un résumé des principales évolutions dans la recherche du temps réel. Souvent, les analyses temps réel sont en retard sur les technologies. Par exemple, les premiers processeurs sont apparus en 1958 [Wil92] et les premières analyses sont proposées par Liu & Layland [LL73]. La mémoire cache fut pour la première fois nommée en 1968 [Lip68] et les premières études sur les CRPD eurent lieu en 1999 [FW99].

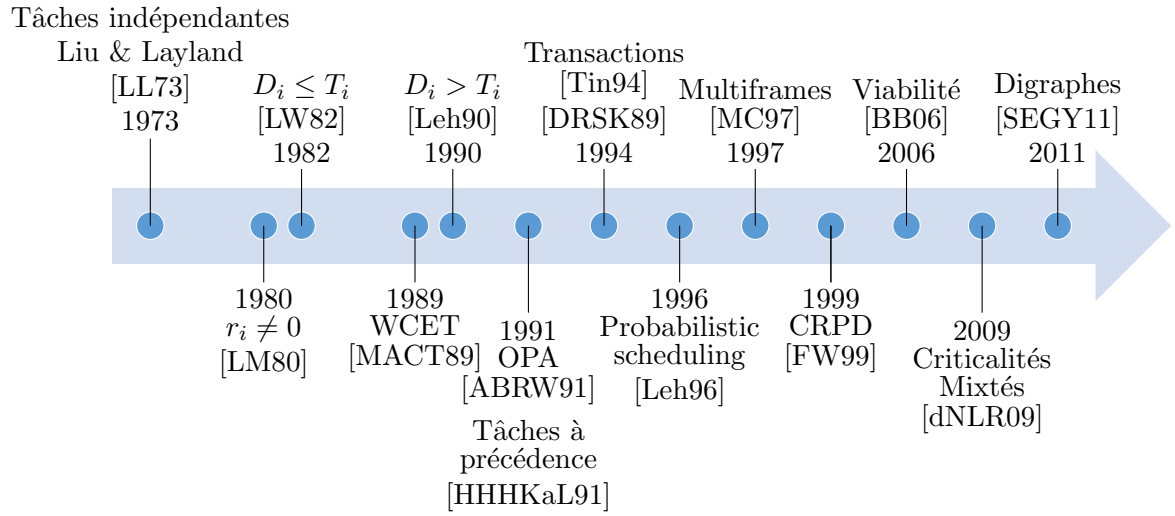


FIGURE 2.13 – Une chronologie des analyses temps réel

Le domaine du temps réel n'a fait que s'adapter aux nouvelles technologies, aux systèmes distribués, au multi-coeurs, et *many-core*. Les réseaux présentent des caractéristiques différentes et des analyses différentes. Les analyses et les algorithmes d'ordonnancement sont résumés dans le tableau 2.1.

La complexité croissante des systèmes embarqués implique une complexité des analyses temps réel croissante. L'évolution de la technologie a permis des modifications profondes dans les systèmes. En effet, cela a permis, par exemple, la réduction de poids dans les avions civils grâce à la technologie AFDX.

Les systèmes temps réel et la recherche associée deviennent de plus en plus complexes. La complexité des analyses implique également le besoin en validation d'une expertise avancée dans le domaine du temps réel. Cette expertise ne vient pas sans coût : il est nécessaire de prendre du temps pour former l'expert analyste et du temps pour l'analyse en elle-même.

Dans le domaine industriel, les contraintes temporelles de développement sont très fortes et les systèmes temps réel prennent une place de plus en plus centrale dans un système aéronautique par exemple. La réduction de temps de certification par rapport aux contraintes non fonctionnelles du système tout en les garantissant n'est que bénéfique pour l'industriel. L'idée de rapprocher la validation d'un système de la conception et ainsi construire un système valide par construction offre de grandes possibilités de réduire le temps et les coûts de conception, réduisant ainsi les risques d'erreur de conception. Dans le domaine temporel, un moyen pour y arriver serait de rapprocher la validation temporelle de l'architecture système.

L'ingénierie dirigée par les modèles présente des avantages permettant de manipuler les systèmes devenus de plus en plus complexes. Dans la suite de ce mémoire, on se propose de

mettre en place une méthode à l'aide de l'ingénierie dirigée par les modèles pour permettre l'analyse des systèmes complexes tels les systèmes distribués ou les systèmes requérant une expertise avancée en temps réel, domaine devenu vaste comme montré dans ce chapitre.

Chapitre 3

Conception dirigée par les modèles des systèmes temps réel

Le temps, c'est de l'argent.

— Benjamin Franklin

Sommaire

Un système temps réel n'est souvent pas le fruit d'un travail de quelques jours, en particulier pour les grands systèmes. Lorsqu'un projet est mis en place, et plus particulièrement pour des systèmes critiques, un très long processus commence, de plusieurs mois pour un satellite, à plusieurs années pour un aéronef civil, depuis la conception jusqu'à la validation du système et

l'utilisation par le client. Ce chapitre est dévoué aux principes de développement des systèmes. Le développement des systèmes critiques prend ainsi beaucoup de temps et des paradigmes pour réduire le temps et la complexité de mise en œuvre existent pour différents stades du développement. On développe dans la suite le principe de l'ingénierie dirigée par les modèles. Nous présenterons ce paradigme ainsi que les outils utilisés pour le développement des systèmes temps réel.

3.1 Introduction

La complexité des systèmes temps réel critiques fait que l'analyse temporelle devient également tout aussi complexe. Le paradigme de l'ingénierie dirigée par les modèles facilite l'interprétation de l'analyse temporelle au moment de la conception. Par conséquent, le temps de mise sur le marché ainsi que la complexité de mise en œuvre se retrouvent réduits. L'application de l'ingénierie dirigée par les modèles dans le cadre des systèmes temps réel critiques tel en aéronautique ou en automobile est donc devenu essentielle pour réduire les coûts.

D'abord, la section 3.2 présente succinctement les cycles de développement logiciel classiques. Ces cycles de développement peuvent privilégier la sûreté logicielle, ou bien minimiser le temps nécessaire à l'élaboration d'un prototype, ces deux objectifs semblant antinomiques. Le cycle de vie choisi dépend alors grandement du type d'application visée : un système critique se doit d'être développé en privilégiant la correction et la sûreté de fonctionnement, en partant d'exigences énoncées complètement, alors qu'un prototype de laboratoire par exemple pourra voir ses exigences évoluer au fil du développement, et privilégier un cycle de vie permettant l'élaboration rapide de prototype voué à évoluer dans ces cycles de vie rapide.

Le développement de logiciels complexes et sûrs utilise des modèles intermédiaires, sur lesquels le concepteur pourra raisonner, et plus tard faire des preuves. L'ingénierie dirigée par les modèles se propose de réduire le temps de développement en établissant un ensemble de modèles valables tout au long du développement, si possible traçables, et permettant à un modèle de servir de base à la génération en partie automatique du modèle suivant. Ces modèles peuvent être vus comme un unique modèle, exprimé avec plusieurs langages en fonction de la phase de conception considérée. Ce modèle est utilisé depuis les exigences du système jusqu'à la livraison. Ce principe est présenté dans la section 3.3. Le principe de l'ingénierie dirigée par les modèles est ensuite appliqué au domaine du temps réel. La section 3.4 présente les langages dédiés aux systèmes temps réel pour l'ingénierie dirigée par les modèles. La section 3.5 présente les modèles de tâches temps réel ainsi que les outils utilisés pour l'analyse temporelle implémentant les travaux de recherches que l'on a décrits dans le chapitre 2 ainsi que les travaux liés à l'analyse dirigée par les modèles. Enfin, la section 3.6 présente la structure du framework MoSaRT, langage orienté analyses permettant le lien entre architecture logicielle et analyse temporelle.

3.2 Cycles de développement d'un système critique

Cette section présente les principales étapes pour établir, construire, et valider un système en particulier dans les systèmes critiques. Ce processus, d'une durée variable selon la taille du projet, est obligatoire pour s'assurer du bon fonctionnement du logiciel. Les cycles de développement permettent de faciliter la mise en place des processus de développement des grands projets en équipe.

Dans les années 70, les logiciels étaient développés pour un usage interne avec des besoins internes, mais rapidement le besoin d'un cycle de développement se fait ressentir : les produits sont terminés en retard, coûtent plus chers que prévu, sont peu fiables et sont difficiles à entretenir [Som06]. Pour les systèmes embarqués du domaine des transports, la qualité du logiciel est très importante pour réduire les risques d'incidents logiciel.

3.2.1 Normes logicielles d'un système critique

Des normes sont apparues pour harmoniser le développement des systèmes critiques :

La norme ISO 26262 [ISO11] est une norme mise en place en 2011 pour les systèmes embarqués automobiles pour le développement depuis la phase de projet jusqu'à la maintenance du système afin d'assurer la sûreté du système. La norme définit un cadre et les processus à utiliser durant le développement ainsi que la conduite à tenir selon le niveau de criticité. Cela permet de prendre en compte la sûreté du logiciel durant le cycle de vie. Pour aider l'application de cette

norme, le consortium AUTOSAR [HSF+04] s'est donné pour but de standardiser les systèmes embarqués automobiles par le biais de spécifications génériques à destination des constructeurs et sous traitants automobiles. Cela permet d'améliorer les relations entre les différents acteurs et au final d'accélérer le développement et la maintenance des systèmes.

Les normes européennes EN 50126, EN 50128, EN 50129 [Eur00a, Eur00b, Eur00c] définissent, à l'instar de la norme ISO 26262, les processus de développement pour les systèmes ferroviaires et comment le développement logiciel doit être effectué.

La norme DO-178 / ED-12 [EUR11, RTC12] est une norme aéronautique préconisant la méthode de développement logiciel à appliquer sans toutefois expliciter un cycle de développement à utiliser. En revanche, la norme prescrit des objectifs ou des moyens que les processus de développement doivent respecter tels l'analyse de couverture fonctionnelle.

Malgré leurs domaines différents, toutes ces normes définissent néanmoins des étapes communes :

— une phase de **définition des besoins** ainsi que de la faisabilité du système demandé où le cahier des charges fonctionnel du système est établi. Cela va constituer la partie que l'utilisateur pourra voir et utiliser. Il s'agit ici de déterminer les exigences du système sous toutes ses coutures : logiciel, mécanique, matériel, fonctionnel, par exemple. Dans le cadre des systèmes temps réel, il s'agit aussi de définir des requis non-fonctionnels.

— une phase de **conception détaillée**. Il s'agit de mettre en place la structure interne du système, c'est-à-dire la relation entre les composants du système et les tâches tout en respectant la phase de définition des besoins.

— une phase de **développement** où les fonctions sont codées selon la spécification avec le langage approprié et adapté qui a été défini lors de la spécification.

— une phase de **validation et vérification logicielle** dont le but est de vérifier le bon fonctionnement du logiciel.

3.2.2 Approche ARCADIA

Pour appliquer ce développement aux systèmes critiques, l'approche ARCADIA (ARChitecture Analysis and Design Integrated Approach) [Voi18], fut envisagée en prenant en compte d'autres langages comme AADL [Fei04], ou SysML [OMG17a]. Cette approche part du principe que pour définir entièrement un système, tous les acteurs doivent pouvoir le spécifier. Le système est défini de façon large au début et la spécification est précisée au fur et à mesure par approche descendante. Une fois les exigences du système définies, chaque composant est alloué avant de vérifier des exigences. Aux débuts de cette méthode, celle-ci se révéla trop compliquée lors du passage à l'échelle et trop rigide pour correspondre aux cycles de développement utilisés dans l'industrie. De façon similaire à la première expérimentation, l'outil était très générique et laissait ainsi peu de place pour les concepts spécifiques.

Une évolution de cette approche, inspirée des méthodes Agile, base la nouvelle approche sur le cahier des charges global. Cela développe une pratique adaptée au domaine avec un langage dédié à ce domaine impliquant ainsi les experts de chaque domaine dans la conception du système. La méthode ARCADIA permet de couvrir toutes les activités essentielles du développement, depuis la spécification jusqu'à la validation. Elle permet donc la coopération entre les ingénieurs des différents domaines via des points de vue pour développer les systèmes mais également diriger les moyens mis en œuvre pour déployer le système. Le modèle est donc commun à tous les domaines.

Pour appliquer cette approche, un outil de modélisation a été créé, l'outil Capella, basé sur la méthode Arcadia dont la version finale est publiée en 2015. L'outil permet de modéliser les différentes couches logicielles d'un système soit dans l'ordre : la couche opérationnelle, la couche des besoins fonctionnels, la couche logique, la couche physique et enfin la couche spécifique à chaque domaine (temps-réel, électrique par exemple)).

Ce principe consiste à modéliser au fur et à mesure le système depuis les besoins jusqu'à la spécification détaillée des fonctions et des tâches. La modélisation d'une couche est faite, en s'appuyant sur la modélisation précédente.

Ce type de modélisation est particulièrement adaptée à un cycle en V, et aux systèmes critiques. La figure 3.1 présente le principe général du cycle en V. Ce cycle est constitué de deux grandes parties :

— Une première pente descendante concernant principalement la spécification du système (phases bleues sur la figure 3.1),

— Une seconde pente montante désigne la série de tests à appliquer (phase oranges sur la figure 3.1).

La jonction entre les deux correspond au codage du système. La série de tests appliquée doit correspondre à la spécification définie dans la pente descendante : plus le test est haut, plus la partie du logiciel qui sera testée sera grande.

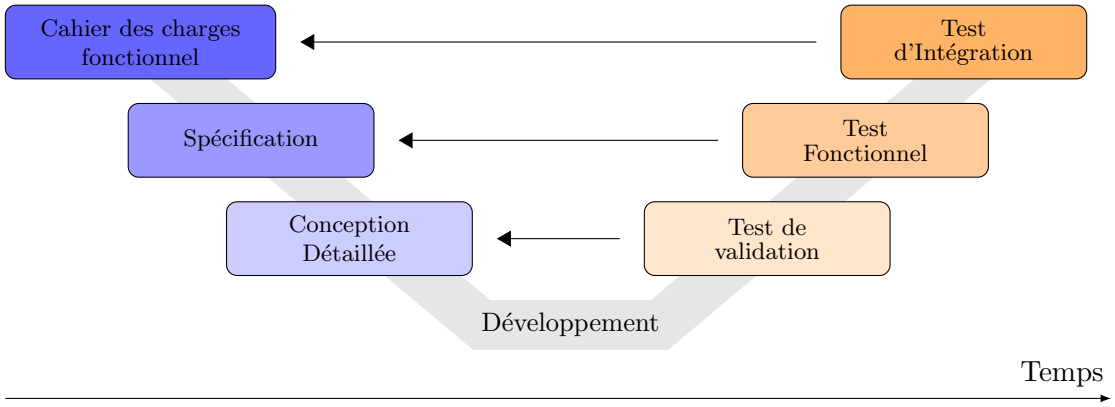


FIGURE 3.1 – Cycle de développement en V

Ce type de cycle développement est l'un des plus répandus et l'un des plus sûrs en terme de développement dans le domaine logiciel normalisé par la norme ISO 15288 : 2015 [ISO15] et compatible avec les normes de développement aéronautiques DO-178C [RTC12] (ou ED-12C [EUR11] en Europe) et MIL-STD498 [US 94]. C'est un cycle de développement répandu dans les systèmes critiques car il permet de spécifier le cahier des charges global dès le début du projet, contrairement à des applications mobiles par exemple dont les besoins sont évolutifs. D'autres cycles de développement existent mais celui-ci permet d'assurer un système correct par construction. Cela permet non seulement une meilleure qualité de code mais aussi une réduction des risques d'erreurs logicielles.

Un inconvénient de ce processus de développement est la rigidité de celui-ci. Effectivement, si une erreur est découverte lors des tests de validation, c'est-à-dire proche de la mise en service du système, il faut remonter jusqu'à l'origine de l'erreur qui peut se situer au moment du développement voire même lors de la spécification du cahier des charges impliquant alors de refaire tout le processus après correction. De plus, en cas de changement au niveau d'une exigence, le temps de développement peut être impacté de façon importante. Dans le domaine du temps-réel, c'est lors de la conception détaillée que l'architecte système décide du placement des fonctions dans les tâches et des tâches sur les calculateurs. Or, la validation temporelle d'un système se fera quasiment en bout de développement, lors des tests de validation. Cependant, comme nous l'avons vu dans le chapitre précédant, le choix des paramètres des tâches et des messages est primordial dans la validation temporelle. Ces choix sont opérés par l'architecte système lors de la conception détaillée sont donc primordiaux pour la validation qui aura lieu bien plus tard. Une mauvaise conception de l'architecture système est donc détectée très tardivement et aura donc un coût important sur le temps de développement.

Étape de développement	Erreur logicielle Coût estimé	Erreur système Coût estimé
Spécification	1	1
Conception détaillée	5-7	3-8
Développement	10-26	7-16
Tests d'Intégration/Fonctionnel	50-177	21-78
Opérations	100-1000	29-1615

TABLEAU 3.1 – Coûts financiers estimés d'une erreur [HSD+04]

Haskins et al. [HSD+04] a mené une étude sur les coûts de l'erreur et selon la méthode de calcul utilisée, une erreur logicielle peut représenter entre 1 fois à 1000 fois le coût de la spécification. Le tableau 3.1 présente les coûts financiers estimés pour une erreur pour différentes étapes de développement.

3.2.3 Conclusion

Les cycles de développement sont un moyen efficace pour le développement des systèmes embarqués. Par contre, suivre un cycle de développement rigide implique de mettre en place des méthodes rigoureuses, en particulier pour éviter les coûts conséquents résultant d'une erreur dans le système. En effet, faire ces étapes sans aide logicielle relève d'une tâche ardue : les erreurs sont aisées et comme on a vu précédemment, une erreur peut coûter cher. Éviter les erreurs prend beaucoup de temps, et vérifier l'absence d'erreurs aussi. Il faut donc des méthodologies qui permettent de réduire le temps de développement. Dans la suite, on présente l'ingénierie dirigée par les modèles, une méthode permettant de réduire les temps de développement et ainsi réduire les coûts financiers. En effet, l'ingénierie dirigée par les modèles permet de traverser les étapes de développement plus facilement dans la modélisation et permet aussi de se référer au modèle lors de la validation du système.

3.3 Ingénierie dirigée par les modèles

Le paradigme de l'Ingénierie Dirigée par les Modèles (IDM) [Sch06] est un processus génératif permettant de passer d'une étape de conception à une autre depuis la spécification à l'exécution du système en mettant les modèles au coeur de ce processus. Une variation de l'IDM a été normalisée sous le nom de Model-Driven Architecture (MDA) par l'*Object Management Group* (OMG) [OMG00]. La problématique principale auquel répond l'IDM est comment développer des systèmes de plus en plus complexes tout en garantissant la satisfaction des besoins. Une des premières expérimentations dans ce domaine remonte à 2001 [EN04] : MDSysE (*Model-Driven SYStem Engineering*). Cela permettait de mettre en œuvre la documentation des systèmes, et envisager ainsi une analyse du modèle. Cette expérimentation utilisait le langage UML publié dans sa version définitive en 2003 [OMG15] mais le langage UML manquait de concepts spécifiques : les domaines non liés directement à l'implémentation devaient être encore adressés de façon classique.

3.3.1 Modélisation et méta-modélisation

Les modèles permettent de représenter les systèmes informatiques et ainsi permettent de mieux communiquer entre les parties prenantes, de spécifier les fonctions logicielles, de mieux visualiser le code. Une méthode se basant sur ce modèle, telle la génération automatique de code par exemple, permet d'assurer que le code est conforme à la spécification. Dans cette section, nous présentons l'élément clé de l'ingénierie dirigée par les modèles à savoir la modélisation.

3.3.1.1 Définitions

Les modèles dans le génie logiciel ont de multiples utilisations. Ils servent à exprimer les fonctionnalités, les exigences du système, à étudier différentes possibilités de conception du système, etc. Les informations contenus dans les modèles, au lieu d'être cantonnées à leur fonction initiale, peuvent servir de base pour engendrer le système réel.

Les utilisations des modèles sont multiples : génération de code, construction de l'électronique, câblage d'un système, etc. Il parait nécessaire de définir le vocabulaire de l'ingénierie dirigée par les modèles. Un système est **représenté par** un modèle selon le domaine d'application. Dans un modèle, un objet est **conforme** à un langage de modélisation lorsque cet objet est dépendant de celui-ci. Pour un modèle, on parle de méta-modèle du langage. Le méta-modèle représente toutes les règles que doit respecter le modèle pour que ce dernier soit **conforme** au méta-modèle.

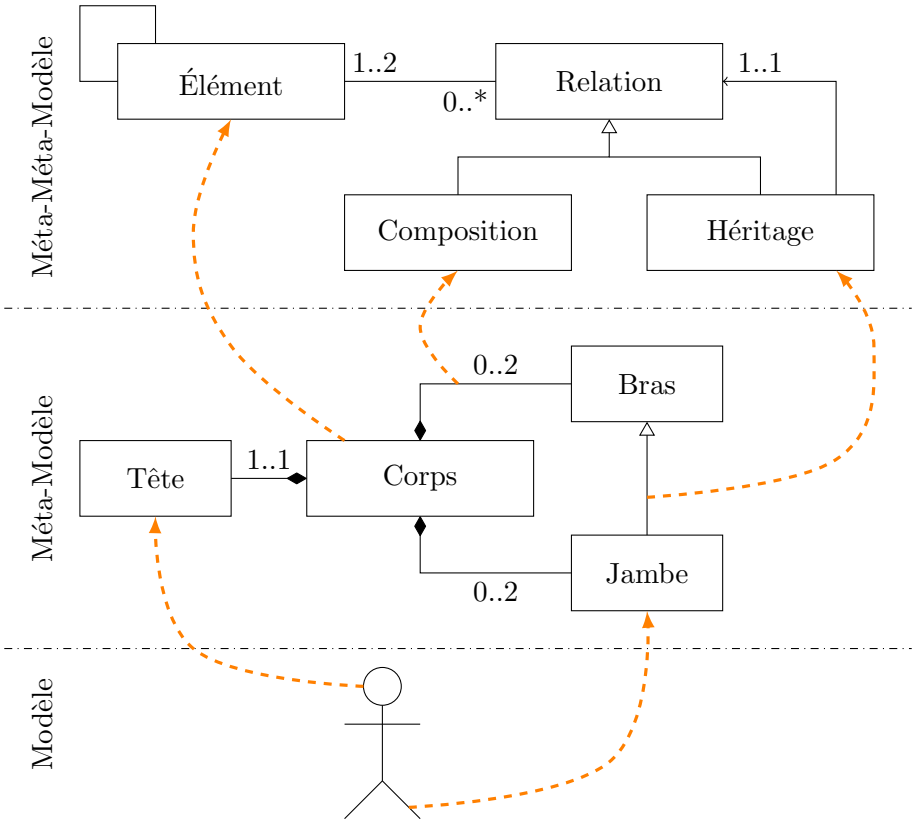


FIGURE 3.2 – Concepts de Modèle et méta-modèle

Par exemple, la figure 3.2 présente la modélisation simple d'un corps humain ainsi que les règles associées. Le modèle représente ici seulement les éléments spécifiques aux membres d'un corps humain. Le modèle est conforme au méta-modèle (sur la même figure), s'il compte au plus 2 bras, 2 jambes et possède exactement 1 tête, tous rattachés au corps. Ce méta-modèle est ensuite conforme à un méta-méta-modèle, permettant ainsi de définir les relations entre les objets dans le méta-modèle. Les objets étant ici, les membres du corps les relations entre les objets. Ensuite, non seulement le méta-méta-modèle définit le méta-modèle, mais il se définit par lui-même. L'OMG a normalisé le méta-méta-modèle dans le *Meta Object Facility* (MOF) [OMG15]. Il peut également définir d'autres méta-modèles comme le méta-modèle d'un animal par exemple.

3.3.1.2 Outillage de modélisation

Pour créer des méta-modèles, des *frameworks* permettent de créer des langages de modélisation basés sur le méta-méta-modèle MOF. Le langage le plus connu est le langage Ecore, inclus dans *Eclipse Modeling Framework* (EMF) [SBPM08a]. Le framework EMF permet d'établir un méta-modèle basé sur Ecore et de générer le code Java pour construire un langage de modélisation conforme au méta-modèle. Le méta-modèle Ecore est un méta-méta-modèle, comme MOF, proposé par l'OMG.

Le langage Kermeta [JBF11], basé sur EMF et Ecore, permet la méta-modélisation avec une annotation du code pour spécifier le comportement du méta-modèle lors de transformations de modèles par exemple.

Epsilon [KPP06] est aussi un ensemble de langages indépendants de EMF permettant la génération ainsi que la transformation de modèles entre autres. Le *framework* permet de créer un méta-modèle à partir d'un ensemble d'instructions.

3.3.1.3 Expression de contraintes structurelles

Les contraintes structurelles permettent d'assurer que le modèle est bien formé et que les incohérences sont absentes du modèle. Par exemple, selon la classe référencée, les valeurs disponibles à la modélisation ne seront pas les mêmes. Pour définir ces règles, le langage *Object Constraint Language* (OCL) [OMG06a] permet à l'expert méta-modèle de définir des contraintes sur le méta-modèle. Ces contraintes doivent être vérifiées par le modèle pour assurer une modélisation correcte selon les règles définies par l'expert méta-modèle. Les règles OCL permettent également de déterminer si une contrainte structurelle est respectée par le modèle. Le langage *Epsilon* contient également *Epsilon Validation Language* (EVL) pour vérifier les contraintes structurelles des modèles.

3.3.2 Transformations de modèles

On distingue deux grands types de transformations :
- les transformations *Model-to-Model* (M2M), c'est-à-dire de modèle à modèle
- les transformations *Model-To-Text* (M2T) qui engendrent un fichier sous forme de texte pour créer du code ou bien de la documentation à partir d'un modèle similaire à *Unified Modeling Language* (UML).

Dans les transformations M2M, on distingue :
- les transformations exogènes où les méta-modèles de départ et d'arrivée sont différents,
- les transformations endogènes où les méta-modèles de départ et d'arrivée sont identiques.

Les transformations exogènes nécessitent ainsi de faire correspondre chaque classe du méta-modèle de départ à chaque classe du méta-modèle d'arrivée. Ce type de transformation est requis pour une migration de modèles par exemple. Par conséquent, cela implique de connaître les deux méta-modèles pour éviter la perte d'informations au cours de la transformation. Par exemple, une transformation depuis UML vers un métamodèle Ecore est exogène.

Les transformations endogènes, résultent souvent d'une volonté de raffinement du modèle ou bien de détailler ou abstraire certains concepts du modèle. Par exemple, une transformation de UML vers UML est endogène.

Outillage de transformation *Query/View/Transformations* (QVT) [OMG17b] est un standard de l'OMG pour normaliser les transformations entre langages du MOF. Il est composé de requêtes (*Query*) pour sélectionner des éléments d'un modèle, de vues (Views) pour visualiser certains aspects et de transformations pour transformer les modèles.

Le langage *Atlas Transformation Language* (ATL) [JABK08] permet la transformation M2M à l'aide d'un langage de transformations s'appuyant sur une syntaxe proche de OCL.

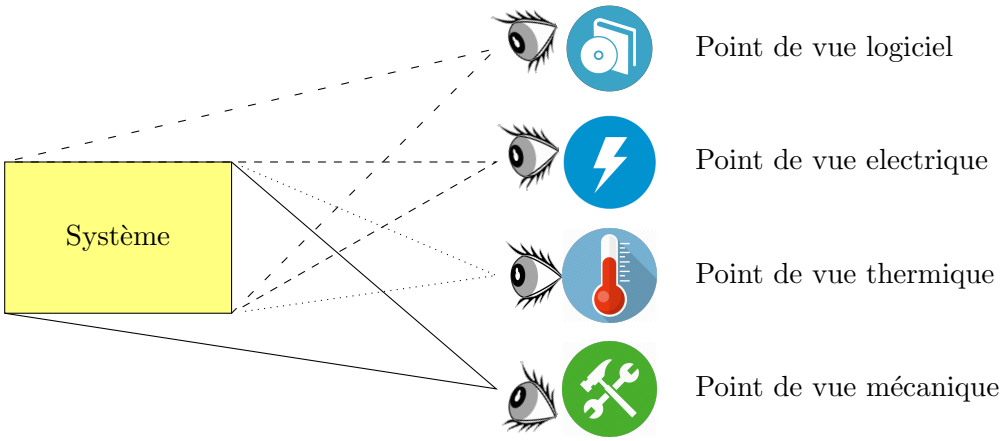


FIGURE 3.3 – Quelques points de vues existant

Acceleo [MJL+12] est un des outils qui permettent la transformation M2T pour générer du code HTML ou Java par exemple.

3.3.3 Langages dédiés de modélisation (DSML)

Dans cette section, le principe des langages spécifiques de modélisation ou *Domain-Specific Modeling Language* (DSML) est présenté pour construire des langages spécifiques aux domaines. Les DSML permettent de créer des modèles et de les manipuler simplement en utilisant des concepts métier. Chacun de ces langages est spécialisé dans un domaine particulier des systèmes, un langage pouvant être dédié au génie électrique, un autre pour l'électronique, ou encore à la partie temporelle d'un logiciel.

Un langage spécifique résulte souvent d'une grande réflexion sur les parties à représenter d'un système. Ce principe permet d'abstraire le système de façon à se focaliser sur un point de vue particulier. Comme présenté sur la figure 3.3, un système peut être vu selon différents domaines. L'ensemble de tous les modèles d'un système donné peut ainsi représenter le système dans sa globalité. Dans la figure 3.2, le corps humain n'est représenté que par des membres, il pourrait correspondre au méta-modèle sous-jacent à un DSML du corps humain vu par un enfant. Il serait donc nécessaire d'avoir un autre méta-modèle bien plus complexe si l'on souhaite créer un DSML à destination d'un médecin, et un autre méta-modèle particulièrement détaillé au niveau du cœur, sans doute moins au niveau du reste, si l'on souhaite créer un autre DSML à destination d'un chirurgien en cardiologie.

Les langages dédiés fournissent tous les éléments de modélisation nécessaires pour répondre à une problématique précise. Ces langages sont définis autour de deux principales syntaxes :

— la **syntaxe abstraite** : c'est la syntaxe utilisée pour définir les éléments ainsi que les relations entre ces éléments, c'est le méta-modèle.

— la **syntaxe concrète** : c'est une syntaxe, graphique ou textuelle, qui représente le système, c'est le modèle.

Par la suite, le modèle peut servir de base pour une génération automatique de code ou bien pour l'analyse. Il est parfois difficile de créer en entier un DSML et lorsque qu'un langage est créé, il faut, pour faciliter le processus, que toutes les parties prenantes utilisent le même langage. Des langages de conception sont donc apparus pour unifier la modélisation d'un domaine. Dans la suite, des langages orientés temps réel sont présentés.

3.4 Langages de conception des systèmes temps réel

L'ingénierie dirigée par les modèles peut être appliquée à divers domaines : pour le domaine du temps réel, des langages de conception ont été normalisés par l'industrie. Cette section se focalise sur ces langages adaptés aux systèmes temps-réel. C'est dans ces langages que l'on peut retrouver les concepts associés aux systèmes temps-réel comme les fils d'exécutions (*thread*), les processus, l'arbitrage, les ressources, la communication entre threads.

3.4.1 AADL

L'*Architecture Analysis & Design Language* (AADL)[FG12] est un standard de modélisation d'architecture logiciel/matériel normalisé par SAE International depuis 2004. Le langage fait partie des plus anciens langages adaptés pour la modélisation des systèmes temps-réel. En effet, c'est un langage dérivé du langage MetaH [VK00] développé par Honeywell à partir des années 1990. Dans cette version figurent déjà beaucoup de fonctionnalités ainsi que des objets d'AADL. Les composants matériels – comme les processeurs, la mémoire, les bus de communication– et les composants logiciels – comme les méthodes de communication, les flux de bout en bout, le temps d'exécution – faisaient partie du langage MetaH. La syntaxe ainsi que l'idée d'un langage typé furent également repris dans AADL.

Une dizaine d'années plus tard, le langage MetaH fut repris et développé pour devenir ce qu'est AADL aujourd'hui. AADL – auparavant nommé *Avionics Architecture and Design Language* – est un langage orienté temps-réel ciblant originellement l'avionique. Le langage est particulièrement adapté à la description des systèmes embarqués distribués critiques.

Pour représenter un système, il est nécessaire de respecter la syntaxe graphique représentée sur la figure 3.5 ou bien la forme textuelle tout en respectant la syntaxe (i.e. le méta-modèle). Un package, présenté sur la figure 3.4 contient les composants ainsi que l'implémentation désirée pour chaque composant. A ceux-ci peuvent s'ajouter un ensemble de propriétés ainsi que des librairies annexes.

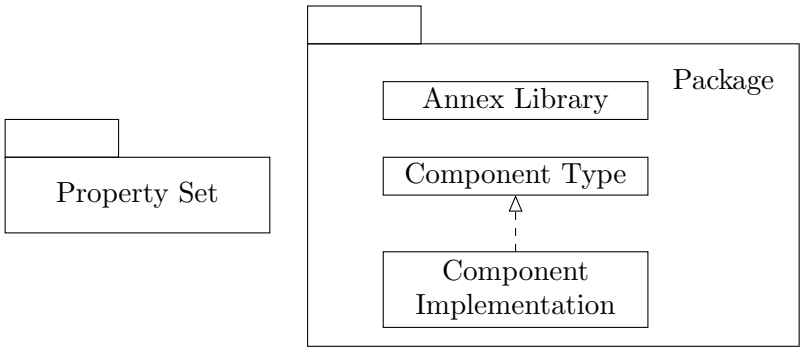


FIGURE 3.4 – Structure d'un modèle AADL [FG12]

Dans chacune des catégories, il existe une multitude de composants basiques permettant de représenter les systèmes. Les composants basiques sont généralisés de telle sorte qu'ils soient représentés par une seule catégorie, par exemple, différents processeurs peuvent être modélisées sous une même représentation. Dans la partie matérielle, les processeurs, la mémoire, les bus ainsi que les composants de façon générale sont représentés. Dans la partie logicielle, on retrouve les processus, les tâches, les sous programmes ainsi que les données nécessaires au fonctionnement du système.

Afin de manipuler le langage, des éditeurs de modélisation existent comme OSATE [Fei04]. Cela permet de manipuler aussi bien les modèles AADL sous forme graphique que sous forme textuelle.

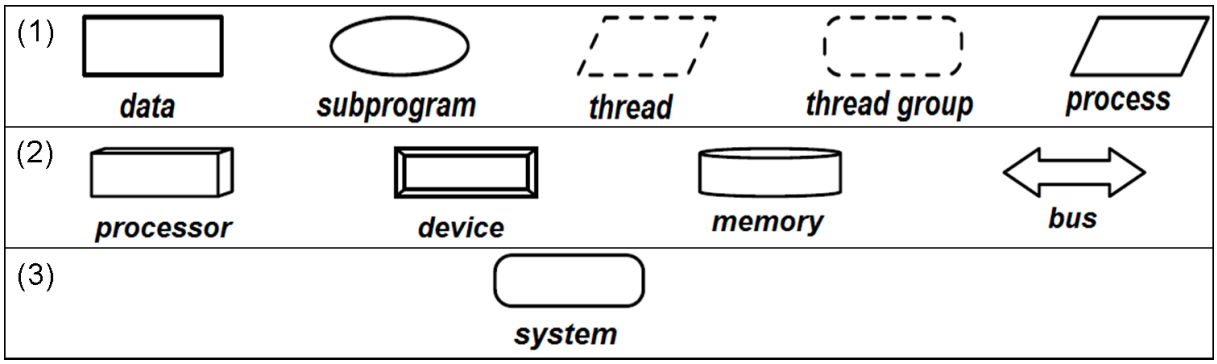


FIGURE 3.5 – Composants existants dans AADL

3.4.2 UML MARTE

UML [RBJ17] fait partie des paradigmes de modélisation les plus anciens pour les systèmes logiciels. UML fut créé dans les années 1990 pour les langages objets. Une multitude de langages pour la représentation des logiciels existaient alors. Le langage UML, lors de sa création, avait pour objectif d'unifier les langages de modélisation – d'où le le nom d'UML, *Unified Modeling Language*. À ce jour, le modèle UML version 2.5.1 publiée en décembre 2017 normalisé par l'OMG (*Object Management Group*) propose 14 diagrammes dont le diagramme de classes couramment utilisé par les développeurs.

Les besoins des concepteurs ne sont pas forcément satisfaits par les concepts proposés en UML. Des profils UML peuvent être élaborés pour correspondre spécifiquement aux domaines des parties-prenantes, comme le profil UML pour les télécommunications TelecoML [OMGa] ou le profil VOICP pour les applications voix [OMGb].

Dans le domaine des systèmes embarqués temps réel, l'OMG a adopté le profil *Modeling and Analysis of Real-time and Embedded systems* (UML - MARTE) [OMG11]. Le profil MARTE permet de modéliser les systèmes embarqués dans le détail. Le langage est composé de quatre parties représentées sur la figure 3.6 :

— Les fondations de MARTE (*MARTE foundations* dans le langage) définissant les éléments basiques de MARTE comme les propriétés, les relations entre celles-ci, les éléments temporels ou encore les propriétés non fonctionnelles du système.

— Le modèle de spécification décrivant dans le détail les ressources matérielles ou logicielles utilisées ;

— Le modèle d'analyse pour décrire les transactions, les activations, les relations entre les ressources ainsi que les scénarios de comportement possibles ;

— Les annexes de MARTE (*MARTE annex* dans le langage) définissent les outils utilisés par les autres modules de MARTE.

Ce langage est implémenté par des éditeurs de modélisation comme Eclipse Papyrus [1].

1. https ://www.eclipse.org/papyrus/index.php

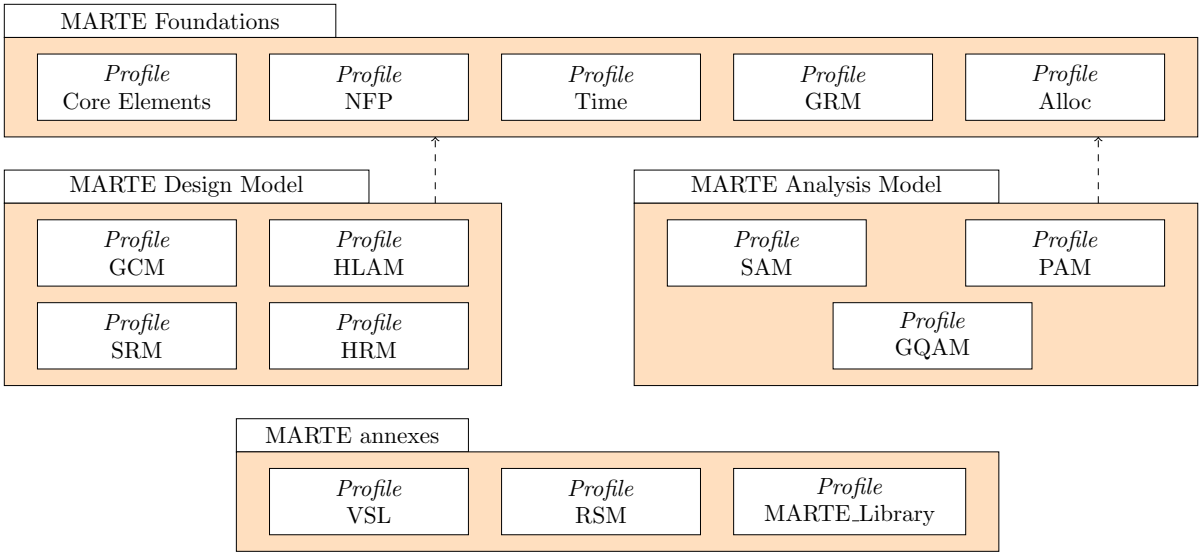


FIGURE 3.6 – Organisation et structure du profil UML - MARTE

3.5 Analyse dirigée par les modèles

De multiples paradigmes d'aide au développement existent pour améliorer et accélérer le processus de développement tout en garantissant le bon fonctionnement du système. Dans la suite, on présente des outils permettant l'analyse des systèmes temps réel. Ces outils appliquent les théories mathématiques présentées dans la section 2.3 pour déterminer les temps de réponse, par exemple, dans un système. Chaque outil possède un modèle d'entrée très souvent spécifique à celui-ci.

3.5.1 Modèles de tâches temps réel

La communauté d'analyse des systèmes temps réel a proposé, différents modèles de tâche permettant l'analyse temporelle. L'expressivité du modèle influe sur le test d'ordonnancement : plus le modèle est expressif, plus le test d'ordonnancement sera difficile à mettre en œuvre en termes de complexité.

Le modèle de tâche proposé par Liu & Layland [LL73] est simple à représenter : seuls le temps d'exécution des tâches ainsi que leur période d'activation sont nécessaires à l'analyse. En revanche, le cadre de l'analyse est très restreint, les tâches doivent être indépendantes, sur un système monoprocesseur, démarrées simultanément, ce qui n'existe pas dans les systèmes complexes d'aujourd'hui.

Néanmoins, beaucoup d'autres modèles de tâches ont été proposés. Baruah *et al.* [BCGM99] ont proposé le principe d'un modèle généralisé *multiframe* de tâches prenant en compte les tâches ayant plusieurs temps d'exécution, plusieurs échéances et plusieurs périodes d'activation possibles, généralisant ainsi le modèle de tâches sporadique de Mok [Mok83] et le modèle *multiframe* de Mok *et al.* [MC97]. Baruah a généralisé à plusieurs reprises son modèle *multiframe* : il le généralise pour des tâches récurrentes de façon restreinte (*Recurring branching model*) [Bar98], puis de façon générale (*Recurring model*) [Bar03], et enfin de façon non-cyclique (*Non-cyclic Recurring model*) [Bar10]. Ce dernier modèle est inspiré également des travaux de Moyo *et al.* [MNLM10] qui généralisent le modèle sporadique. Le modèle transactionnel de Palencia et Harbour [PH98, RGRR12], généralise également le modèle sporadique : ici, l'activation des tâches est différée et peut être retardée par des gigues. À ce jour, le modèle de tâches en digraphe, étendu d'abord à un nombre restreint de contraintes ("k-étendu") et ensuite généralisé ("étendu"), proposé par Stigge *et al.* [SEGY11, SY15] s'avèrent être le modèle le plus générique

pour représenter des tâches. Ce modèle utilise des graphes orientés pour décrire l'activation de tâches. La figure 3.7 résume les différents modèles de tâches utilisés dans le domaine de l'analyse temporelle.

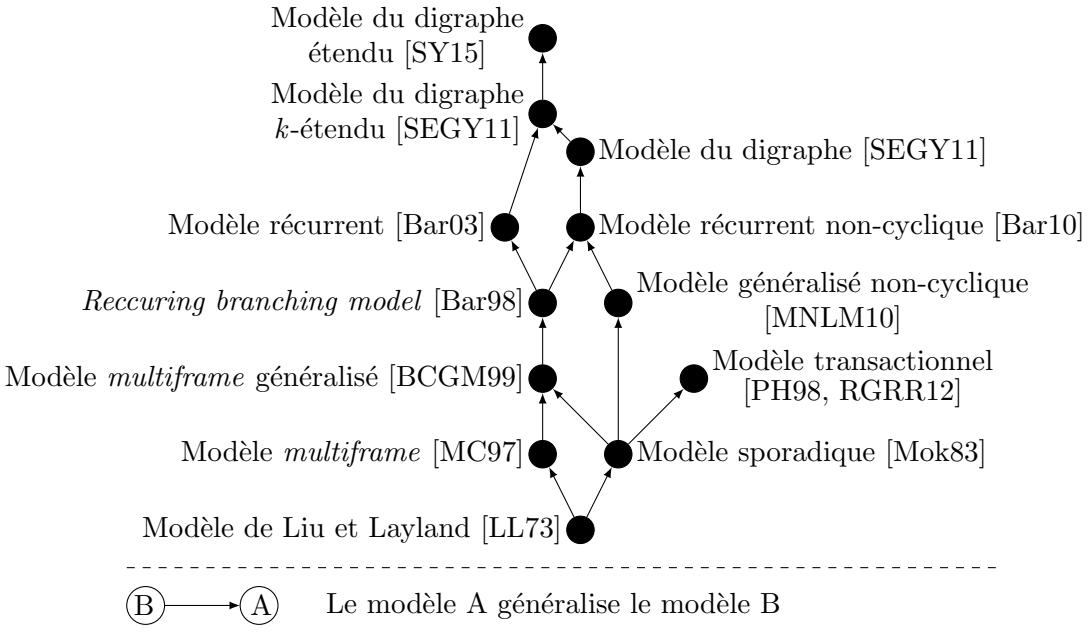


FIGURE 3.7 – Généralisations des modèles de tâches

Les travaux dans le domaine ont permis d'exhiber de multiples modèles pour l'étude et l'analyse. Cette multitude de modèles traduit la difficulté de représenter les systèmes temps réel de façon générique. Or, plus le modèle est générique, plus la quantité d'informations nécessaires est grande, plus il est difficile à analyser [SY15] mais plus l'analyse est proche du comportement du système.

Un **contexte d'analyse**, est dépendant non seulement du modèle de tâche mais aussi du matériel d'exécution et de l'algorithme d'ordonnancement. Ces trois paramètres vont ensuite régir la complexité des analyses temps réel.

3.5.2 Outils pour l'analyse temporelle des systèmes temps réels

La validation des systèmes par rapport au cahier des charges est une étape obligatoire avant la mise en service d'un système. La validation temporelle devrait se faire tout au long du développement du système mais en réalité, n'apparaît trop souvent qu'à la fin du développement.

L'analyse des systèmes temps réel peut représenter une tâche fastidieuse. Pour analyser des systèmes, des outils existent afin d'aider les ingénieurs dans l'ordonnancement des systèmes.

Une façon de vérifier les temps de réponse d'une transaction par exemple serait de simuler complètement le système. En revanche, cela implique la connaissance des analyses utilisées ainsi que leur conditions d'utilisation : la viabilité de la simulation n'est pas assurée par rapport au temps d'exécution. On peut citer comme outils de simulation, Cheddar [SLNM04] qui permet la simulation de systèmes temps réel. SimSo [CHD14, Sim14] propose également la simulation de systèmes avec la prise en compte d'algorithmes spécifiques créés par l'utilisateur. De même, Pégase++ [2] simule les systèmes distribués basés sur un réseau CAN ou AFDX.

Des logiciels d'analyse ont été développés afin de mettre en œuvre de façon automatique l'analyse des systèmes temps réel. L'analyse par réseaux de Petri temporels est outillée depuis 2005 dans Roméo [GLMR05].

2. `http://www.realtimeatwork.com/software/rtaw-pegase/`

Modeling and Analysis Suite for Real-time applications (MAST), développé depuis 2001 [HGGM01], fait partie des logiciels d'analyse les plus anciens. Le logiciel généralise les modèles d'analyse afin de faire une analyse par transaction. MAST permet également de calculer divers paramètres comme le temps de blocage avec partage de ressource critiques ou bien le temps de réponse pour un algorithme d'ordonnancement donné. Le logiciel permet également de conduire une analyse holistique à la condition d'avoir des transactions ayant un seul évènement d'activation. Notons que les industriels étant très intéressés par la notion de temps de réponse de bout-en-bout, la notion de transaction est particulièrement adaptée à de nombreux types de systèmes industriels.

Avec les progrès dans les systèmes distribués embarqués, les réseaux nécessitent également une analyse pour déterminer le temps de réponse de bout en bout c'est-à-dire depuis l'activation de la tâche jusqu'à la tâche finale. Pour analyser des réseaux, de multiples méthodes existent selon la complexité du réseau. Quelques méthodes sont présentées dans la Section 2.4.

pyCPA [DAE12] est une extension de python rendant disponibles des outils d'analyse pour des systèmes de tâches distribués qui se base sur le principe d'analyse compositionnelle. Cette extension est le cœur du logiciel SymTA/S édité par Symtavision.

Des logiciels comme MPS-CAN [MMTS14] permettent de prendre en compte les différents paramètres existants sur CAN selon la version ou les données transportées dans une trame d'informations. Il existe aussi des prototypes de laboratoires comme le calcul des temps de réponse dans un réseau commuté AFDX [KRBR14].

MAST	pyCPA
Pire temps de réponse	
Pire temps de réponse de bout en bout	
–	Maximum d'arriéré d'activation de tâches
Pire temps de blocage	–
Sensibilité du système *Slack*	–
Affectation de priorités	–

Tableau 3.2 – Analyses supportées par les outils MAST et pyCPA

Il existe donc une multitude de logiciels permettant l'analyse temporelle des systèmes. Cette multitude de logiciels implique un grand nombre de modèles d'entrée et de paramètres d'entrée et surtout une maîtrise parfaite de la sémantique des modèles d'entrée. En effet, prenons l'exemple de MAST et de pyCPA. Le tableau 3.2 résume quelques analyses supportés par ces logiciels, indépendamment des hypothèses temporelles. Il y a, certes, des analyses en commun telles l'analyse du temps de réponse mais certaines analyses ne sont disponibles que dans certains logiciels.

3.5.3 Travaux connexes

Actuellement, les langages d'analyses se basent sur des modèles d'architecture logicielle mais souvent, ces modèles ne présentent pas les éléments non fonctionnels tels les contraintes temporelles. En effet, la validation temporelle étant conservative, plus la modélisation proposée est grossière et éloignée du système réel, plus l'analyse sera pessimiste pour le système.

Bien entendu, le domaine de l'ordonnancement temps réel est sujet aux problèmes de complexité algorithmique : un modèle précis dans la description du système est essentiel dans le développement logiciel mais plus le modèle est complexe et précis, plus le test d'ordonnançabilité sera compliqué, alors que les résultats seront plus proches de la réalité. Ainsi, l'analyse exacte de modèles précis, telles par exemple les analyses basées sur les réseaux de Petri, souffrent d'ex-

plosion combinatoire, ne peuvent passer à l'échelle pour des cas réels industriels, et ne peuvent s'appliquer que sur des études de cas de petite taille.

Les langages de modélisation temps réel sont centrés sur l'architecture des systèmes et ainsi peuvent être adaptés pour diverses utilisations comme montré dans la figure 3.8. Pour analyser un modèle centré architecture, il faut mettre en place des transformations vers le modèle d'analyses.

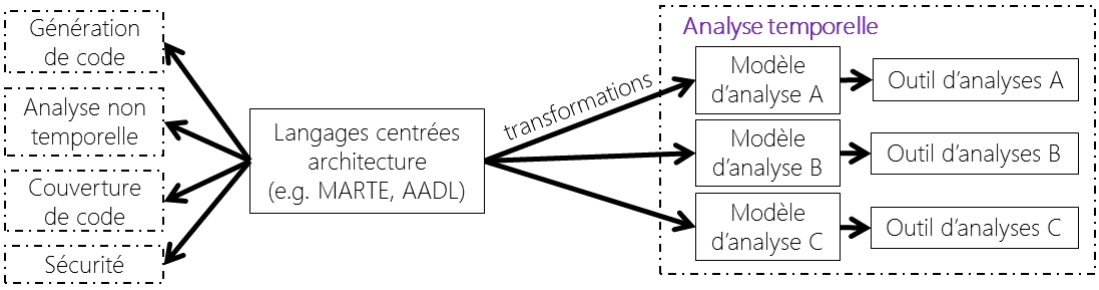


FIGURE 3.8 – Positionnement des langages d'architecture

Autrement dit, il existe un écart de syntaxe et parfois de sémantique (manque d'information sur la gestion d'une file d'attente, détails d'implémentation non précisés, etc.) entre la conception et l'analyse : il est nécessaire d'effectuer des transformations entre les modèles de conception et les modèles d'analyse. Reprenons l'exemple de MAST et de pyCPA du tableau 3.2. Étant donné un modèle exprimé dans un langage centré architecture, il faut, si possible, transformer ce modèle vers un modèle d'entrée de l'outil d'analyse choisi. Par exemple, pour un même modèle, le pire temps de réponse peut être calculé par les deux logiciels mais le temps de blocage ne peut pas être calculé par pyCPA. Réciproquement, les analyses sont dédiées à des cas spécifiques. C'est le travail de l'expert en analyses, ou à une transformation déjà implémentée, de modéliser le cas exprimé avec le langage centré architecture dans le bon formalisme conforme au modèle d'analyse.

La figure 3.9 résume les problèmes existants : dans le langage de modélisation noté B, selon que le modèle soit représenté par la méthode 1 ou la méthode 2, il ne pourra pas être analysé de la même façon dans le logiciel d'analyses. De la même façon, le logiciel d'analyse peut ne pas être compatible avec certains cas, une transformation "sur mesure" peut être nécessaire.

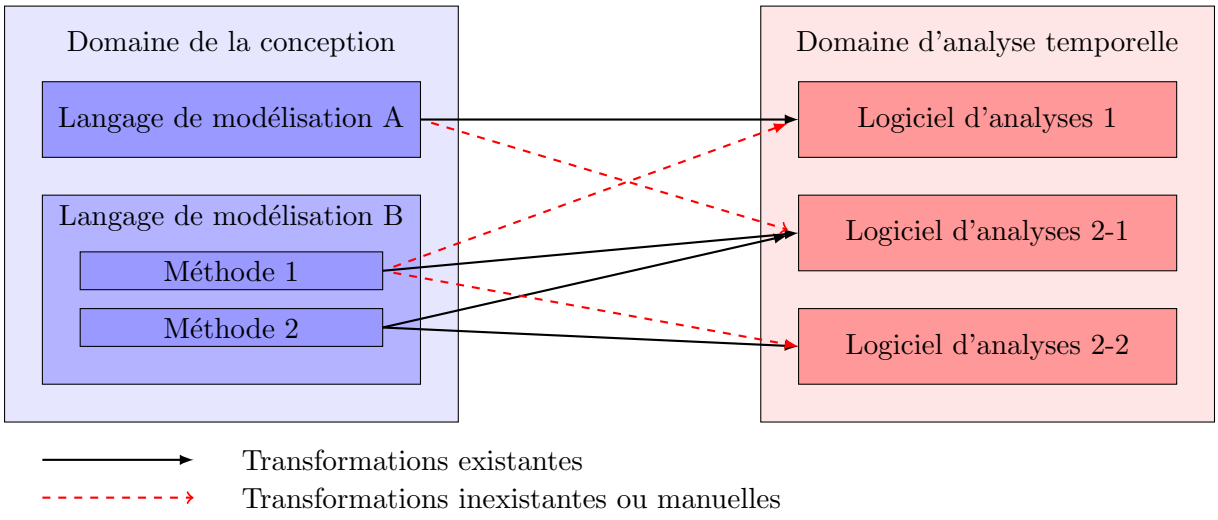


FIGURE 3.9 – Schéma de transformations entre les langages

L'analyse d'un modèle requiert du temps pour expliciter les caractéristiques temporelles du

système. De nombreux travaux ont été menés pour combiner les langages d'architecture avec les outils d'analyses. Nous citons quelques uns par la suite.

En 2016, la plateforme OCARINA [HZPK08] a été proposée pour faire le lien entre AADL et les logiciels d'analyses tels MAST et Cheddar. Cependant, cette plateforme ne considère que les architectures monoprocesseurs.

Modeling oriented Scheduling analysis of Real Time systems (MoSaRT) est un framework [Ouh13] permettant de modéliser un système en vue de l'analyser. Plutôt que de créer $m * n$ transformations entre m langages centrés architecture et n analyses se trouvant dans des outils, l'idée générale est de proposer un langage pivot centré ordonnancement permettant de créer seulement $m+n$ transformations. Des m langages centrés architecture vers le langage pivot, et du langage pivot vers les n analyses. Le tout en mutualisant l'aide au choix de la méthode d'analyse. Il s'agit ici, comme présenté dans la figure 3.10, de construire un langage de modélisation prenant en compte toutes les analyses possibles. Le langage centré architecture peut être importé dans MoSaRT et peut être ensuite enrichi par des paramètres temporels. En effet, le framework MoSaRT se compose du langage de modélisation orienté analyse et offre également un moyen de persistence de transformations nécessaires vers les outils d'analyses. Le langage MoSaRT est ainsi conçu comme un méta-modèle pivot servant de base commune entre la partie conception logicielle et la partie analyse temporelle.

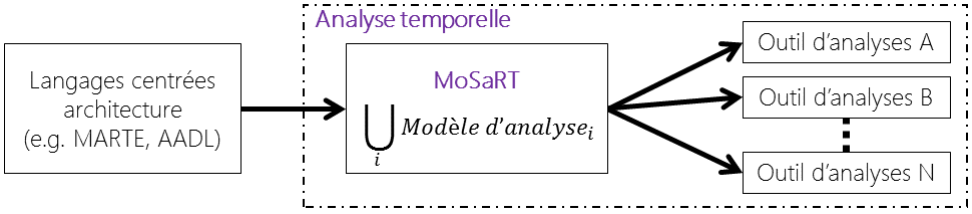


FIGURE 3.10 – Positionnement du framework MoSaRT

Le framework Tempo créé par Thales [HRS13] se propose, à partir d'un modèle, de déterminer les réponses temporelles via un logiciel externe. Le méta-modèle sous-jacent est basé sur un extrait de UML - MARTE et permet la modélisation ainsi que l'analyse à partir du modèle. En revanche, les utilisateurs de Tempo ont besoin de connaître le test d'ordonnancement associé au système avant d'initier le modèle d'analyses et ainsi analyser temporellement le système.

Par la suite, Gaudel [Gau14] a développé des patrons de conception afin de sélectionner les analyses temps-réel les plus adaptées au modèle. Cette méthode prend en entrée le langage AADL et génère une sortie en vue de l'analyse sous Cheddar. Les patrons de conceptions répondent à un certain nombre de contraintes et permettent ainsi de déterminer les paramètres d'analyse.

Récemment, les travaux de Brau et al. [BNH17] proposent MAIWEn (*Modeling and Analysis Integration Workbench for the Engineering of embedded systems*), une approche multi-couches permettant de modéliser tout le processus depuis l'architecture système jusqu'à l'analyse par un outil externe. La nouveauté du framework est l'orchestration des analyses, qui est pré-calculée par la plateforme. Cette plateforme se base sur AADL et CPAL. Les modèles basés sur un autre langage, tel UML - MARTE, ne sont pas supportés par le framework.

La plupart de ces travaux permettent l'analyse temporelle mais reposent toujours sur des langages centrés architecture logicielle. Le langage MoSaRT étant plus orienté analyse temporelle, il sera utilisé dans la suite pour implémenter les contributions, bien que de nombreux éléments aient été repris dans le cadre du projet WARUNA pour le développement de *Time4Sys*.

3.6 Framework MoSaRT

Le framework MoSaRT est un précurseur dans l'utilisation de l'ingénierie dirigée par les modèles en vue de l'analyse temporelle. Dans cette section, nous présentons la structure générale du

langage pivot composant le framework. Comme le framework MoSaRT est orienté analyse, les éléments requis pour décrire le système sont réduits simplement aux éléments nécessaires pour l'analyse des systèmes temps réel.

MoSaRT est un langage de modélisation orienté analyse temporelle des systèmes temps réel créé au laboratoire en 2013 [Ouh13]. Ce langage a été proposé car les langages de modélisation actuels souffrent d'un défaut d'expressivité pour l'analyse des systèmes temps réel.

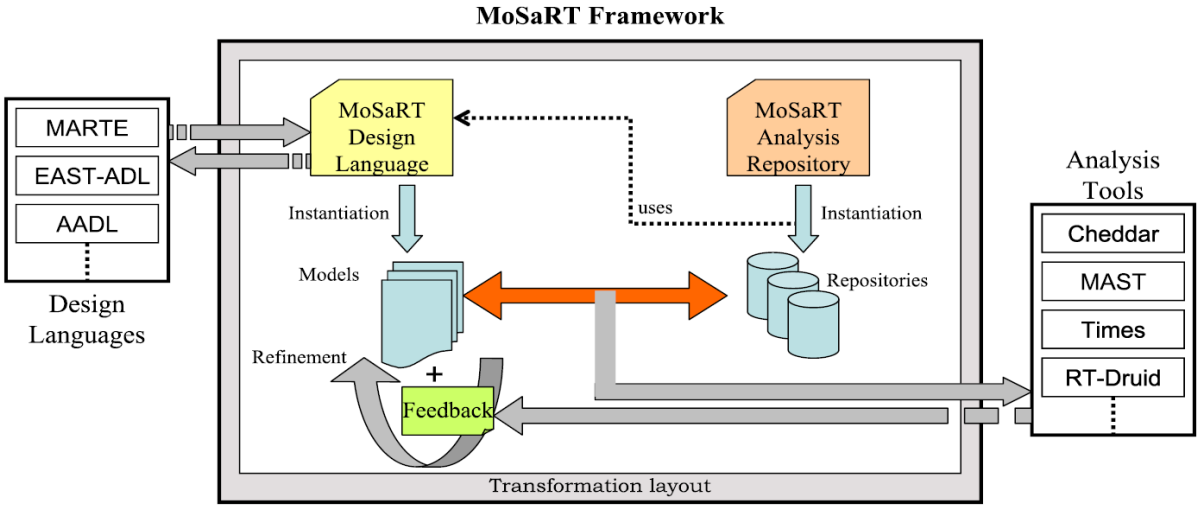


FIGURE 3.11 – Organisation du processus d'analyse à l'aide du framework MoSaRT [Ouh13]

Le processus complet du framework MoSaRT est illustré par la figure 3.11. MoSaRT définit plusieurs passerelles afin d'importer et pouvoir affiner par exemple un modèle AADL ou UML - MARTE. Par ailleurs, le modèle peut être instancié directement dans le logiciel pour être analysé par un outil. Pour décider de l'outil d'analyse à utiliser, le modèle est soumis à un processus d'identification géré par le référentiel d'analyse (*MoSaRT Analysis Repository* sur la figure 3.11). Le référentiel d'analyse est une partie de MoSaRT stockant les règles d'application des outils d'analyse. Si, pour un contexte donné, le modèle vérifie toutes les règles associées au contexte, alors il sera possible de mettre en place une analyse pour ce système via un logiciel dédié à ce contexte. Par exemple, si le système respecte les règles d'applications du modèle de tâches de Liu & Layland (évoqué dans la section 2.3), il est alors possible de tester l'ordonnancabilité avec l'équation 2.1. MoSaRT propose ensuite grâce à son référentiel d'analyse de transformer automatiquement le modèle vers le formalisme d'entrée des outils qui implémentent le test de l'équation 2.1.

Le développement du framework MoSaRT s'est basé essentiellement sur Ecore et EMF (Eclipse Modeling Framework) [SBPM08b] (Ecore servant ici de méta-méta-modèle). Ecore permet ainsi de construire le méta-modèle, qui sera présenté dans la section 3.6. EMF a facilité la création du framework comme étant un module complémentaire (*plug-in*) à Eclipse. Afin de créer l'interface d'instanciation de MoSaRT, le plug-in Sirius [MP15] a été utilisé permettant ainsi de créer une interface graphique et facilitant la création de la synxtaxe concrète graphique.

La structure de la racine du méta-modèle MoSaRT est présentée sur la figure 3.12. Toute instance de la classe `GlobalSystem` représente le système complet et se compose donc d'une partie matérielle (`SystemHardwareSide`), d'une partie logicielle (`SystemSoftwareSide`) et d'une partie fonctionnelle (`SystemFunctionalSide`). Le comportement temporel des tâches (`GlobalBehavior`) est adjoint à la partie logicielle.

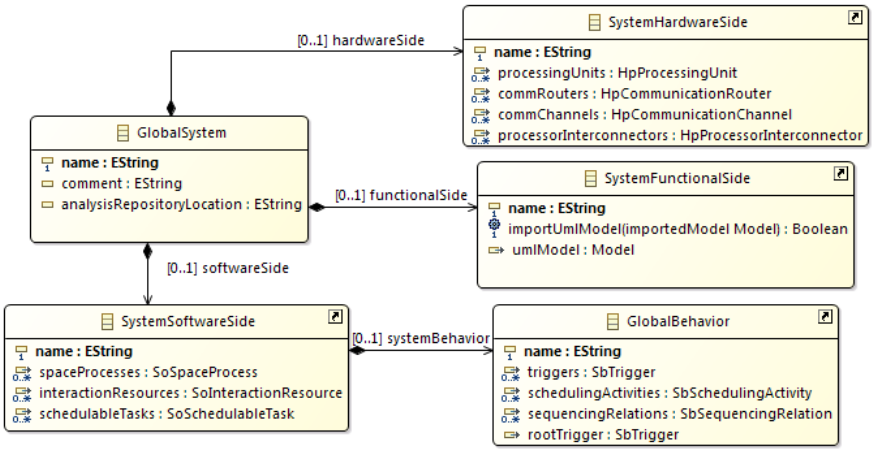


FIGURE 3.12 – MoSaRT - Racine du méta-modèle [Ouh13]

3.6.1 Partie logicielle

Dans cette partie, nous présentons la partie logicielle du langage MoSaRT et les méta-modèles associés au langage.

3.6.1.1 Structure logicielle

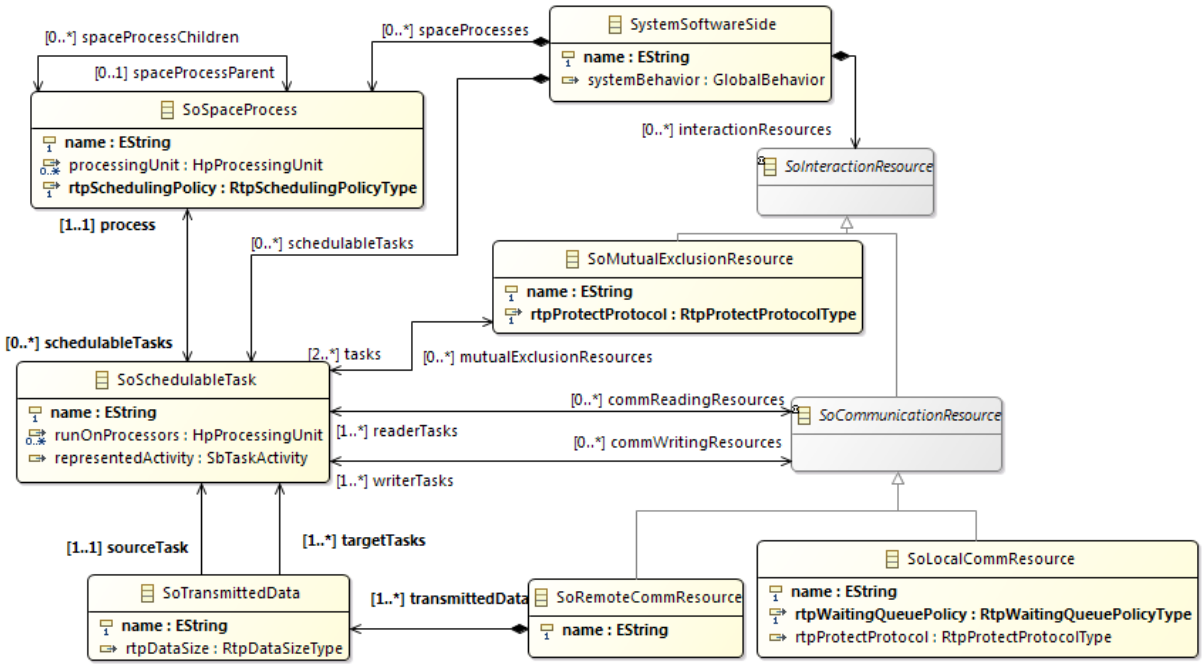


FIGURE 3.13 – MoSaRT - Partie structure logicielle [Ouh13]

La figure 3.13 présente le méta-modèle associé à la structure logicielle du système. La partie logicielle du modèle se compose de tâches (`SoSchedulableTask`) qui doivent être réparties sur les processus (`SoSpaceProcess`). Les tâches s'exécutent sur un processeur dans la partie matérielle. Les tâches peuvent communiquer à l'aide de ressources critiques, typiquement un bus ou bien un emplacement mémoire. Ces ressources critiques peuvent être de plusieurs types, elles sont généralisées sous la forme d'une classe `SoInteractionResource`. Il est ainsi possible de modéliser sous cette classe des ressources protégées par une exclusion mutuelle (`SoMutualExclusion`) ou bien par un protocole à priorités plafond par exemple. Pour un système distribué, la donnée transférée

est spécifiée dans la classe `SoRemoteCommResource`. Enfin, la taille des données transmises ainsi que les tâches réceptrices ou émettrices peuvent être spécifiées dans `SoTransmittedData`.

Dans chacune de ces classes, des paramètres temporels sont spécifiés permettant d'affiner les résultats d'analyse. Dans la suite, ces classes commençant par "Rtp" pour *Real Time Property*. Par exemple, la politique d'ordonnancement (`rtpSchedulingPolicy` dans `SoSpaceProcess`), ou bien la méthode de synchronisation (`rtpProtectProtocol` dans `SoLocalCommResource`) peut être spécifiée. Ces éléments ne sont pas suffisants pour décrire entièrement un système temps réel. Il manque des éléments essentiels liés au comportement du système (comme la périodicité des tâches) pour conduire une analyse temporelle. Ces éléments sont présentés dans la section suivante.

3.6.1.2 Aspects temporels pour le comportement logiciel

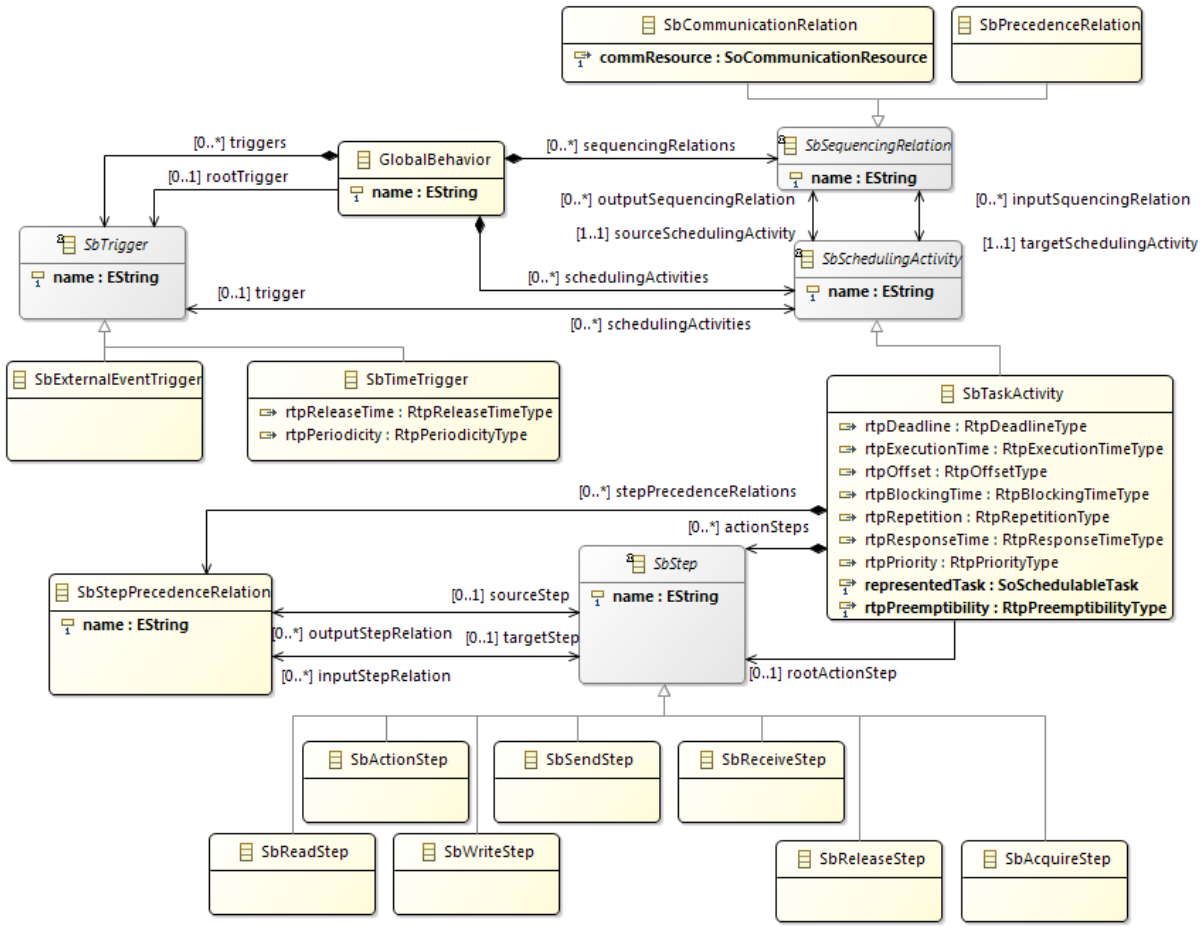


FIGURE 3.14 – MoSaRT - partie aspects temporels logiciel (ou comportement logiciel) [Ouh13]

La figure 3.14 présente les éléments permettant la description du comportement temporel dans le modèle MoSaRT. La partie "comportementale" est composée de `SbTaskActivity` représentant l'activité d'une tâche et celle-ci se compose des étapes `SbStep` composant l'activité d'une tâche. Ces étapes peuvent être de différents types :

- une opération simple du logiciel ;
- une lecture/écriture d'une ressource locale (`SbReadStep` ou `SbWriteStep`) ;
- un envoi/réception de message par une ressource à distance (`SbSendStep` ou `SbReceiveStep`) ;
- une opération de type prendre/vendre un sémaphore (`SbReleaseStep` ou `SbAcquireStep`).

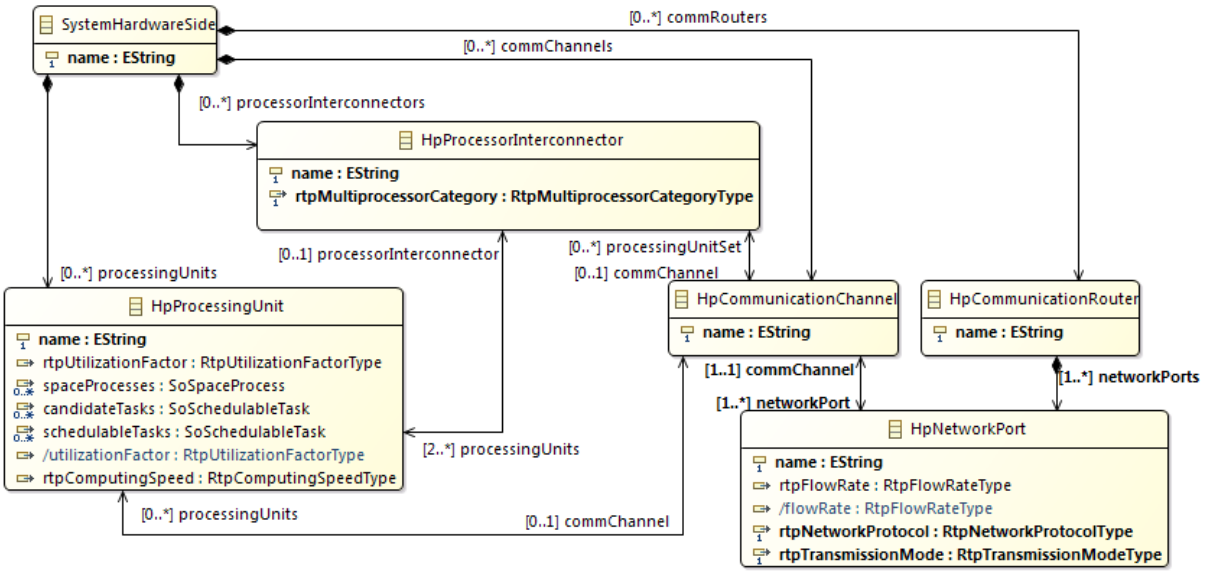


FIGURE 3.15 – MoSaRT - partie matérielle [Ouh13]

Une tâche peut être activée par un évènement extérieur ou bien par le temps (`SbTimeTrigger`). Si nécessaire, une relation de précédence `SbSequencingRelation` entre les tâches peut être instanciée que ce soit pour modéliser une relation de communication ou une relation de synchronisation.

3.6.2 Partie matérielle

Le support d'exécution est primordial dans la modélisation car selon les processeurs présents dans le système, selon le réseau utilisé et surtout la répartition des tâches dans le système, l'ordonnancement des tâches n'est pas le même. La figure 3.15 présente le méta-modèle associé au matériel. La partie matérielle comprend donc des processeurs (`HpProcessingUnit`), possiblement des multiprocesseurs et des réseaux (`HpCommunicationChannel` et `HpNetworkPort`). Pour spécifier le réseau, la classe `HpCommunicationRouter` permet de choisir le protocole réseau souhaité par le concepteur.

3.6.3 Une aide à l'analyse temporelle : référentiel d'analyse

La multitude de tests d'ordonnancabilité présentés dans la section 2.3 représente une quantité considérable d'analyses possibles par les logiciels (affectation de priorités, affectation de tâches, calcul de temps de réponse). Afin de déterminer la bonne analyse ou bien le bon logiciel à utiliser, l'expérience d'un analyse en temps réel est souvent nécessaire. Cette étape est donc coûteuse en temps et en formation : il est nécessaire que l'ingénieur soit formé sur le logiciel d'analyse, et qu'il soit apte à comprendre un système depuis la conception jusqu'à l'analyse.

La plateforme MoSaRT propose de faire une pré-analyse du système en vue de guider la conception d'abord. La figure 3.11 montre le processus : le modèle créé sera traité par un référentiel d'analyse représentant les connaissances nécessaires pour le choix de test d'analyse adéquat.Ce référentiel de connaissances contient un ensemble de contextes, chacun étant défini par un ensemble de règles basiques que le modèle doit vérifier pour être conforme au contexte. La figure 3.16 présente le méta-modèle du référentiel d'analyse.

Le référentiel d'analyse se compose de tests d'ordonnancement analysant les systèmes, ainsi que d'outils implémentant ces tests. Un ensemble de règles contenues dans le référentiel d'analyses permettent de vérifier les conditions d'applicabilité d'un test. Ainsi, un test ne peut être appliqué que lorsque le contexte y correspondant est vérifié par un certain nombre de règles.

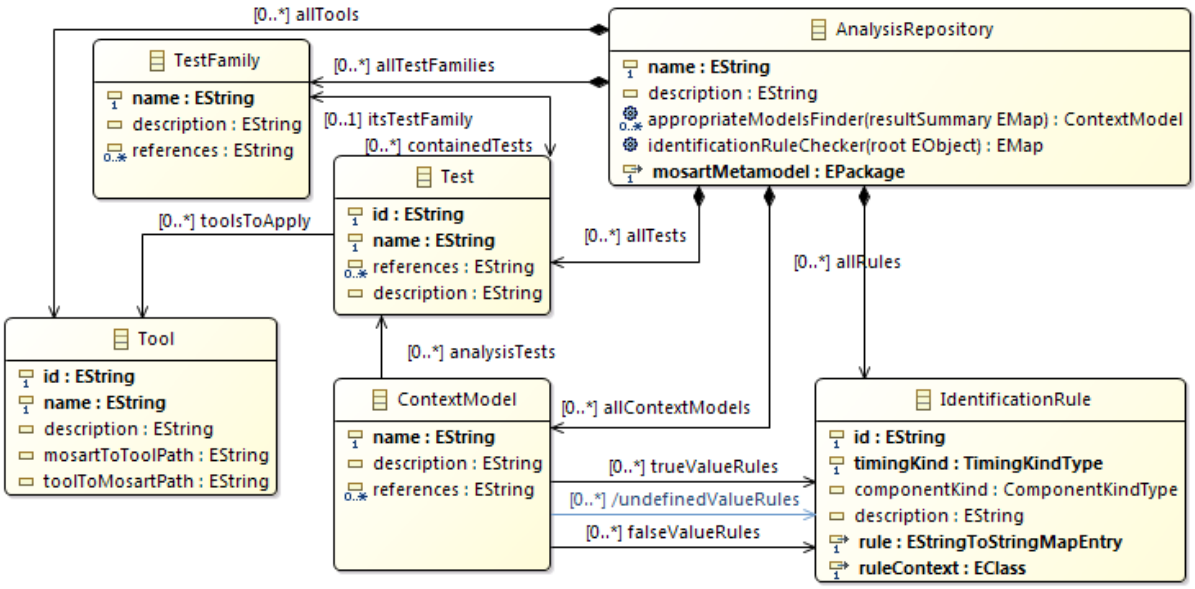


FIGURE 3.16 – Référentiel d'analyse : méta-modèle

3.6.4 Exemple de modélisation

Un exemple de modélisation est présentée sur la figure 3.17 . Le langage permet la modélisation incrémentale, c'est-à-dire qu'à partir des éléments existants, il est possible de construire un système au fur et à mesure. En effet, dans le cadre de MoSaRT, il est possible de s'arrêter à la modélisation de l'architecture système et temporelle de la figure 3.17 (respectivement *Software architecture diagram* et *Software behavior diagram* sur la figure) sans support matériel pour, par exemple, analyser l'affectation des tâches possible sur n processeurs.

3.6.5 Comparaison des langages pour l'analyse des systèmes temps réel

Les langages de modélisation dédiés (ou DSML) sont souvent crées car ils comblent des besoins que les langages existants ne peuvent remplir. En effet, quand certains langages sont adaptés pour le concepteur comme AADL ou UML - MARTE, d'autres sont faits pour l'analyse tels les modèles intégrés dans les logiciels d'analyse. Les langages de conception sont faits selon différentes approches. Pour AADL en tant que langage de description d'architecture, une approche par composants fut choisi pour ainsi abstraire les composants réels ce qui rend un modèle assez proche de la réalité. Le profil UML - MARTE permet d'effectuer une modélisation en profondeur, tous les détails du composant peuvent être adjoints à l'instance d'un composant faisant ainsi un profil très détaillé.

MoSaRT est un langage qui est orienté vers l'analyse des systèmes temps réel. C'est donc un langage qui va se placer lors des tests logiciels pour évaluer que le logiciel vérifie bien les spécifications non fonctionnelles temporelles. Par ailleurs, pour préparer cette analyse, seuls les paramètres utiles pour l'analyse sont présents réduisant ainsi le nombre d'objets à manipuler.

La sémantique de chaque langage varie grandement en raison des approches utilisées. La représentation AADL est claire grâce à l'approche composants permettant ainsi de ne pas autoriser plusieurs possibilités pour un même système physique. Ceci n'est pas le cas du profil MARTE : en effet, il est possible de modéliser un processeur de plusieurs façons différentes tout comme les tâches car le modèle peut être construit de façon incrémentale, c'est-à-dire qu'un processeur peut avoir plus ou moins de détails selon le niveau d'abstraction du modèle.

En allant plus en profondeur dans la comparaison, considérons la modélisation des réseaux. AADL et MoSaRT prennent une approche simple : le réseau est spécifié par une seule classe sans

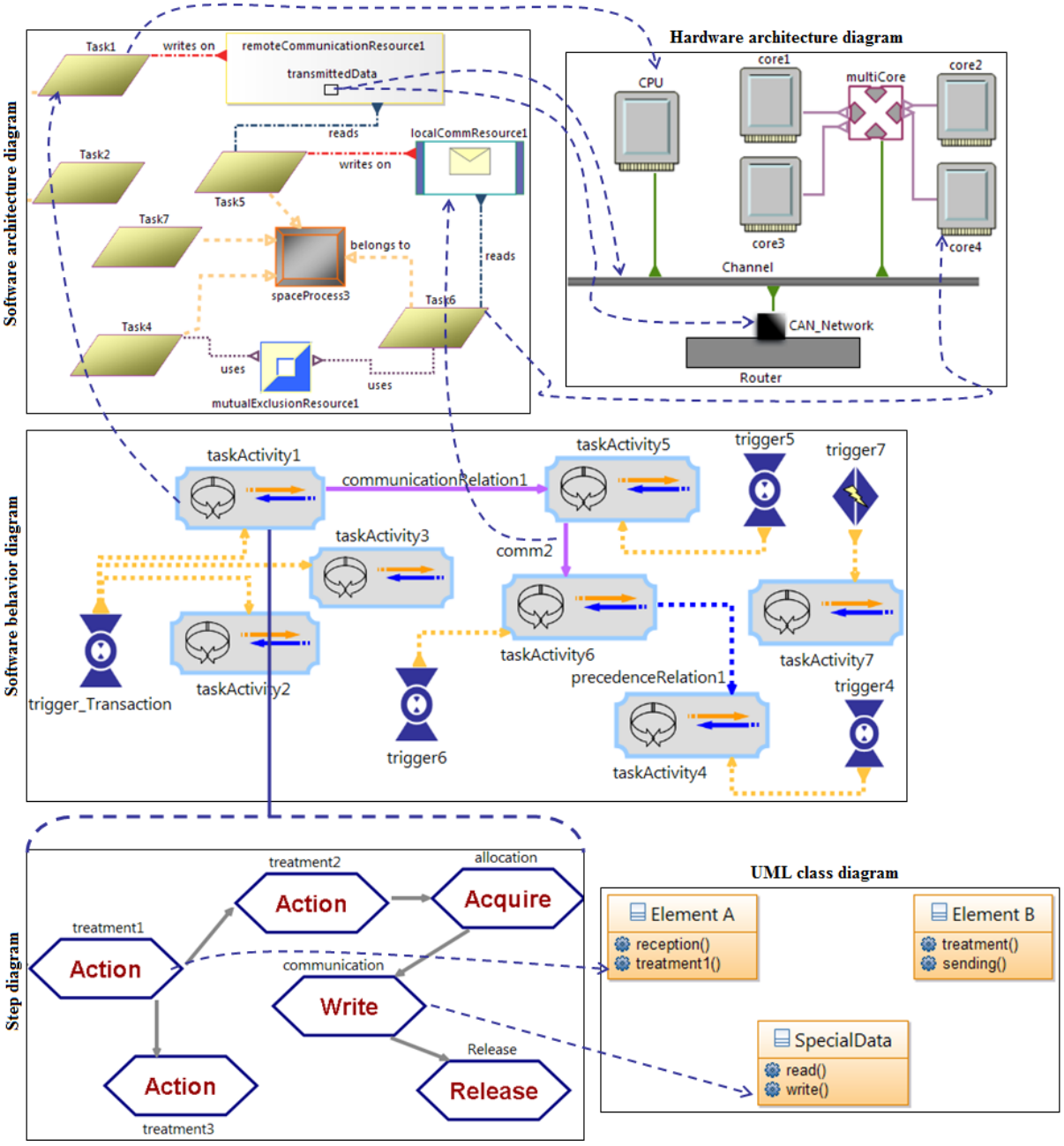


FIGURE 3.17 – Exemple complet modélisé sous MoSaRT [OGR+15]

pouvoir détailler de commutateur réseau. Dans AADL, les réseaux peuvent être détaillés à l'aide d'annexes AADL définis par l'utilisateur et ainsi étendre le langage. Cependant, la sémantique sous-jacente ne se comprend qu'à la lumière de l'annexe ainsi définie, qui pourra être normalisée par SAE ou non. Les réseaux commutés dans UML - MARTE peuvent se représenter mais pas de façon explicite : des informations manquent telles que la quantité d'informations transportée, et la quantité d'objets à représenter peut se retrouver conséquent pour représenter un simple réseau.

Les modèles peuvent ainsi représenter des systèmes mais peuvent également être utilisés pour l'analyse du système. Pour contenir les informations temporelles sur le réseau, il existe différentes possibilités selon le paradigme. Par exemple, le délai de bout en bout est représenté par un attribut *End To End Flow* dans AADL, par une transaction dans UML - MARTE. Dans MoSaRT, cela se traduit par le choix du paramètre à calculer, c'est-à-dire qu'un paramètre

temporel peut être soit déterminé à l'avance, soit être à calculer par le logiciel d'analyse défini par le concepteur.

Il existe plusieurs outils d'analyse des modèles temps réel. Un modèle AADL peut être analysé par Cheddar ou MAST par exemple via une transformation exogène de modèle, de AADL vers Cheddar [SLNM04] ou MAST [PGA+11]. En revanche, les transformations depuis AADL ne prennent pas en compte les annexes utilisateurs écartant ainsi l'annexe dans le modèle d'analyse. Des transformations existent également depuis UML - MARTE vers Cheddar [MV07] ou vers MAST [MGC11]. De même, depuis MoSaRT, il existe des transformations vers MAST et Cheddar pour permettre l'analyse du système.

	AADL	UML ProfileMARTE	MoSaRT
Utilisateur	Ingénieur Concepteur		Ingénieur Analyste temps réel
Phase de développement	Conception		Test et analyse logicielle
Approche de modélisation	Approche Composants	Approche incrémentale	Approche analyse
Représentation temporelle des tâches	Sémantique claire, uniforme	Sémantique dépendante de l'utilisateur	Sémantique unique réduite à l'analyse
Finesse de représentation des réseaux	Finesse liée à des annexes utilisateur non normalisées	Serveur réseau, Peu explicite	Une seule classe représentant le réseau
Flux de bout en bout	"End to End flow"	Représentation de transactions	Reconstruction lors de la transformation
Analysabilité des réseaux avec cette représentation	Possible mais syntaxe (AFDX, par ex) non normalisée	Possible mais syntaxe peu claire	Possible mais pessimiste

Tableau 3.3 – Comparaison des langages dédiés

De grandes différences existent entre les trois langages de modélisation : le tableau 3.3 résume les différences, les avantages et les inconvénients de ces trois langages.

3.7 Points de vues d'un système

Avec la complexité croissante des systèmes temps réel, il peut devenir nécessaire de réduire les modèles afin de mieux les maîtriser. En effet, manipuler des modèles de grande taille peut rendre le traitement des données plus lent pour le charger dans le logiciel de modélisation ou bien pour l'analyser dans le logiciel d'analyses. De plus, un ingénieur doit connaître le système complet pour le manipuler que ce soit pour l'analyser ou bien simplement modéliser l'architecture logicielle. Lors du processus de développement, la modélisation requiert du temps pour prendre en main toutes les fonctionnalités d'un modèle utiles pour l'utilisateur. Simplifier le processus de modélisation et de validation temporelle est ainsi important dans le domaine de l'ingénierie système. De plus, certains outils ne peuvent analyser qu'une partie d'un modèle, mais pas l'autre, et il existe des méthodes, par exemple pour l'ordonnancement temps réel, l'analyse holistique, qui pourraient servir de *glue* entre l'analyse de nœuds individuels indépendamment par des outils, l'analyse du réseau par un autre, l'analyse holistique permettant de choisir les paramètres de gigue à passer en entrée de chaque analyse.

Des efforts ont été effectués dans le sens de la simplification de modélisation des systèmes.

En effet, le découpage de modèles permet de réduire le champ d'applications d'un méta-modèle. A partir de ce principe, on peut en tirer deux possibilités présentées sur la figure 3.18 :
- découper un modèle pour produire des sous-modèles dérivés (voir la figure 3.18 (a)),
- découper un méta-modèle afin de ne garder que les éléments nécessaires à une modélisation et ainsi créer un point de vue spécifique de modélisation (voir la figure 3.18 (b)).

Le découpage de modèle peut être appliqué dans domaine temps réel : comme présenté dans la figure 3.18 (a), il s'agirait d'analyser séparément les sous-modèles résultant du modèle original. En se sens, il serait possible de diviser les modèles temporels en fonction des processeurs, des processus, des partitions de tâches, ou des réseaux afin d'analyser localement le modèle. De façon plus large, les résultats locaux peuvent être utilisés dans une analyse holistique (présenté dans la section 2.4.2.3) en les regroupant dans le modèle global original.

Le découpage de méta-modèle consiste en la production d'un sous-méta-modèle sur lequel l'architecte logiciel peut se baser pour modéliser un système. En effet, les langages actuels sont très grands pour permettre la modélisation d'un grand nombre de systèmes. Par exemple, le langage UML - MARTE compte près de 350 classes : manipuler ce langage requiert beaucoup de temps de formation. Pour le domaine temporel, l'idée est donc de réduire le langage à un domaine plus précis tel l'*Integrated Modular Avionics* (IMA) [Pri92] ou les systèmes monocoeurs. En restreignant ainsi le langage aux monocoeurs, les classes concernant les multiprocesseurs ainsi que les réseaux n'apparaissent pas dans le langage. De plus, dans le cadre de systèmes distribués, cela permet de développer de façon séparée le système en sectorisant le système : un point de vue réseau, et autant de modèles que de sous-systèmes (un point de vue actionneur, un point de vue capteur, etc).

Dans la suite, on présente des approches existantes afin de découper un modèle au sens général.

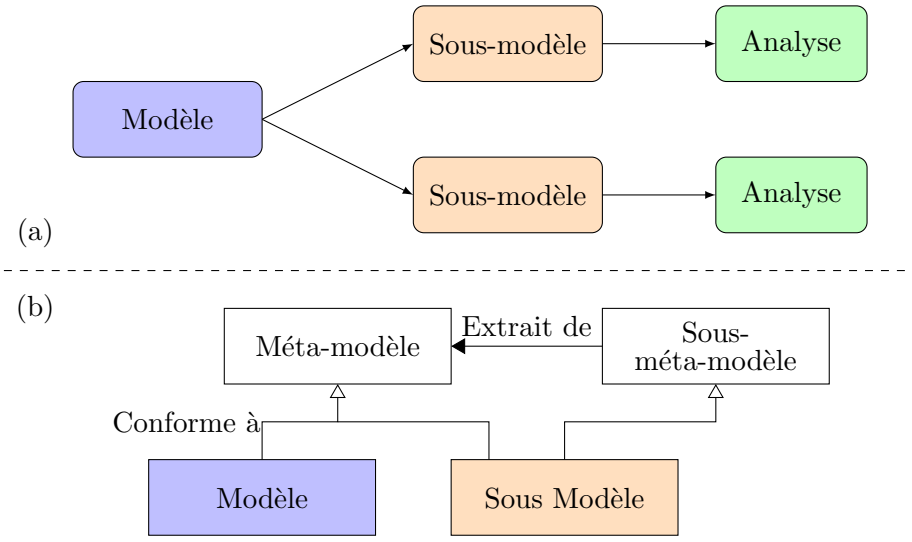


FIGURE 3.18 – Principe de découpage de (a) modèle, de (b) méta-modèle

3.7.1 Découpage de modèles

3.7.1.1 Principe

Le principe de découpage de modèle ou d'élagage de modèle dérive du découpage logiciel qui retire du code les éléments non nécessaires à un comportement demandé. S'inspirant ainsi de ce principe décrit dans la figure 3.18, le découpage de modèles (*Model Slicing* ou *Model pruning*) consiste à créer un sous-modèle toujours conforme au méta-modèle de base. Au final, il s'agira de diviser le modèle afin d'analyser de façon séparée le modèle global.

3.7.1.2 Outils de découpage de modèles

L'un des premiers travaux sur le sujet concerne le langage UML par Solberg *et al.* [SFR05] définissant ainsi l'extraction de points de vues sur les modèles UML. Des outils tels TOPCASED [FGC+06] découpent UML en points de vues avec un code incorporé au programme. Le logiciel fut depuis remplacé par Papyrus [LTE+09]. Beaucoup de travaux ont été effectués sur le découpage de modèles mais très souvent, ces recherches s'adressent spécifiquement au langage UML [BC08, SM09, LKR10]. Par exemple, le projet Java *Splittr* [SRTC14] permet de récupérer une sous-modèle UML par récupération d'informations textuelles.

3.7.1.3 Découpage de modèles temps réel

Dans le domaine des systèmes temps réel, Richter *et al.* [Ric05, HHJ+05] propose d'analyser, en suivant l'approche de composition de systèmes consistant à partager un modèle selon plusieurs sous-systèmes. Le logiciel détermine l'ordonnancement local aux sous-systèmes, crée des évènements d'activation de l'ordonnancement local à destination des autres sous-systèmes puis propage les évènements pour calculer les pires temps de réponses. Ce processus est répété jusqu'à obtenir un point fixe représentant le résultat. Ce principe est implémenté dans SymTA/S [Ric05] et pyCPA [DAE12].

Cette analyse des temps de réponses ne divise pas réellement les modèles pour les analyser séparément : tous les sous-systèmes sont analysés par le même logiciel. Or, cette analyse peut se révéler pessimiste car des analyses plus proches de la réalité peuvent exister de façon locale dans d'autres logiciels.

3.7.2 Découpage de méta-modèles

3.7.2.1 Principe

Dans cette section, on présente les travaux existants concernant le découpage de méta-modèles pour ainsi engendrer des sous-méta-modèles sur lesquels l'utilisateur pourra baser sa modélisation. De façon générale, Sen *et al.* [SMBJ09] a proposé de généraliser le découpage de modèles aux DSML. Un algorithme de découpage pour extraire un sous-méta-modèle dérivant du méta-modèle de base selon les contraintes désirées par l'utilisateur. En reprenant l'exemple de la figure 3.2, si on ne retient pas l'élément "Tête" du méta-modèle, après découpage, le modèle deviendra un corps sans tête.

3.7.2.2 Outils de découpage de méta-modèles

À partir des travaux de Sen *et al.*, le framework *Kompren* a été créé [BCBB11, BCBB15]. Ce framework prend en entrée un méta-modèle basé sur Ecore et un modèle défini sur ce méta-modèle. L'utilisateur définit une fonction de découpage qui sera utilisée comme base pour rendre un modèle découpé. Cette fonction de découpage est définie avec les classes du méta-modèle qui doivent être retenues dans le modèle découpé.

Kelsen *et al.* [KMG11] propose de mettre en place un algorithme permettant de construire des sous-méta-modèles conformes au méta-modèle original, algorithme implémenté dans le framework *Democlès*.

En se basant sur le méta-modèle Ecore, la création d'un framework de modélisation est aisée. Garmendia *et al.* ont développé *EMF Splitter* [GGKL14], un module complémentaire à Eclipse adapté pour les langages développés sous Ecore. *EMF Splitter* permet ainsi de développer des plug-in Eclipse à partir de sous modèles Ecore crées par l'outil par le biais d'annotations données par l'utilisateur. L'outil propose également de mettre en place la composition des sous-modèles

pour n'en créer qu'un seul. cet outil est intégré à un plug-in de plus grande envergure : *DSL-tao* [GPGdL15]. Pour réduire un méta-modèle Ecore, l'utilisation du langage ATL est possible, par une transformation endogène, en spécifiant les classes du méta-modèle à ne pas transformer avec le mot-clé *drop*.

3.7.2.3 Découpage de méta-modèles temps réel

Les langages de modélisation du temps réel sont très centrés sur l'architecture logicielle comme vu dans la section 3.5.3. En particulier, le profil UML - MARTE en proposant une orientation temps réel du langage UML. Le *framework Tempo* [HRS13] prend un sous ensemble du langage UML - MARTE pour conduire des analyses temporelles mais cette extraction est incorporée au programme n'autorisant pas l'extraction d'un processus par exemple. Tempo propose ainsi un point de vue de UML - MARTE se focalisant sur les aspects non fonctionnels du profil.

Une ontologie de domaine est un modèle de données représentant les concepts de ce domaine ainsi que les relations entre eux. C'est une représentation qui doit être sémantiquement complète et sans ambiguïtés. Les travaux de Hacid [Hac18] permettent la mise en correspondance entre une ontologie de domaine et un méta-modèle par le biais d'annotations sur le méta-modèle. Les annotations sont effectués par l'expert du méta-modèle pour faire correspondre un méta-modèle avec une ontologie. La méthode va donner un méta-modèle enrichi par des annotations. Cette méthode implique de connaître le langage dans lequel l'ontologie est décrite. Ensuite, un modèle est créé, rassemblant toutes les classes qui doivent être retenues dans le point de vue. À l'aide du méta-modèle enrichi ainsi que du modèle des classes à retenir, un méta-modèle réduit est généré. Pour exécuter cette méthode, l'utilisateur doit entrer classe par classe les éléments du méta-modèle requis pour obtenir le point de vue demandé par celui-ci : le méta-modèle du langage utilisé doit être connu.

Le profil Ravenscar, créé en 1997, s'applique au langage de programmation ADA [ISO12] pour restreindre le langage aux éléments nécessaires pour la programmation temps réel. Il est formalisé plus tard par Burns [Bur99]. Ce profil a pour finalité, entre autres, d'aider l'analyse formelle de ces programmes.

Le principe de découpage de méta-modèles permet ainsi la création de points de vue correspondant au domaine de l'architecte logiciel. Ces points de vue permettent de réduire le modèle aux connaissances de l'architecte et par conséquent réduit le risque d'erreur de conception.

En revanche, il n'existe pas de découpage de méta-modèle temps réel assez puissant pour répondre à nos besoins. En effet, nous voulons un découpage qui puisse être dynamique pour adapter les sous-modèles pour unifier plusieurs outils d'analyses locales. Ce découpage doit se faire à l'aide de patrons de conception pour assurer la validité du sous-méta-modèle.

3.8 Conclusion

Ce chapitre d'ingénierie dirigée par les modèles présente des techniques de développement utilisées dans le domaine industriel. Divers processus de développement existent selon le domaine d'application du système conçu. Ce processus permet ainsi le développement de systèmes complexes ainsi qu'un temps réduit de mise sur le marché.

Dans le domaine du temps réel, les langages permettent de modéliser les systèmes avec assez de précision les spécifications non fonctionnelles. Ce domaine s'inscrivait difficilement dans ces processus de développement, et par conséquent cela implique du temps de vérification et du temps pour rectifier le programme le cas échéant.

Pour réduire ce temps d'analyse, des outils ont été proposés pour appliquer les recherches menées dans le domaine temporel mais cela crée une multitude de logiciels requérant ainsi de se

former sur tous ces logiciels afin de conduire une analyse. Des outils ont donc été proposés afin de lier les langages de modélisation du type AADL ou UML - MARTE aux logiciels d'analyses.

Pour lier les langages de modélisation à l'analyse temporelle, MoSaRT propose un langage orienté sur l'analyse des systèmes temps réel. Le langage MoSaRT se propose de prendre en compte le domaine temps réel dès la conception et ainsi réduire les risques de problèmes d'ordonnancement rencontrés tardivement dans le développement impliquant dans ce cas une nouvelle implémentation du système.

Les DSML sont souvent composés de méta-modèles très grands demandant ainsi une expertise dans le domaine. Le principe de découpage de modèles et de méta-modèle est proposé pour réduire le champ d'application des langages. Ceci permet au final de proposer des points de vue à l'utilisateur, en cachant les objets inutiles à l'architecte logiciel.

Dans la suite de ce mémoire, nous allons proposer des solutions supplémentaires afin d'améliorer le processus de modélisation et de vérification temporelle dans le cadre de l'ingénierie dirigée par les modèles.

Deuxième partie

Contributions

Chapitre 4

Adaptation conservative des cas pratiques à l'analyse

Pour créer un marché il faut inventer un problème, puis trouver sa solution.

— Scott Adams, *Le principe de Dilbert*

Sommaire

Les exemples utilisés dans la littérature scientifique de l'ordonnancement temps réel font souvent référence à des cas académiques ou peu fréquents dans le monde industriel. En effet, comme vu dans le chapitre 2 la théorie du temps réel doit s'adapter à la technologie. Dans ce chapitre, on se propose de décrire des cas rencontrés dans l'industrie et leur adaptation à des modèles analysables. Nous verrons d'abord des cas pour lesquels des analyses ne sont, a priori, pas disponibles. Ensuite, nous verrons une possibilité de mettre en place les adaptations et enfin comment intégrer cela dans un processus d'analyse.

4.1 Contexte industriel et d'analyses

La modélisation des systèmes temps réel devient de plus en plus complexe en raison des modèles dédiés aux domaines transverses comme présenté dans le chapitre 2. Ce faisant, la modélisation, ainsi que l'analyse, prend de plus en plus de temps. Actuellement, le processus entre l'architecture et l'analyse est très souvent assuré par un expert en analyses, celui-ci connaissant les outils ainsi que les manipulations nécessaires pour un cas donné. L'expert en analyse doit mettre en œuvre des modifications sur le modèle d'analyse correspondant au système. Ces modifications sont nécessaires pour que l'analyse puisse être appliquée au modèle.

Exemple Prenons l'exemple d'une transaction ayant deux évènements périodiques d'activation représenté en figure 4.1 par MAST. Ce type de modèle est très fréquent dans les cas industriels, puisqu'il indique que la troisième activité (nommée t3 sur la figure) doit s'activer dès lors que l'une des deux activités en amont des transactions lui fournit des données d'entrée. L'une des deux activités d'entrée pourrait par exemple s'activer périodiquement, l'autre sporadiquement lors de l'arrivée de message sur un réseau. L'activité de droite devrait alors s'activer périodiquement à la suite de l'activité en bas à gauche, mais aussi à la suite de l'activité en haut à gauche, qui s'active à l'arrivée d'un message sur le réseau.

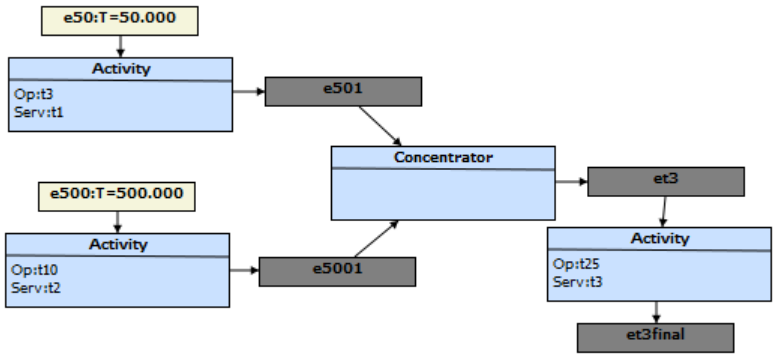


FIGURE 4.1 – Exemple introductif dans MAST

Ce modèle n'est pas analysable par MAST : une exception est levée par le logiciel (`Tool Failure exception. Feasible_Processing_Load not yet implemented for unrestricted multi-path systems`)et l'analyse est arrêtée. Il faut apporter des modifications dans le modèle afin de permettre l'analyse par le logiciel. Ces transformations ne sont connues que par l'expert en analyses et non par l'architecte logiciel dans l'industrie.

Dans ce chapitre, on propose d'abord d'identifier des cas rencontrés dans l'industrie pour lesquels l'analyse n'est pas faisable par les outils existants. Ensuite, on propose de référencer les adaptations de modèle dans la section 4.3 pour faciliter le travail d'analyse. Ce travail d'analyse est retracé dans la section 4.4. Enfin, ce processus est appliqué sur un exemple dans la section 4.5. Selon les processus utilisés, il est possible d'extraire un modèle afin d'analyser le sous-modèle.

4.2 Exemples de situations à adapter

Dans cette section, nous présentons des cas ne pouvant être traités tels quels car à notre connaissance, il n'y a pas de littérature les analysant formellement, et encore moins d'outils d'analyse les traitant directement. On se propose d'abord de les décrire, dans quels contextes ils peuvent être utilisés et nous proposons de les adapter pour les rendre analysables pour les outils existants. Il faut noter que cette adaptation doit être conservative pour éviter de donner

un résultat optimiste. Il s'agira donc de mettre en place une adaptation dont le modèle adapté soit équivalent ou plus pessimiste que le modèle d'origine.

4.2.1 Activation alternative en "OU"

4.2.1.1 Description

L'activation d'une tâche peut être effectuée de plusieurs façons : selon un schéma connu ou bien par d'autres tâches. La figure 4.2 présente un cas simple de précédence en "OU". On y considère 3 tâches communicantes entre elles, la tâche τ_3 est activée soit par la tâche τ_1, soit par la tâche τ_2 par le biais d'un message envoyé à la tâche τ_3. Les tâches τ_1 et τ_2 sont activées de façon sporadique et précédent ainsi la tâche τ_3.

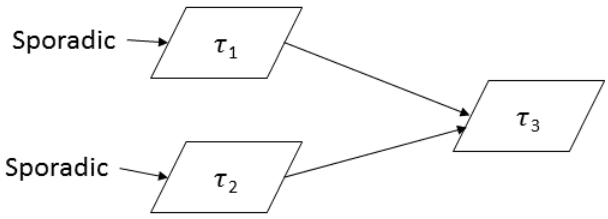


FIGURE 4.2 – Activation alternative d'une tâche par deux autres tâches

A l'heure actuelle, des outils existent pour analyser de tels types de systèmes. Henia *et al.* ont proposé une analyse valide uniquement pour des tâches périodiques [HHJ+05] en proposant une borne supérieure sur la gigue de la tâche activée par alternance. Cette analyse donne une estimation pessimiste car cela crée plus d'occurrences que nécessaire de la tâche τ_3 en reprenant le cas précédent. Cette solution a été implémentée dans l'outil d'analyses SymTA/S.

4.2.1.2 Proposition d'adaptation

On propose ici une autre solution de modélisation permettant à d'autres outils, comme MAST, d'opérer une analyse. Le système de la figure 4.2 est analysable si on peut transformer l'architecture en une autre structure analysable. Dans ce cas de figure, il est difficile de déterminer finement le schéma d'activation de la tâche τ_3. On se propose de dupliquer la tâche τ_3 en deux tâches τ_{3A} et τ_{3B} comme sur la figure 4.3. Cela conduit à une tâche activée uniquement par τ_1 et une tâche activée uniquement par τ_2. Dans le modèle original, si la tâche τ_3 n'est pas réentrante, les nouvelles tâches τ_{3A} et τ_{3B} ne doivent pas se préempter l'une l'autre : il y a ici plusieurs choix de modélisation par l'expert. Par exemple, en monoprocesseur (ou en multiprocesseur partitionné) avec priorités fixes aux tâches, les duplicatas doivent avoir la même priorité, elles sont donc ex-æquo en termes de priorité. Dans la plupart des outils d'analyse basés sur l'analyse de temps de réponse, le cas de chaque tâche ex-æquo est traité de sorte à considérer lors de l'analyse d'une tâche que l'autre tâche se verra toujours plus prioritaire. On introduit ici du pessimisme car on va considérer que le duplicata, s'il est réveillé après la tâche analysée, pourra la préempter, alors que dans la réalité, la requête ne pourra être exécutée qu'après, en FIFO. Si on a une analyse considérant les ex-æquo gérées en FIFO, alors le modèle de la figure 4.3 est suffisant pour capturer le pire cas de chaque tâche. Dans le cas d'un ordonnancement EDF sur un système monoprocesseur (ou multiprocesseur partitionné), la priorité étant donnée par l'échéance absolue, et l'échéance relative de chaque duplicata étant la même, la prise en compte FIFO des duplicatas est assurée. Enfin, en multiprocesseur global, il faut assurer que les duplicatas ne puissent s'exécuter en même temps, par conséquent, on est amenés à les mettre en exclusion mutuelle dans le modèle.

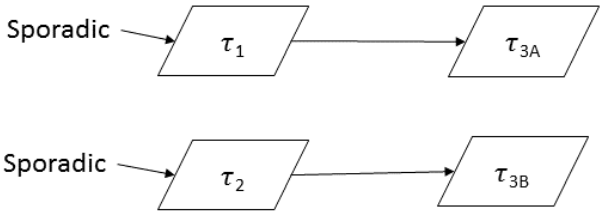


FIGURE 4.3 – Résultat de transformation de l'exemple en figure 4.2

De façon générale, cette transformation est valide pour beaucoup de contextes et est indépendante du support qu'il soit monoprocesseur ou multiprocesseur. Il faut que les deux nouvelles tâches ne s'exécutent pas en même temps et qu'elles soient exécutées sur la même partition si l'ordonnanceur est partitionné. Si l'ordonnanceur est basé sur des priorités fixes, il faut aussi que la priorité des nouvelles tâches correspondent à la priorité de la tâche originale pour refléter le système original.

4.2.2 Systèmes temps-réel probabiliste

4.2.2.1 Description

L'analyse temps réel probabiliste permet de mieux rapprocher l'analyse temps réel du système réel [CGGM14]. Diverses distributions existent, en particulier pour le temps d'exécution. La loi de Weibull est souvent utilisée de façon semblable à la figure 4.4. Néanmoins peu d'outils existent pour calculer le pire temps de réponse d'un système possédant des tâches à durée probabiliste. Si un système utilisant ce type de durées d'exécutions est décrit, il est possible aussi que ce système contienne d'autres éléments comme des exclusions mutuelles, ou des systèmes distribués.

Des logiciels existent pour prendre en compte l'aspect probabiliste tels que PRISM [KNP11] ou ROMEO [GLMR05]. En revanche, ces logiciels se basent sur l'utilisation des méthodes formelles pour la verification des modèles : à cause de l'explosion combinatoire ces méthodes ne passent pas à l'échelle surtout pour les grands cas industriels.

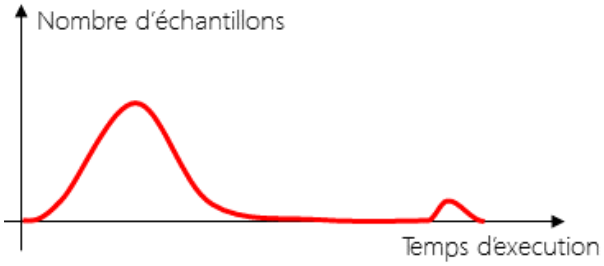


FIGURE 4.4 – Distribution probabiliste du temps d'exécution

4.2.2.2 Proposition d'adaptation

Dans le cadre d'une analyse non probabiliste, pour le cas du temps d'exécution exprimé sous forme probabiliste, on gardera ainsi la valeur maximale pour proposer le pire cas avec la plus grande charge processeur. Plus généralement, lorsqu'une valeur est exprimée de façon probabiliste, la valeur la plus défavorable pour le système de tâches sera choisie.

Par la suite, après l'adaptation, il faut vérifier que l'analyse qui sera effectuée soit viable par rapport au temps d'exécution. En effet, comme le temps d'exécution peut prendre des valeurs inférieures au WCET, il faut que l'analyse soit viable par rapport à celle-ci.

4.2.3 Tâches activées par chien de garde

4.2.3.1 Description

Les tâches activées par chien de garde (ou *watchdog*) ont une activation particulière. Un *watchdog* est une alarme qui doit être réinitialisée régulièrement avant un temps limite w. Une ou plusieurs tâches peuvent réinitialiser le *watchdog*. Dans le cas où il n'est pas réinitialisé, le *watchdog* s'exécute et fait appel à une tâche *handler*, une tâche exécutant des instructions appropriées pour la sauvegarde du système par exemple. En effet, dans ce cas, les tâches qui devaient réinitialiser ne se sont pas exécutés.

Ce type de fonctionnement est surtout utilisé dans des cas pour lesquels l'utilisation informatique est critique et pour lequel le système doit toujours être capable de fonctionner. Par exemple, lors du fonctionnement en mode dégradé d'un commutateur, la résolution d'erreurs interne, faire sonner une alarme, etc... Dans le cas du rover Mars Pathfinder, un *watchdog* redémarrait le système car il observait des échéances qui n'étaient pas respectées, ceci étant dû à une inversion de priorités. Le blocage avait pour conséquence de ne pas réinitialiser le *watchdog*.

4.2.3.2 Proposition d'adaptation

Il n'est pas possible de déterminer précisément quand le *watchdog* va se déclencher étant donné que cette tâche ne doit pas s'activer en temps normal. Pour rendre un système avec watchdog analysable, le watcdog est assimilé à une tâche sporadique activée au minimum tous les w unités de temps. Pour cela, il faut supposer que l'analyse est viable par rapport à la période d'exécution de la nouvelle tâche sporadique : le système doit être ordonnançable avec la tâche du watchdog s'exécutant moins fréquemment que lors de l'analyse.

4.2.4 Fenêtre glissante d'activations

4.2.4.1 Description

Une fenêtre glissante d'activation est un schéma particulier d'activation. On définit une fenêtre glissante de taille w_i, pour tout instant t, comme étant l'intervalle ouvert $[t, t + w_i)$. Une tâche τ_i est activée dans une fenêtre glissante de telle sorte que pour toute fenêtre de temps de largeur w_i, il y ait au plus n_i activations de la tâche τ_i. Il est à noter que cela n'exclut pas d'avoir plus de n_i instances inachevées dans le processus. Dans le cas particulier où la tâche est périodique, on a alors w_i égal à la période de la tâche et $n_i = 1$.

Ce modèle d'activation est rencontré dans l'industrie mais aucun outil d'analyse n'existe avec une activation de ce type.

4.2.4.2 Proposition d'adaptation

On considère un ensemble de tâches constitué de n tâches $\tau_{i,i=1...n}$ avec pour chaque tâche, le pire temps d'exécution C_i et une fenêtre d'activation définie avec au plus n_i activations sur une fenêtre de taille w_i.

Néanmoins, dans ce contexte il est possible de définir la RBF pour représenter le pire schéma d'activation de tâches. Il est possible de construire une fonction enveloppant l'ensemble des schémas d'activations possibles par une fenêtre glissante d'activations. Cette fonction est telle qu'il y ait une rafale d'activation simultanée des n_i instances avec une période égale à w_i, en commençant à $t = 0$. Cela montre que dans un contexte où la RBF peut être utilisée pour l'analyse, une fenêtre d'activation peut être représentée avec des activations sporadiques.

Nous nous plaçons dans un contexte à priorités fixes aux tâches. On représente sur la figure 4.5 deux RBF :
- la première, en pointillés, représente la RBF pour un schéma d'activations en fenêtre glissante,

- une seconde en bleue, en continu, représente la RBF pour une activation simultanée des instances des tâches.

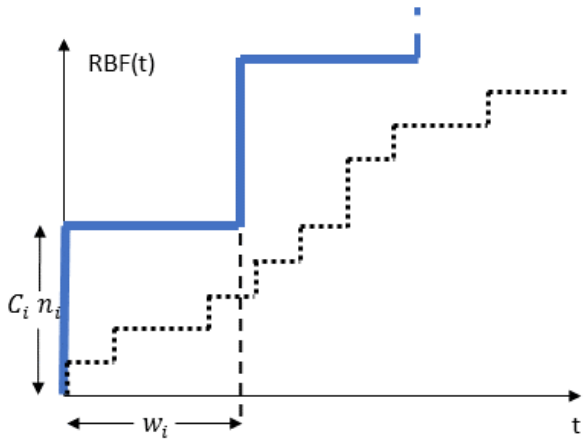


FIGURE 4.5 – RBF d'une fenêtre glissante

On a considéré pour cette figure $n_i = 5$. Une "marche" de la RBF en pointillés étant donc de hauteur C_i et une "marche" de la courbe continue est de hauteur C_i n_i. Il n'est pas possible que la fenêtre glissante puisse dépasser la courbe bleue sauf si la tâche s'exécute plus de fois dans la fenêtre de temps attribuée ce qui est impossible.

Ce cas de figure représente donc le pire cas de demande processeur pour ce schéma d'activation.

Il manque toutefois quelques éléments de contexte pour appliquer la transformation pour ce système :
- L'analyse doit être viable par rapport à la durée d'exécution ;
- L'analyse doit être viable par rapport à la période de la tâche ;
- L'analyse doit être viable par rapport à la date d'activation : il faut que le système puisse fonctionner malgré l'absence d'un instant critique.

Pour adapter le principe d'activation de fenêtre glissante de n_i activations sur toute fenêtre w_i, on l'adapte ainsi en une vague de n_i activations espacées de w_i unités de temps.

4.3 Explicitation et modélisation de l'adaptation conservative : CONSERT

Le processus de l'adaptation dirigée par des transformations endogènes est fortement lié au savoir-faire de l'analyste. On propose ici de stocker ces transformations dans un référentiel pour pouvoir les partager avec les autres utilisateurs (les architectes). Ce référentiel de transformations peut ainsi jouer un rôle décisionnel et rendre l'utilisateur autonome et indépendant de l'expert en analyses.

4.3.1 CONSERT - CONServative Endogenous Repository based Transformations

On utilise le principe de référentiel d'analyses proposée dans MoSaRT [Ouh13] pour proposer un référentiel de transformations : *CONServative Endogenous Repository based Transformations* (CONSERT). Il s'agit de déterminer et d'appliquer de façon automatique les transformations nécessaires pour un système donné basé sur l'expertise en analyse et architecture logicielle des

ingénieurs. CONSERT permet de sauvegarder les hypothèses sur les modèles afin d'appliquer les transformations appropriées pour obtenir un modèle analysable qui servira ensuite de base pour les outils d'analyse d'ordonnancement. Ce référentiel permet également d'aider les architectes logiciel en gardant la traçabilité des changements avant analyse afin de remonter au modèle original.

Les contextes de transformations représentent les hypothèses requises sur un modèle avant transformation. Par exemple, la transformation depuis une fenêtre glissante d'activation vers un modèle de tâche sporadique ne peut s'exécuter que si toutes les instances sont localisées sur le même processeur. En effet, dans le cas contraire, cette transformation ne serait pas valide, les instances pouvant alors s'exécuter parallèlement.

On définit ainsi un contexte $\mathcal{C}$ comme étant un ensemble d'hypothèses $\{a_1, a_2, ..., a_n\}$. Une hypothèse a_i étant, par exemple "l'architecture matérielle est monoprocesseur", ou bien, pour le cas précédent, "les instances des tâches activées par une fenêtre glissante sont localisées sur un même processeur".

Chaque hypothèse a_i est ainsi un ensemble de propositions pouvant être exprimées par des contraintes sur le méta-modèle, modèle devant respecter, ou pas, l'ensemble de propositions.

Les contextes présentés précédemment ont un ensemble d'hypothèses :

— Contexte d'activation alternative en "ou" $\mathcal{C}_A$:
  — Il existe au moins une tâche précédée par deux ou plus de tâches et activée en "OU",
— Contexte probabiliste pour le temps d'exécution $\mathcal{C}_B$:
  — Le temps d'exécution est défini de façon probabiliste.
— Contexte d'activation par chien de garde $\mathcal{C}_C$:
  — Il existe au moins une tâche activée par un chien de garde.
— Contexte d'activation par fenêtre d'activation $\mathcal{C}_D$:
  — Au moins une tâche est activée par une fenêtre d'activation
  — Toutes les instances sont localisées sur un seul processeur
  — La(Les) tâche(s) concernée(s) ne se suspendent pas d'elle-même(s)

Ces ensemble d'hypothèses permettent la détection des contextes par CONSERT. Néanmoins, après la transformation, il faut également prendre en compte les conditions de validité de l'adaptation du modèle. Par exemple, il est nécessaire de connaître les conditions de viabilité d'une analyse.

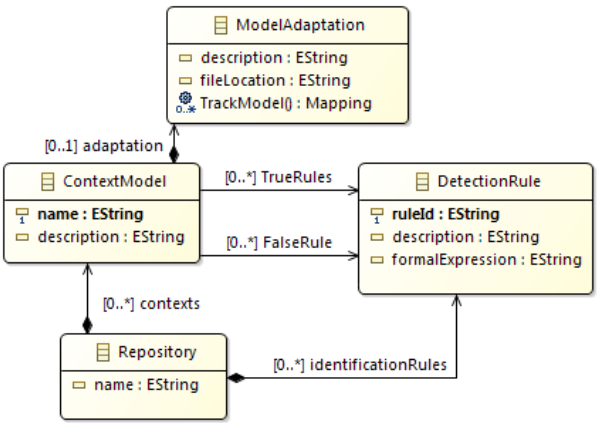


FIGURE 4.6 – Extrait du méta-modèle CONSERT

Pour stocker les transformations, il faut répertorier les contextes. On propose ainsi le méta-modèle proposé dans la figure 4.6. La figure est exprimée dans le langage Ecore. La classe racine est `Repository` où chaque instance est composée d'un ensemble de règles de détection (`DetectionRule`) et d'un ensemble de contextes de transformations `ContextModel`. Chaque règle de détection est caractérisée par un identifiant `id`, une description (`description`) de la règle et une expression formelle (`formalExpression`). Une expression formelle est la "traduction" dans le langage de modélisation dans lequel il faut appliquer une transformation. Par exemple, dans le domaine du temps réel, pour un système de tâches $sys = \{\tau_1, \tau_2, ..., \tau_n\}$. Chaque tâche τ_i est caractérisée par une période T_i et une échéance relative D_i telle que $\tau_i :=<T_i, D_i>$. Alors, l'expression formelle d'une règle de détection vérifiant si les échéances relatives sont contraintes correspondrait à $\forall\ \tau_i \in sys$, $T_i \leq D_i$.

Le contexte spécifie également les règles qui doivent être vraies (`TrueRules` sur la figure) et les règles qui doivent être fausses (`FalseRules` sur la figure) pour valider le contexte. A chaque contexte, on peut ainsi faire correspondre une adaptation de modèle (`ModelAdaptation`) décrivant les transformations à appliquer dans le modèle. L'adresse du fichier de transformation, qu'il soit en ATL ou Java ou un autre langage, sera décrit dans le paramètre `fileLocation`. La fonction `trackModel()` permet de déclencher la traçabilité des transformations afin de faire correspondre les classes ajoutés par la transformation au modèle original. La traçabilité des transformations est expliquée dans la section 4.3.3.

Le référentiel CONSERT ne dépend pas d'un langage de modélisation particulier et peut ainsi être utilisable avec plusieurs langages. Dans la suite, nous intégrerons CONSERT dans Time4Sys, un langage basé sur des artefacts issus de Tempo (sous-profil de UML-MARTE) et le langage MoSaRT.

4.3.2 Intégration de CONSERT dans Time4Sys

La figure 4.7 expose un exemple d'instance du référentiel de transformation. Le référentiel, ainsi construit, est constitué de contextes de transformations endogènes. Ces transformations se réfèrent ainsi aux règles de détection de contexte pour déterminer le(s) bon(s) contexte(s).

On résume dans le tableau 4.1 les transformations à appliquer pour chaque cas identifié précédemment et lorsque le cas se présente une représentation graphique dans Time4Sys.

4.3.2.1 Choix techniques pour l'implémentation

L'utilisation de CONSERT avec un langage de modélisation requiert les choix techniques suivants : un langage pour exprimer les règles de détection et un langage de transformation supportant le langage de modélisation. Time4sys est un langage dont le méta-modèle est basé sur Ecore. Comme OCL est compatible Ecore, nous avons choisi ce langage pour exprimer simplement les règles de détection sous forme d'invariants. Nous avons également choisi ATL pour implémenter la transformation nécessaire pour chaque contexte car celui-ci s'appuie sur le langage OCL. Bien entendu, d'autres langages, comme QVT (Query/View/Transformation), peuvent aussi être utilisés vu qu'ils supportent Ecore.

L'adaptation est faite de telle sorte que l'algorithme de transformation (par exemple implémenté sous forme de code ATL comme illustré par le listing 4.1) puisse être utilisé pour la transformation. Cette partie algorithmique est utilisée avec la fonction de raffinement d'ATL permettant de faire des transformations endogène de modèle. Comme ce code s'applique pour Time4Sys, l'exemple du listing 4.1 montre que si on détecte une activation par une fenêtre glissante, on applique une transformation concernant la classe `SlidingWindowPattern` en `Burst-Pattern`, une rafale d'activation. Le nouvel élément prend alors la même référence, la période

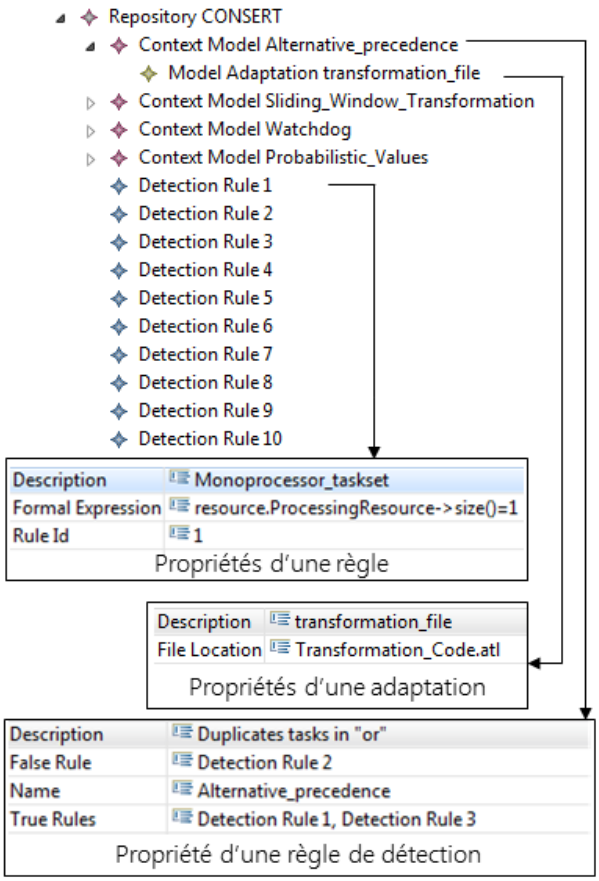


FIGURE 4.7 – Instance du référentiel de transformation

d'arrivée correspond à la taille de la fenêtre et la taille de la rafale correspond au nombre maximum d'activations dans la fenêtre.

Listing 4.1 – Extrait du code ATL pour un contexte d'activation par fenêtre glissante $\mathcal{C}_D$

```
lazy rule SlidingWindow{
   from s : Time4SysModelNFP!SlidingWindowPattern (true)
   to t : Time4SysModelNFP!BurstPattern{
      reference <- s.reference,
      minInterarrival <- thisModule.NFP_Duration(s.windowSize),
      burstSize <- s.nbEvents
   }
}
```

4.3.2.2 Processus de détection des contextes

Notons également que pour lier les instances de CONSERT aux modèles (instances de langages de modélisation) un processus de détection de contexte a été implémenté. Ce processus de détection s'occupe de faire appel aux règles de détection et vérifie une à une si elle est satisfaite par le modèle (soumis à l'adaptation) ou pas. Suite à cela, un contexte d'adaptation peut être soulevé. Dans ce cas, le processus de détection exécute le script de transformation lié au contexte afin d'obtenir un modèle adapté. Dans les cas que nous avons traités dans cette thèse, il s'avère que les transformations sont indépendantes les unes des autres n'induisant ainsi aucun problème d'orchestration d'adaptation. Le processus de détection est résumé dans l'Algorithme 1.

Le processus de détection s'occupe également de produire la trace de transformation comme cela sera expliqué par la suite. Le processus de détection est générique (fonctionnera avec tout langage de modélisation supportant CONSERT) et a été développé en utilisant Java.

Algorithme 1 Détection et transformation

Entrée: Modèle {modèle à adapter}
Entrée: RéférentielAnalyses
Sortie: Modèle transformé {modèle adapté}
  $transformations \leftarrow \emptyset$;
  $trueRules \leftarrow \emptyset$;
  $falseRules \leftarrow \emptyset$;
  Étape 1 : Trouver les règles vérifiées
  pour *rule* dans RéférentielAnalyses.identificationRules **faire**
    si *rule* est vérifié sur Modèle **alors**
      $trueRules \leftarrow trueRules \cup r$;
    sinon
      $falseRules \leftarrow falseRules \cup r$;
    fin si
  fin pour

  Étape 2 : Trouver les contextes
  pour *contexte* dans $Référentiel Analyses.contexts$ **faire**
    si [($contexte.trueRules \subset trueRules$) ET ($contexte.falseRules \subset falseRules$)] **alors**
      $transformations \leftarrow transformations \cup contexte$;
    fin si
  fin pour
  Executer($transformations$) ;

4.3.3 Traçabilité des adaptations

Très souvent, lors des transformations de modèles, il est nécessaire de garder une trace des transformations menées sur le modèle. A moins de garder tous les modèles en mémoire, il n'est pas possible de revenir sur les modèles précédents. Or, lors de transformations endogènes, le concepteur doit pouvoir interpréter les éléments ajoutés qui ne sont que des artéfacts de modélisation pour l'analyse. On propose ainsi un méta-modèle permettant de créer une trace des transformations apportés sur les systèmes et ainsi remonter au modèle original.

La figure 4.8 montre le méta-modèle que les instances de traces doivent respecter. Une trace se compose donc de la classe `Mapping` représentant les liens entre les objets, instances de `MappableArtefact`. On peut définir ainsi les instances sources et cibles, avant et après la transformation. Ces instances sont localisées dans le modèle défini par `ResourceArtefact`, hérité de `MappableArtefact`. De même, le modèle source et cibles sont liées entre elles. Les modèles sont définies par un Uniform Resource Identifier (URI) et par une ressource racine du modèle Ecore `Resource`.

Ces liens sont associés à des règles de transformations `Rule`. Ces règles sont dépendantes du référentiel de transformations : il s'agira d'indiquer la raison pour laquelle la transformation a été effectuée c'est-à-dire le contexte de la transformation ainsi que la description de la transformation.

Ce méta-modèle permet de proposer un exemple de trace de transformation dans la figure 4.9. Un lien `Mapping` va récupérer deux `ResourceArtefact` pour définir les modèles d'origine et transformé et pour retrouver ces modèles l'URI est également spécifié. Chaque lien `Link` fait

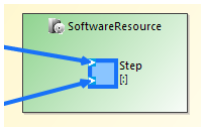
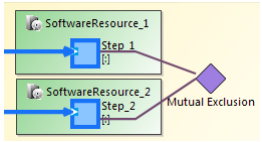
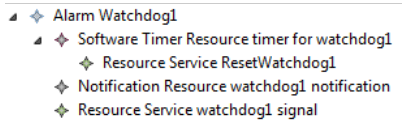
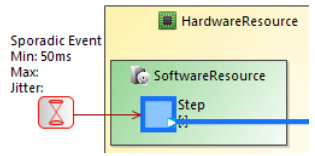
Classe Time4Sys	Avant Transformation	Après Transformation
Contexte Précédence Alternative en "OU" $\mathcal{C}_A$		
`PrecedenceRelation`	Connecteur dont la propriété est *Merge*	Connecteurs dont la propriété est *Sequence*
`Step (i.e. fonction)`	Une étape avec deux précédences ou plus	Une étape dupliquée pour autant de précédences
`SoftwareSchedulableResource`	Une tâche	Autant de tâches que de précédences
`MutualExclusion`	—	Une exclusion mutuelle entre les nouvelles instances de tâches
Représentation Time4Sys		
Contexte probabiliste $\mathcal{C}_B$		
`ProbabilisticDuration` (hérité de `NFP_Duration` & `NFP_TimeInterval`)	Distribution probabilistique	—
`Duration` (pour le temps d'execution)	Valeurs probabilistes	Égal à la valeur maximale
`Duration` (pour les dates d'arrivées)	Valeurs probabilistes	Égal à la valeur minimale
Contexte de tâche activée par Watchdog $\mathcal{C}_C$		
`Alarm`	Modélisée comme watchdog	—
`SoftwareSchedulableResource`	Taskset	Création d'une tâche Watchdog
`ActivationPattern`	—	Création d'une activation sporadique associée à la tâche watchdog
Représentation Time4Sys		
Contexte d'activation en fenêtre glissante $\mathcal{C}_D$		
`SlidingWindowPattern` (hérité de `ActivationPattern`)	n_i activations au maximum dans toute fenêtre de taille w_i	Inexistant
`BurstPattern`	Inexistant	Rafale de n_i activations toutes les w_i unités de temps

TABLEAU 4.1 – Résumé des transformations dans Time4Sys

donc la relation entre les `MappableArtefact` d'origine et transformé et indique également par la propriété `Rationale` la règle utilisée pour cette transformation. Dans ce cas, la règle utilisée sera une transformation liée à une activation alternative de tâche. Les instances de modèle non transformées ne sont pas déclarées permettant ainsi d'alléger le modèle.

Le modèle de trace est généré par l'application après une transformation endogène.

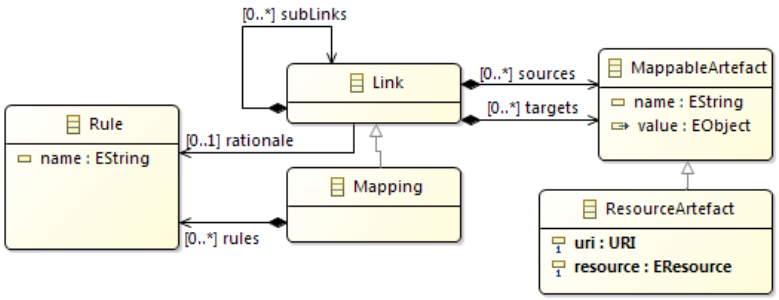


FIGURE 4.8 – Méta-modèle de traçabilité

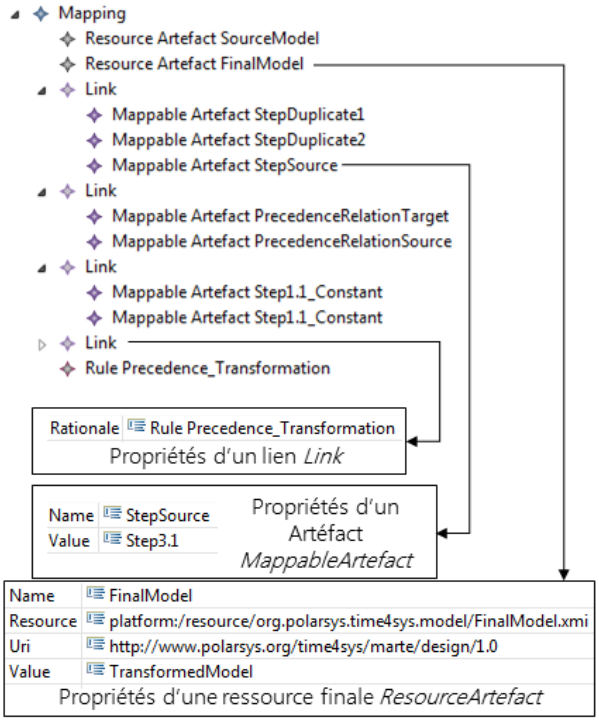


FIGURE 4.9 – Instance de trace

4.4 Intégration de CONSERT dans le processus d'analyse

L'application CONSERT est ainsi un référentiel d'analyses avec une génération automatique de trace afin de garder une trace des transformations. Cette application indépendante du langage peut être connectée avec des langages existants pour mettre en place un processus d'analyse adapté à l'analyse temps réel. L'existence d'un référentiel d'analyses permet de déterminer les analyses et les outils adaptés au modèle considéré. Dans cette partie, on présente la place de CONSERT dans un processus plus large d'analyses temps-réel ainsi que les relations possibles. Par la suite, on présentera le processus complet d'analyses que l'on peut appliquer à un modèle.

4.4.1 Structure globale du système

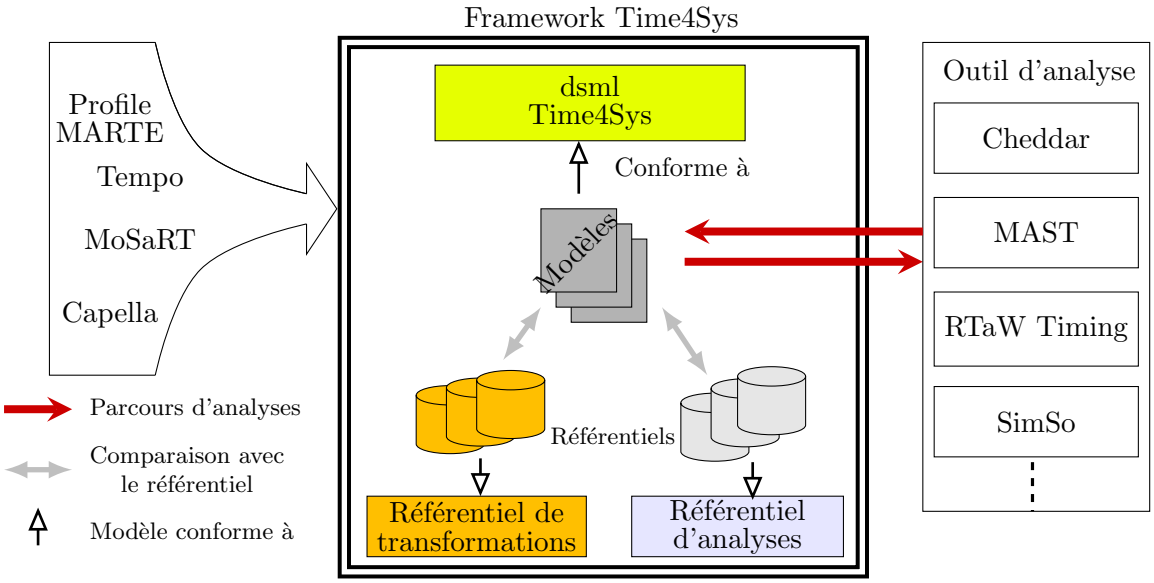


FIGURE 4.10 – Synthèse de Time4Sys

Le framework *Time4Sys* est inspiré de multiples concepts. La figure 4.10 montre le framework *Time4Sys* ajouté du référentiel de transformations CONSERT en orange. La modélisation est basée sur UML - MARTE et le concept général sur le framework MoSaRT. Le principe de modélisation Capella, avec la méthodologie Arcadia, permet de modéliser les systèmes de façon descendante. Le point de vue Tempo [HRS13] permet de préparer la transformation de modèles et se place avant l'analyse. Time4Sys est ainsi constitué de ces principes et modèles afin de constituer un pont entre l'architecture et l'analyse.

Avant d'analyser un modèle, les référentiels sont d'abord construits par un expert en analyses qui peut être l'utilisateur lui-même. Les modèles construits sont conformes au langage Time4Sys. Des transformations de modèles sont disponibles entre le modèle Time4Sys et les modèles d'analyses tels que Cheddar, MAST ou SimSo [CHD14].

4.4.2 La chaîne d'analyses

Il s'agit maintenant d'interconnecter les éléments les uns aux autres. On présente dans la figure 4.11 le processus complet de Time4Sys que l'on propose. Le modèle peut être importé depuis un autre langage de modélisation (Étape (0) sur la figure). Une fois le modèle transcrit sous le langage Time4Sys et tous les paramètres temporels renseignés, le modèle est soumis au référentiel d'analyses pour déterminer la meilleure analyse possible pour le modèle donné (en (1) dans la figure). Dans le cas où l'outil d'analyse proposé n'est pas satisfaisant (Étape (2)) – dans le cas d'un outil payant par exemple–, on peut proposer une transformation endogène qui peut ainsi être adapté aux exigences demandées par l'architecte logiciel. Le modèle est comparé au référentiel de transformations et ainsi le modèle peut être adapté le cas échéant (Étape (3)). Le modèle adapté peut ainsi être comparé une nouvelle fois avec le référentiel d'analyses pour déterminer l'outil d'analyses pour ce modèle (Étape (4)). Lorsque l'analyse proposée est satisfaisante, le référentiel permet la transformation vers cet outil d'analyses (Étape (5)). Après analyses, il est possible de récupérer les résultats (Étape (6)) et ainsi raffiner le modèle avec les les analyses (Étape (7)).

Dans la section suivante, on présentera un cas d'études traité en globalité dans le cadre d'un processus d'analyses au complet.

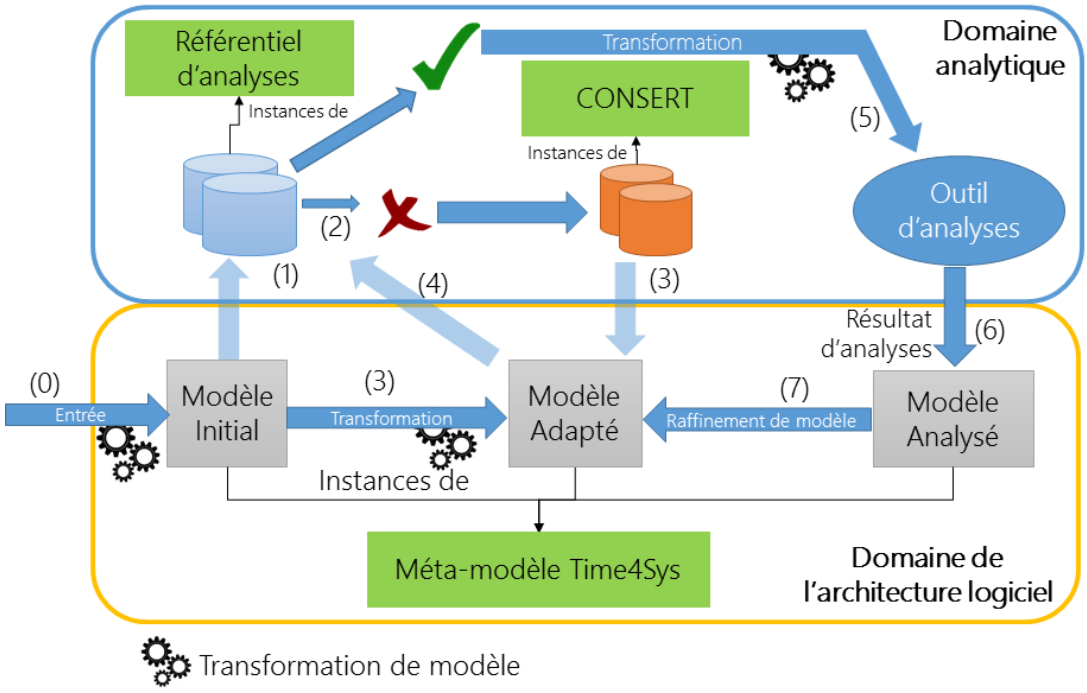


FIGURE 4.11 – Processus de CONSERT intégré dans Time4Sys

4.5 Mise en Application

Dans cette section, on se propose d'appliquer le processus proposé. Il s'agira de représenter le système dans Time4Sys et ensuite analyser le système à l'aide du référentiel CONSERT.

4.5.1 Présentation du cas d'étude

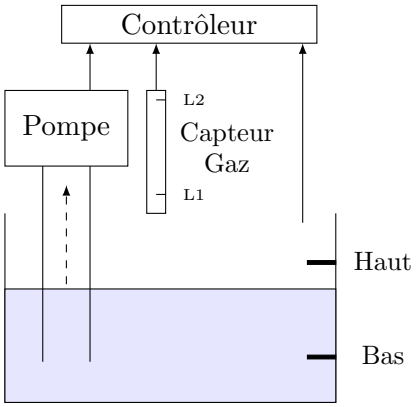


FIGURE 4.12 – Cas d'études d'une pompe à eau

La figure 4.12 présente le schéma du cas étudié, cité pour la première fois par Mahony [MH92]. On considère une mine pouvant se remplir d'eau. Le niveau d'eau doit être maintenu entre les niveaux "Haut" et "Bas". Si le niveau d'eau passe au dessus du niveau "Haut", la pompe doit fonctionner, et si elle passe en dessous du niveau "Bas", la pompe doit s'arrêter pour éviter son dysfonctionnement. Si la quantité de méthane présente dans la mine est supérieure à un niveau $L1$ défini, une alarme doit être activée pour évacuation du personnel dans la mine. Si la quantité de méthane dépasse le niveau $L2$, la pompe doit s'arrêter pour éviter une explosion.

Ces évènements sont gérés par un contrôleur monoprocesseur.

4.5.2 Modélisation sous Time4Sys

Le système informatique est modélisé sous Time4Sys dans la figure 4.13.

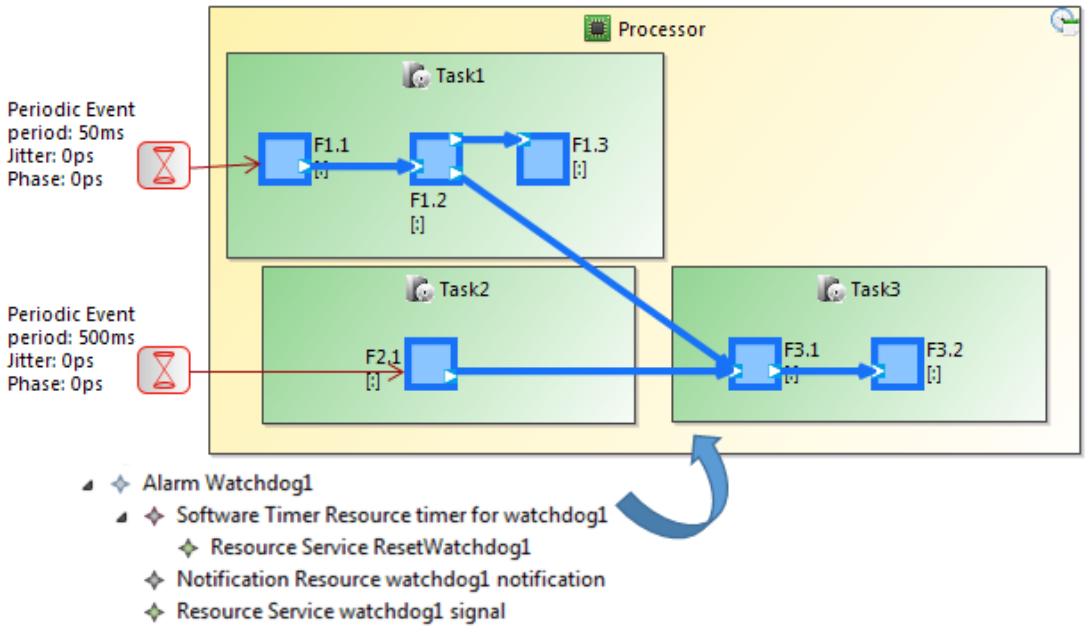


FIGURE 4.13 – Modèle Time4Sys du système

La tâche 1 s'occupe de récupérer les informations du capteur de méthane et traite le niveau d'alerte à transmettre à la tâche 3. La tâche 2 récupère les informations des capteurs de niveau d'eau et les transmet à la tâche 3. Les tâches sont localisées sur le même processeur.

La tâche 1, activée toutes les $50ms$ est composée de trois fonctions exécutées en série et la deuxième fonction, une fois terminée, active la tâche 3. La tâche 2 est activée toutes les $500ms$, en supposant que le niveau d'eau varie peu rapidement, et active la tâche 3 une fois terminée. Afin de pallier les défaillances éventuelles de capteur, un *watchdog* est modélisé sous forme d'alarme avec un minuteur de $51ms$ qui doit être remis à zéro par la tâche 1 régulièrement. Si la tâche 1 est bloquée, le *watchdog* active la tâche 3 et active l'alarme.

Les périodes des tâches ainsi que leur temps d'exécution considéré figurent dans le tableau 4.2. Les priorités y sont fixes et plus la valeur est proche de 1, plus la tâche est prioritaire.

Tâche	T_i	C_i	Priorité
T_1	$50ms$	$10ms$	15
T_2	$500ms$	$10ms$	13
T_3	—	$10ms$	5
Watchdog	—	$2ms$	1

TABLEAU 4.2 – Paramètres du système

Une première analyse par le logiciel MAST, ne permet pas de déterminer un résultat pour le pire temps de réponse, le logiciel rendant une exception, celle-ci semblable à celle de la section 4.1. Il est donc nécessaire de faire une transformation pour pouvoir analyser ce modèle. Une alarme est présente dans le système et est ainsi détectée par le CONSERT.

Une première analyse par le référentiel CONSERT repère cette alarme et transforme l'alarme en une tâche classique activée de façon sporadique. Cette nouvelle tâche active la tâche 3 et cette tâche est maintenant activée soit par le *watchdog*, par la tâche 1, par la tâche 2, situation représentée sur la figure 4.14.

Ensuite, un second contexte de précédence alternative est détecté : il existe un *step* ayant trois précédences, le *step* F3.1. Dans la sémantique de Time4Sys, cela se représente par un port d'entrée sur la fonction comprenant trois antécédences.

Une transformation est donc effectuée pour dupliquer la fonction F3.1 ainsi que la tâche 3. Une exclusion mutuelle entre les tâches est également à ajouter dans le système. Étant donné que la fonction F3.2 comprend ainsi trois précédences, cette fonction est également dupliquée.

Cela donne le modèle adapté de la figure 4.15.

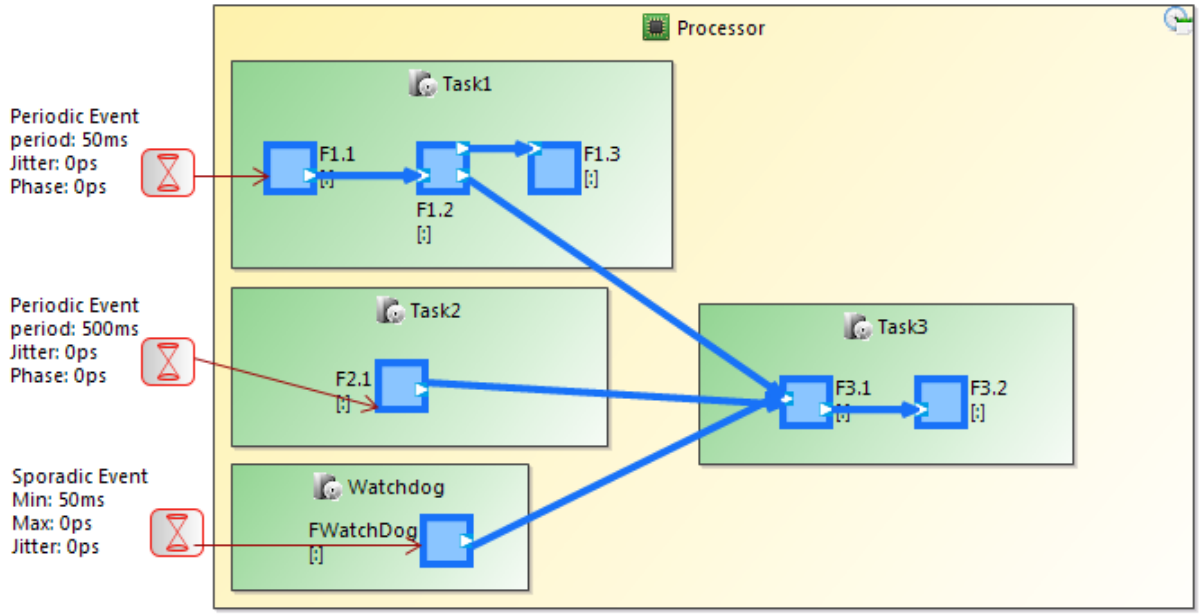


FIGURE 4.14 – Modèle intermédiaire

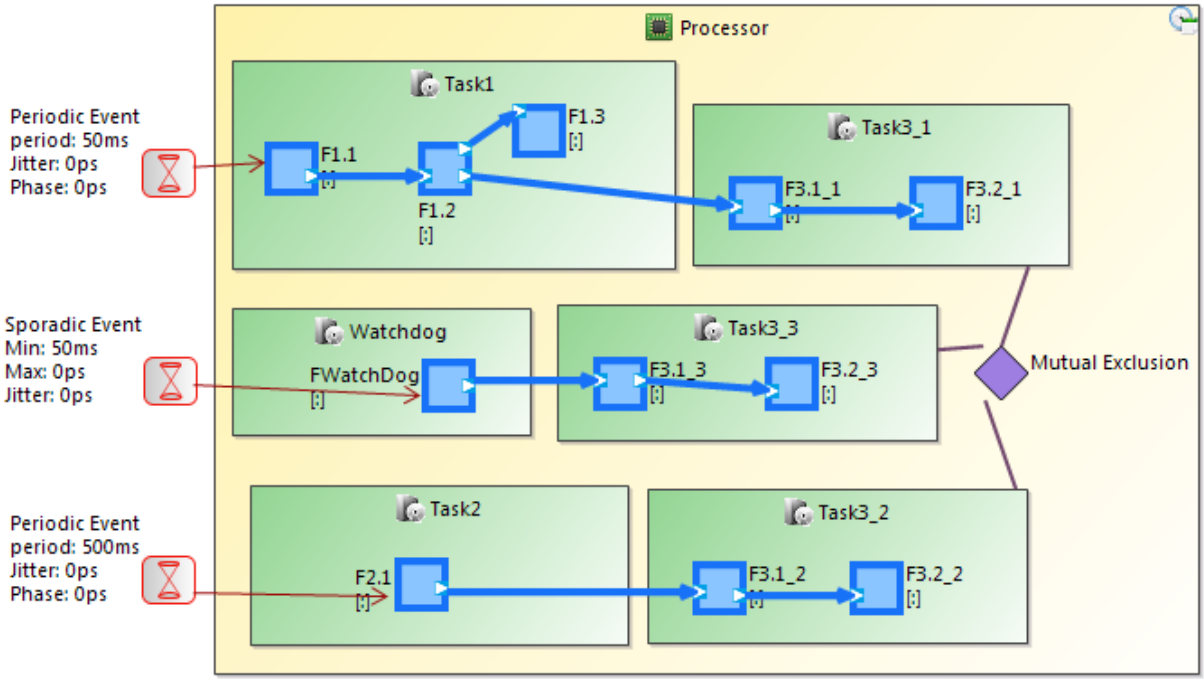


FIGURE 4.15 – Modèle final et adapté à MAST

On peut réutiliser le logiciel MAST pour faire une analyse sur les transactions. Le modèle est donc transposé dans MAST pour donner les résultats dans le tableau 4.3. Les résultats sont

synthétisées dans deux cas : un premier cas dans lequel le *watchdog* ne s'active jamais (Cas 1), et un second cas pour lequel le *watchdog* est considéré comme une tâche sporadique. Les pires temps de réponse s'entendent par rapport à l'évènement d'activation de la tâche 1, la tâche 2 ou de la tâche *Watchdog*.

Dans le pire cas, l'alarme est donnée au plus tard $96ms$ après l'activation du *watchdog*. Dans le cas où le *watchdog* ne s'exécute pas, l'alerte est donnée dans le pire cas, après $235ms$. Ces valeurs sont ensuite à comparer au cahier des charges du système.

Tâche	Pire temps de réponse	
	Cas 1	**Cas 2**
Tâche1	$135ms$	$339ms$
Tâche2	$10ms$	$12ms$
Tâche3 (activation par T_1)	$100ms$	$235ms$
Tâche3 (activation par T_2)	$60ms$	$138ms$
Watchdog	—	$2ms$
Tâche3 (activation par watchdog)	—	$96ms$

TABLEAU 4.3 – Résultats d'analyse par MAST

4.6 Conclusion

Nous avons présenté dans ce chapitre une méthode d'adaptation des modèles à l'analyse. Cette méthode provient de la difficulté à analyser certaines configurations des systèmes industriels. En effet, une adaptation s'avérait indispensable pour évaluer certains modèles. Or, l'adaptation est orientée par l'expertise de l'analyste pour assurer une adaptation qui ne soit pas plus optimiste que le modèle original. Ainsi, CONSERT permet d'agréger les connaissances en analyses pour rendre les architectes indépendants et autonomes dans l'analyse temporelle. CONSERT permet ainsi de conduire des transformations endogènes, stockées dans celle-ci, et de les appliquer de façon automatique. La présentation des résultats à l'ingénieur permet de vérifier la transformation. La transformation est ensuite gardée en mémoire à l'aide d'une trace pour interpréter les éléments ajoutés.

Avec la contribution présentée dans ce chapitre, il est ainsi possible de mettre en place une adaptation conservatrice du pessimisme d'analyse.

Chapitre 5

Les réseaux dans les analyses temps réel

Il faut modéliser pour vivre, et non pas vivre pour modéliser.

— Anonyme, inspiré de Molière, *L'Avare* III, 5, 1668

Sommaire

Ce chapitre présente d'abord les artéfacts actuels du réseau sur les modèles de conception existants. Précédemment dans le chapitre 3, on a présenté des langages de conception permettant la modélisation des systèmes temps réel. La modélisation des réseaux est présentée dans ce chapitre ainsi que les concepts réseaux nécessaires à l'analyse temporelle des réseaux. Une extension de méta-modèle supportant les réseaux est présentée et son utilisation est illustrée à travers un exemple.

5.1 Introduction

L'ingénierie dirigée par les modèles utilise les modèles de conception afin de réduire le temps de développement de ces systèmes par génération de code ou bien en transformant les modèles vers divers outils par exemple. Au cours des dernières décennies, cette méthode est utilisée dans le cadre de l'analyse temporelle des systèmes. Les modèles existants ont alors été réutilisés pour conduire les analyses. Étant donné que ces langages furent orientés architecture logicielle, ils ne peuvent représenter la temporalité des systèmes en particulier dans le cas de l'ordonnancement distribué. En effet, les langages ne sont pas assez expressifs introduisant ainsi un pessimisme dans les résultats d'analyse. Il s'agit donc ici de détailler les méta-modèles réseau pour ainsi adapter les langages de modélisation aux analyses.

La section 5.2 présente les représentations des réseaux dans différents langages. La section 5.3 présente les concepts que l'on peut retrouver dans tous les réseaux ainsi que de façon spécifique pour certains types de réseaux. La section 5.4 propose une modélisation indépendante des langages de conception qui sera ensuite appliquée dans la section 5.5.

5.2 Artéfacts du réseau sur les méta-modèles actuels

Les langages à l'heure actuelle ne permettent pas la modélisation de réseaux en vue de leur analyse. Dans cette section sont présentées les particularités que proposent chaque langage de conception pour les systèmes temps réel.

5.2.1 Exemple introductif

Pour montrer le manque de représentation des modèles, un exemple est présenté sur la figure 5.1 nous allons considérer deux tâches hébergées chacune sur un processus dans deux systèmes différents. Les processeurs peuvent communiquer via un réseau pour l'heure inconnue.

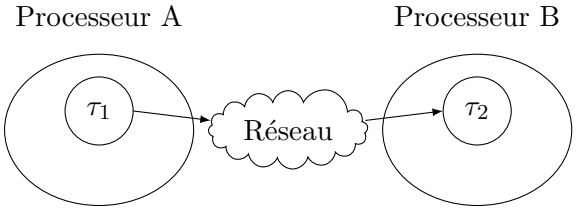


FIGURE 5.1 – Exemple introductif

Dans la suite, cet exemple est modélisé dans les langages de modélisation orientés temps réel. Des limites propres à chaque langage existent et donnent ainsi des résultats d'analyse pessimistes, c'est une conséquence liée à la centralisation des langages sur l'architecture plutôt que l'analyse temporelle.

5.2.2 AADL

Pour le langage AADL, il est possible de représenter un réseau avec les éléments fournis par le langage. La figure 5.2 présente un exemple de modélisation d'un système temps réel.

Il est possible d'extraire les informations essentielles du système comme les tâches, leurs périodes, échéances et priorités. Il est également possible de créer une transaction sur le système. Ainsi, le principe Maître/Esclave de LIN peut être représenté sous AADL. Sans mettre en place une annexe pour les définir, il n'est pas possible d'indiquer, pour les communications sur le bus CAN, la longueur des messages, leur priorité, ou la période. Seules les informations concernant les tâches associées au message sont connues. Il est également impossible d'indiquer

spécifique la présence de commutateurs en tant que tel dans le réseau dans le cas d'un réseau commuté sans annexe. Cet annexe peut varier selon l'utilisateur et ainsi une transformation doit être faite par rapport à chaque annexe c'est-à-dire créer autant de transformations que d'annexes.

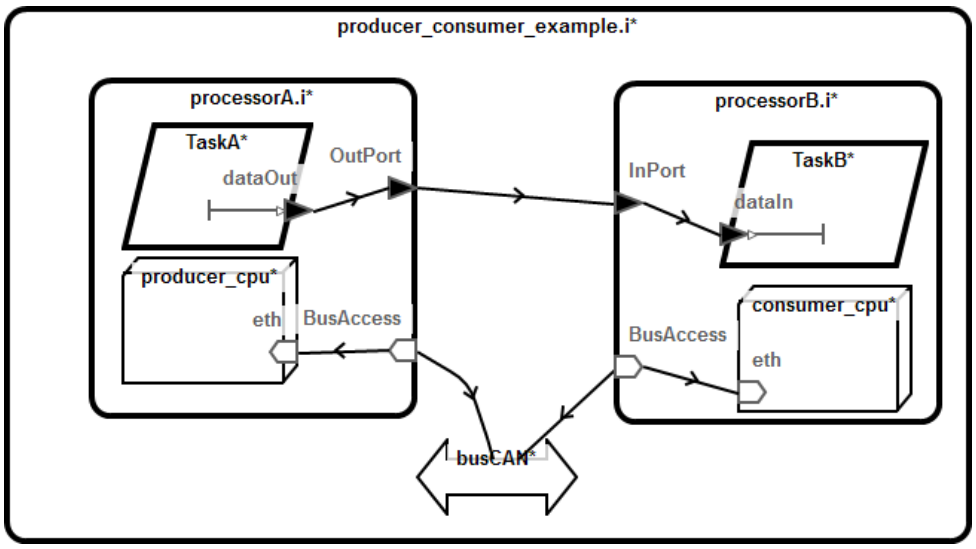


FIGURE 5.2 – Exemple de modélisation réseau sous AADL

5.2.3 UML MARTE

Il n'y a pas de représentation graphique universelle pour le profile UML - MARTE. Pour décrire simplement le langage, nous prenons ici la syntaxe concrète de UML - MARTE dans Time4Sys.

La figure 5.3 présente une modélisation d'un réseau CAN. Les deux tâches, instances de `SoftwareResource`, sont localisées sur leur propre processeur (instances de `HardwareResource`). Les processeurs sont munis d'un protocole d'ordonnancement, noté `FixedPriority` sur la figure. Ces tâches peuvent transmettre leur message sur un bus, instance de la classe `Communication Resource`. Le modèle fonctionne par transactions : une chaîne d'étapes définissant les précédences entre les tâches et les messages. Une priorité peut être affectée à chaque étape (`Step`).

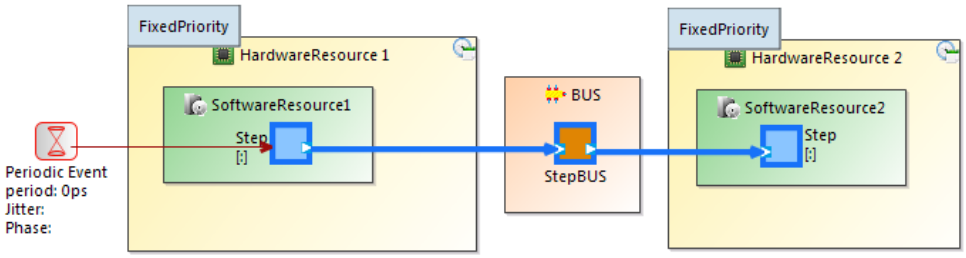


FIGURE 5.3 – Représentation basique du réseau

Supposons que la modélisation requiert de mettre en place une architecture commutée. La figure 5.4 présente une possibilité de représenter les commutateurs. Pour représenter un réseau de type AFDX, un VL transite par plusieurs commutateurs `CommunicationResource` dans le méta-modèle. Pour représenter plusieurs VL, on pourrait ajouter plusieurs étapes dans le commutateur, une étape correspondant à un passage dans un commutateur. De plus, la représentation des réseaux ne permet pas de spécifier d'arbitrage sur les commutateurs, tel le FIFO ou la file

d'attente. Les priorités des flux, pour le réseau ATM, peuvent être indiquées individuellement dans les étapes incluses dans les commutateurs, à condition de s'assurer que les priorités soient les mêmes tout au long du flux.

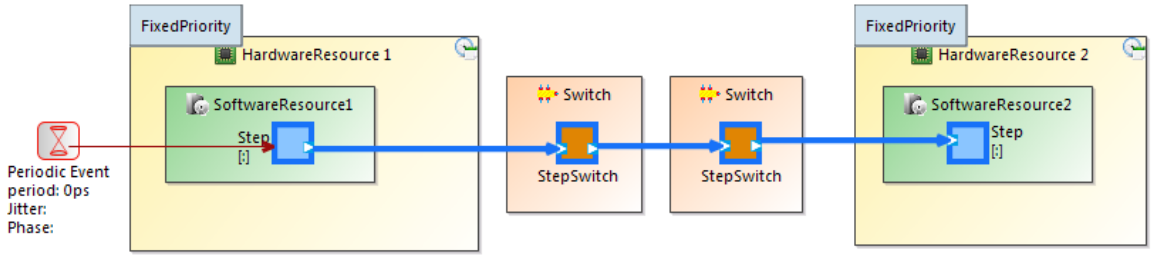


FIGURE 5.4 – Représentation d'un flux (ou virtual link en AFDX)

5.2.4 MoSaRT

Dans le langage de modélisation MoSaRT, les paramètres retenus sont orientés analyse temporelle. Malgré cela, les artéfacts de modélisation réseau sont limités. On montre dans la figure 5.5 la modélisation dans MoSaRT du même cas que précédemment.

Le système contient deux processeurs connectés au réseau (représenté sous `communicationChannel`) et ce réseau est spécifié par un port (instance de `networkPort`) sur le routeur (instance de `communicationRouter`). Ces deux derniers objets sont obligatoires pour modéliser le réseau selon la sémantique MoSaRT. En effet, d'après le méta-modèle décrit en figure 3.15, il est requis de mettre au moins un port sur le réseau et ce, pour décrire le protocole et le débit du système. De plus, afin de créer l'instance d'un port, le routeur est nécessaire.

Dans le cadre d'un bus CAN, les messages transitant par le réseau ne possèdent pas de priorités dans le modèle et les messages sont simplement spécifiés comme transitant sur le bus.

Concernant la partie logicielle, la figure 5.5 présente la modélisation graphique des tâches logicielles. Deux tâches de type `schedulableTask` ont accès à une ressource, instance de `remoteCommResource`, pouvant contenir des données (instance de `transmittedData`). Les deux tâches sont associées à leur processus local `spaceProcess` pour leur appliquer l'algorithme d'ordonnancement. La tâche 1 produit des données qui seront lues par la tâche 2.

Il est possible ensuite de décrire le comportement des tâches entre elles. Cependant, il n'est pas possible, à partir des deux derniers diagrammes, de décrire le système complètement : il est possible que la ressource n'ait qu'une mémoire de type tableau noir au lieu d'avoir un message sur un réseau ce qui pourrait conduire à un comportement différent.

Il s'agit de décrire le comportement qui ne peut être fait par les diagrammes structurels précédents. Dans cette figure, l'activité de la tâche 1, représentée par `taskActivity`, est déclenchée périodiquement par `timeTrigger`. L'activité de la tâche 2 va ensuite être activée à la fin de l'activité de la tâche 1.

Des éléments indispensables pour l'analyse des réseaux ne se retrouvent pas dans la modélisation effectuée ici. Par exemple, il n'est pas possible de décrire le chemin parcouru par les messages dans le cas d'un réseau commuté.

Un réseau CAN ne peut non plus pas être décrit étant donné que la priorité des messages ne peut être décrite dans la partie logicielle du modèle. Néanmoins, il est possible de représenter un paradigme Maître/Esclave avec les précédences de communication entre les activités représentés sur la figure 5.5, partie comportement temporelle, le Maître précédent l'esclave.

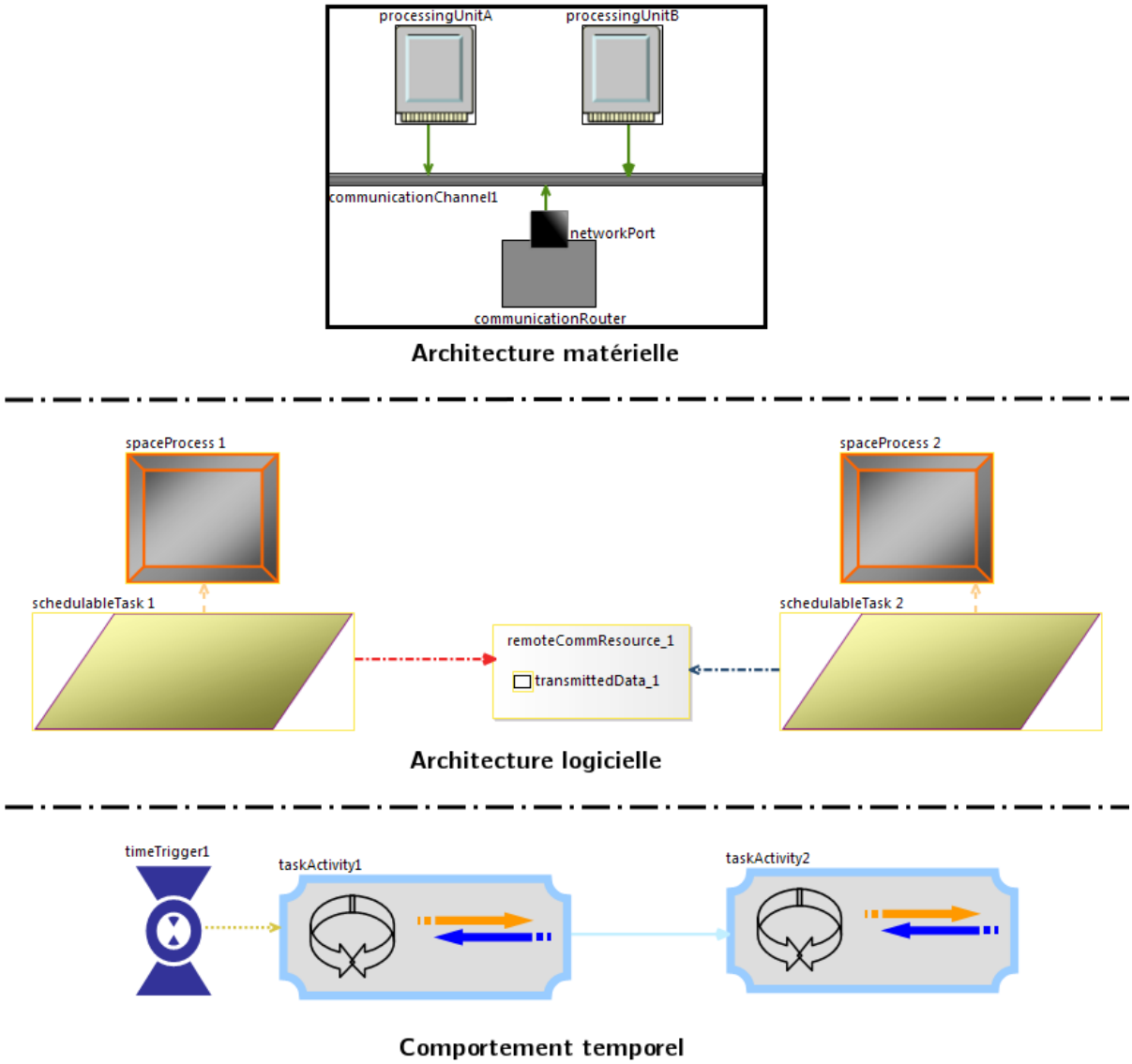


FIGURE 5.5 – Modèle initial MoSaRT

Finalement, le langage manque donc de puissance d'expression pour modéliser les réseaux temps réel, de type AFDX et CAN entre autres, afin de conduire des analyses.

5.2.5 Discussion

La modélisation des systèmes temps réel est souvent faite avec le profil UML - MARTE ou bien AADL. Chaque langage présente des avantages et des inconvénients. Dans la suite, la capacité à représenter de façon fidèle pour analyser le système est discutée pour chaque langage. En effet, comme vu dans la section 3.5.1, plus le modèle est détaillé, moins l'analyse temporelle est pessimiste.

La modélisation avec AADL est orientée conception du système et non analyse du système. Les éléments présents sont utiles majoritairement pour l'architecture système et l'implémentation, par génération de code par exemple, plutôt que pour l'analyse temporelle du système. En conséquence, il n'est pas possible de spécifier les priorités utilisés pour les messages réseau et la création de commutateurs est difficile du fait de l'abstraction des composants. De même, par cette abstraction, la représentation du système est moins proche du système réel et donc

l'analyse en découlant sera pessimiste.

Pour le langage UML - MARTE, malgré une représentation suffisante pour un réseau de type CAN, le langage présente des difficultés pour les réseaux commutés. Cette représentation présente un inconvénient, il n'est pas possible d'associer tous les commutateurs à un seul réseau, dans le modèle UML - MARTE. Pour convertir le protocole réseau, il faut opérer des changements profonds pour rapprocher le modèle du système réel interdisant ainsi une modélisation incrémentale du système. Le protocole d'arbitrage des messages sur les commutateurs est également impossible à représenter restreignant ainsi la représentation d'un réseau de type ATM.

Le langage MoSaRT présente également des difficultés d'expressivité dans la représentation des réseaux. Malgré un langage proche de l'analyse, la représentation des réseaux n'est pas d'assez grande précision dans le cas de réseaux commutés : les commutateurs sont inexistants et donc le chemin d'un message de bout en bout n'est pas décrit. De même, pour des réseaux de type CAN, la priorité n'est pas instanciable.

Le tableau 5.1 ci-dessous résume pour chaque modèle sa puissance d'expression concernant un réseau.

Support du concept par le langage	**AADL**	**UML MARTE**	**MoSaRT**
AFDX			
Modélisation d'un commutateur	✗	✓	✗
Enchaînement des commutateurs (Virtual Link)	✗	✓	✗
Arbitrage sur les commutateurs	✗	✗	✗
Latence fixe sur les commutateurs	✗	✓	✗
ATM			
Priorité sur les messages	✗	✓	✗
Enchaînement des commutateurs	✗	✓	✗
FIFO à priorités sur les commutateurs	✗	✗	✗
LIN			
Priorité sur les messages	✗	✓	✗
Paradigme Maître/Esclave	✓	✓	✓
CAN			
Priorité sur les messages	✗	✓	✗
Débit	✓	✓	✓

Tableau 5.1 – Support des concepts par les langages

5.3 Concepts spécifiques et concepts communs aux réseaux

Les analyses sont fortement dépendantes des hypothèses du modèle et des paramètres mis en jeu dans l'analyse comme vu dans la section 2.4. Il s'agit donc de créer un méta-modèle orienté analyses du réseau qui puisse s'adapter à tout réseau existant. Il faut également s'assurer que le modèle contenant les données du réseau puisse être transformé en un modèle d'analyse avec les informations essentielles pour l'analyse choisie. Dans cette section, nous présentons des concepts pouvant être spécifiques/communs aux protocoles réseau.

Actuellement, les systèmes sans-fil étant peu utilisés dans le cadre d'applications critiques, ce type de réseau ne sera pas traité.

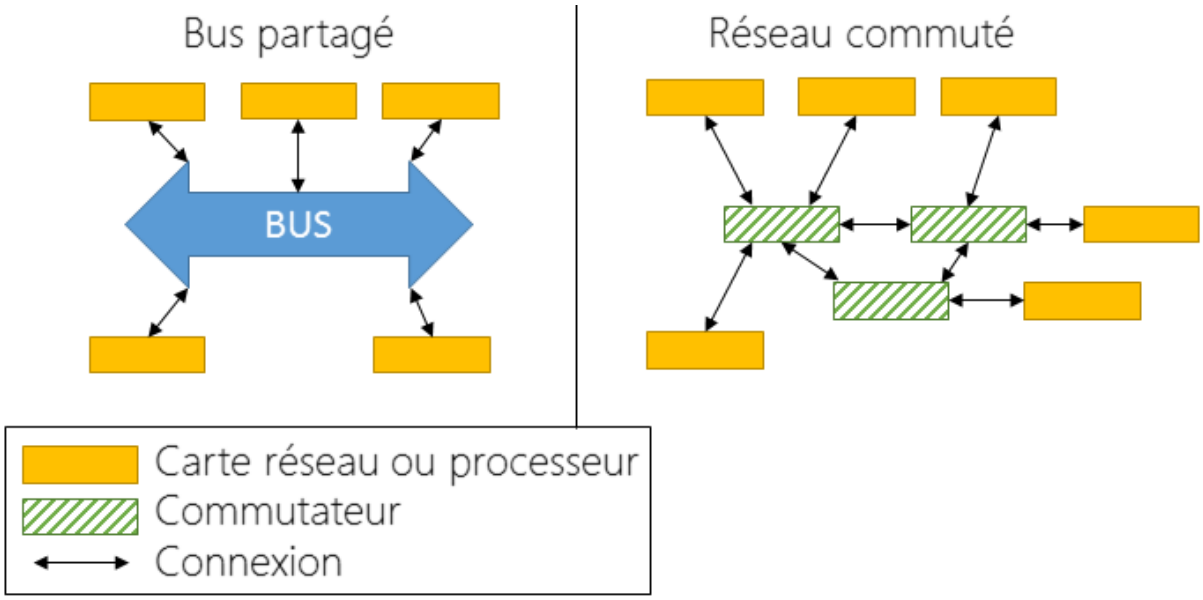


FIGURE 5.6 – Différentes topologies de réseau

5.3.1 Concepts spécifiques

On peut classifier la topologie des réseaux en deux grandes catégories : les architectures commutées et les architectures à bus partagé. La figure 5.6 résume les grandes types de réseau. Il est ainsi possible de catégoriser les différents réseaux existants.

Dans chacune de ces catégories, on peut retrouver des concepts qui sont spécifiques à ces réseaux, qui ne peuvent se retrouver dans les autres réseaux. Par exemple, la notion de commutateur est très courante dans les réseaux Ethernet ou les réseaux AFDX. En revanche, les commutateurs sont inexistants dans les bus tels LIN, CAN ou MIL-STD-1553.

Le chemin physique d'un message diffère également entre ces catégories. Dans un réseau commuté, le chemin transite par les commutateurs et selon la destination, le message est orienté vers le commutateur suivant. Dans les applications critiques, le chemin est statique pour assurer le déterminisme du réseau. Dans un réseau ATM, les chemins peuvent être dynamiques pour éviter des commutateurs défaillants.

Dans le domaine des systèmes temps réel, la politique d'arbitrage des messages ne s'applique pas de la même façon. En effet, pour les réseaux commutés, l'arbitrage s'applique sur les commutateurs. Par exemple, la politique peut être du FIFO comme sur AFDX ou bien du FIFO à priorités.

Cela diffère des réseaux à bus partagé dans lesquels, le réseau doit s'arbitrer par lui-même comme dans les protocoles de type CSMA. Autrement dit, l'arbitrage des messages s'applique sur le réseau en entier. Dans le cas de CAN, les messages possèdent des priorités, ce qui permet d'ordonnancer les messages. Pour les réseaux LIN ou MIL-STD-1533, les messages sont demandées par le nœud maître, donc les messages dans les noeuds esclaves sont en attente du signal du maître pour être transmis.

5.3.2 Concepts communs

Malgré les différences entre les réseaux, on retrouve des concepts communs, des éléments qui peuvent être représentés dans un méta-modèle. Les concepts communs sont des notions qui peuvent être retrouvées dans tous les réseaux.

		Réseaux commutés		Réseaux à bus partagé		
	Concept	**AFDX**	**ATM**	**CAN**	**LIN**	**1553**
Concepts spécifiques	Medium de communication	Switches & câbles Ethernet		Câbles réseau commun à tous les noeuds		
	Politique d'arbitration	FIFO sur chaque switch	FIFO à priorités sur les switches	Priorité fixe sur les messages	Communication Maître esclave	
	Noeuds de communication	Processeurs & Switches		Processeurs		
	Chemin d'un message	Virtual Link	Statique ou dynamique	Statique	Le bus en entier	
	Latence	Sur les switches	Sur les switches, les files d'attents et priorités	La période de consultation du noeud par le maître	Temps de transmission	
Concepts communs	Priorité	Sans incidence	Priorité sur messages	Priorité sur messages	Sans incidence	
	Tâches d'Origine & Destination	Une tâche d'envoi et multiples destinataires				
	Données	Trame et taille de trame				
	Changement de réseau	Routeur de changement de réseau ou un processeur est connecté à 2 réseaux.				

TABLEAU 5.2 – Exemples de concepts indépendants du protocole et spécifiques au protocole

Par exemple, il faut que des messages soient transmis sous forme de trame sur le réseau, entre deux processeurs ou plus, pour que le réseau soit utile. Ensuite, ces messages sont envoyés par une tâche spécifique pour l'envoi et les messages sont reçus par une ou plusieurs tâches. Chaque trame est envoyée avec une certaine longueur de trame en octet. Dans le cadre de l'analyse de réseau, il est possible de fournir un intervalle de longueurs de trames ou bien donner la pire longueur de trame, celle qui occupe le plus de temps le réseau.

La priorité des trames est également un paramètre qui peut être utilisé dans le cadre d'analyses en particulier pour CAN où la priorité est primordial dans le calcul de temps de réponse. Le protocole ATM, réseau commuté, partage cette particularité d'avoir des messages possédant des priorités.

5.3.3 Conclusion

Il est possible de mettre en place une comparaison entre les réseaux commutés et les réseaux à bus partagé. Les différences résident surtout dans la topologie des réseaux, c'est-à-dire comment ils sont connectés. Les protocoles sous-jacents sont la combinaison de divers paramètres qui peuvent être attribués au réseau tels la priorité des messages ou bien l'origine et les destinations des messages. On résume dans le tableau 5.2 les concepts communs et spécifiques aux réseaux. Le but est ici de représenter toutes les concepts dans un seul méta-modèle.

5.4 Une structure de méta-modèle réseau dans MoSaRT

Ces concepts communs peuvent être utilisés pour proposer un méta-modèle incluant ceux-ci, afin de pouvoir modéliser des réseaux temps réel, ainsi que des systèmes distribués, en vue de leur analyse temporelle. Le but de cette contribution est de pallier le manque d'expressivité des langages de modélisation. En effet, ceux-ci ne peuvent retranscrire les détails nécessaires pour obtenir une analyse réseau conservative. Dans cette section, nous proposons une structure permettant de relier les modèles à l'analyse, ceci en proposant le méta-modèle le plus simple possible.

On propose d'abord de mettre en place une structure générique supportant les différents réseaux malgré leur .

La figure 5.7 présente une structure commune aux réseaux : un réseau est composé de noeuds pouvant ainsi être soit un processeur soit un commutateur. Les connexions physiques relient deux noeuds dans le réseaux. À travers ces connexions transitent des messages qui peuvent avoir une priorité dans le réseau.

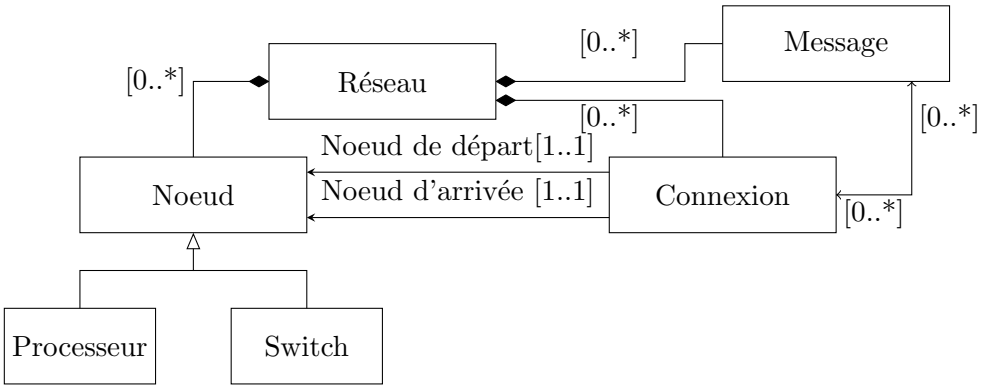


FIGURE 5.7 – Structure commune proposée

5.5 Extension du framework MoSaRT

Cette architecture seule ne permet que d'analyser les sous-systèmes réseaux des systèmes temps réel distribués. Pour analyser la globalité du système, comme dans le cadre d'une analyse holistique, il faut intégrer cette structure dans un méta-modèle plus global comme MoSaRT décrit dans la section 3.6. Ce langage est choisi car il présente la particularité d'être un langage orienté analyses.

5.5.1 Formalisation

Cette section définit les concepts ainsi que les relations entre eux. Pour ce faire, on ne définira que les notions utilisés dans le cadre de réseaux. Un système $\mathcal{SYS}$ est composé :
- d'un ensemble de processeurs $\mathcal{P}$,
- d'un ensemble de réseaux $\mathcal{R}$,
- d'un ensemble de tâches $\mathcal{T}$

Le système est donc tel que $\mathcal{SYS} =< \mathcal{P}, \mathcal{R}, \mathcal{T} >$.
Chaque réseau est constitué de :

— $\mathcal{P}_R \subset \mathcal{P}$ un sous ensemble de processeurs,

— $\mathcal{S}$ l'ensemble des commutateurs (*switchs*) au nombre de k tels que $\mathcal{S} = \{s_1, s_2, ..., s_k\}$,

— $\mathcal{C} = \{c_1, c_2, ..., c_m\}$ les m connexions entre les processeurs tels que $\mathcal{C} = \{c_1, c_2, ..., c_m\}$,

— $\mathcal{M}$ et l'ensemble des n messages tels que $\mathcal{M} = \{m_1, m_2, ..., m_n\}$.

Un réseau sera donc défini tel que :

$$\mathcal{R} =< \mathcal{P}_R, \mathcal{S}, \mathcal{C}, \mathcal{M} > \tag{5.1}$$

De plus, les connexions $c_i, i \in 1..m$, connectent des processeurs et des commutateurs entre eux.

$$\mathcal{C}_j =< n_{origine}, n_{dest} >, \quad n_{origine}, n_{dest} \in \mathcal{P}_\mathcal{R} \cup \mathcal{S} \tag{5.2}$$

Un message est envoyé par une tâche $t_{orig} \in \mathcal{T}$ et est reçu par un certain nombre de tâches $T_{dest} \subset \mathcal{T}$. Le message va transiter par un ensemble C_m de connexions qui maillent le système. C_m est donc défini tel que $C_m \subset \mathcal{C}$. Un message sera donc défini tel que $m_i =< t_{orig}, T_{dest}, C_m >$

5.5.2 Ajout des concepts réseaux dans MoSaRT

5.5.2.1 Extension de la syntaxe abstraite

On intègre la structure de la figure 5.7 dans le méta-modèle de MoSaRT. La figure 5.8 présente un résumé des modifications apportées au méta-modèle MoSaRT, la partie supérieure présentant le méta-modèle initial, et la partie inférieure présentant l'extension supportant les réseaux. Sur cette figure, les éléments logiciels commencent par le préfixe `So` pour *Software Operator* et les éléments matériels commencent par `Hp` pour *Hardware Platform*. Dans l'existant, une tâche `SoSchedulableTask` est ici affectée sur un processeur `HpProcessingUnit`. Les messages `SoRemoteCommResource` peuvent transmettre un ou plusieurs données en référençant la tâche source et les tâches destinations.

Initialement, le méta-modèle d'exécution physique (*Hardware Platform*) comprenait seulement `HpCommunicationChannel` pour instancier le réseau. En faisant la correspondance entre la structure précédente et l'extension du méta-modèle, un réseau `HpCommunicationChannel` possède ainsi des nœuds abstraits `HpNetworkNode`, et des connexions entre les noeuds `HpFlowCarrier`. Un nœud réseau peut ainsi être un commutateur `HpCommunicationSwitch` et peut posséder ainsi une latence `latency` qui peut être fixe (dans le cas d'AFDX) ou à calculer (dans

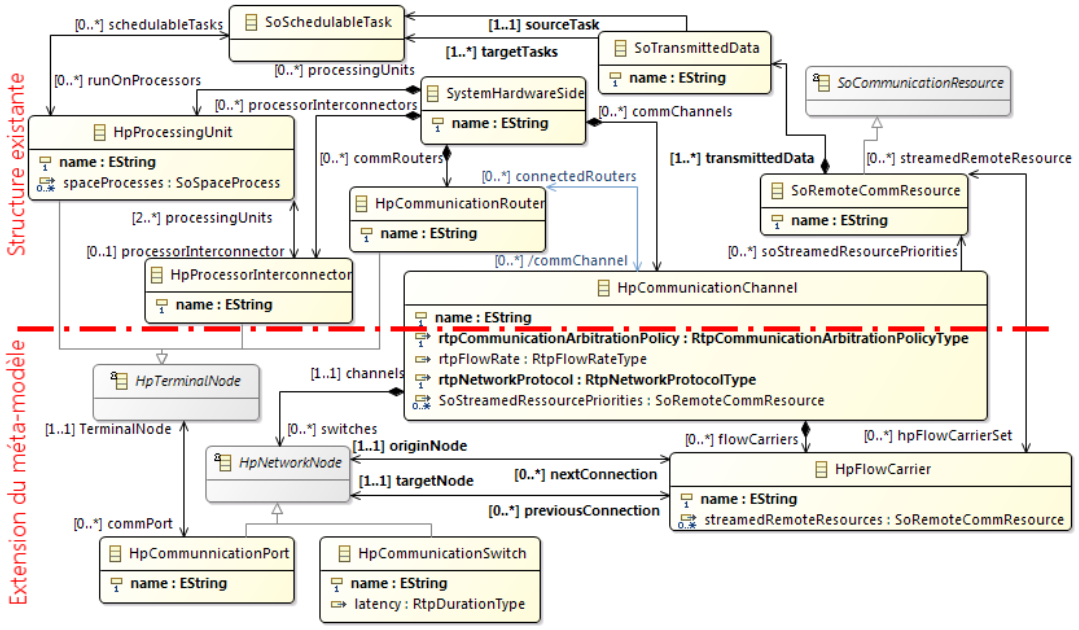


FIGURE 5.8 – Extrait de l'extention du méta-modèle MoSaRT

le cas de ATM, par exemple). Un nœud réseau peut également être un port `HpCommunicationPort` auquel est lié un noeud terminal `HpTerminalNode` tel un processeur `HpProcessingUnit` ou une passerelle chargée de faire le lien entre deux réseaux `HpCommunicationRouter`.

Le réseau `HpCommunicationChannel` possède des propriétés permettant notamment de spécifier le protocole utilisé. Ainsi, il faut déterminer quel est le protocole `rtpNetworkProtocol` utilisé pour le réseau, l'analyse en étant fortement dépendante. Le débit `rtpflowRate` et le protocole d'arbitrage `rtpCommunicationArbitrationPolicy` peuvent être définis pour tout le réseau. Il est également possible de définir les priorités présentes dans le réseau avec la propriété `SoStreamedResourcePriorities` référençant ainsi les messages dans le réseau par priorités. On suppose ainsi que les messages ne peuvent avoir deux mêmes priorités.

Ces messages peuvent emprunter un chemin particulier de connexions. Il référencent donc un ensemble de connexions `hpFlowCarrierSet` pour spécifier les chemins traversés par le message. Réciproquement, les connexions spécifient l'ensemble des messages transitant par ceux-ci par la propriété `streamedRemoteCommResource`.

Ainsi, dans ce méta-modèle, une classe peut représenter un concept spécifiques pour plusieurs réseaux. Par exemple, `HpFlowCarrier` peut représenter soit un message sur un réseau CAN ou bien un VL en AFDX. De même, `HpCommunicationChannel` peut représenter soit le domaine réseau pour AFDX ou bien relie tous les processeurs en relation entre eux pour CAN. Ceci permet au final de créer une méthode de modélisation incrémentale en changeant peu de paramètres entre deux incréments du modèle.

5.5.2.2 Renforcement de la sémantique statique

Le méta-modèle étendu est ensuite enrichi par un ensemble d'invariants permettant de vérifier un certain nombre de propriétés sur les modèles construits à partir de ce méta-modèle. Les invariants suivants ont été ajoutés :

— **Pas de boucles** : les connexions doivent avoir une origine et une destination différentes, i.e. dans la figure 5.7 les nœuds de départ et d'arrivée sont différents, cela n'interdit pas l'existence d'anneaux, mais un anneau ne peut être constitué que d'un seul nœud,

— **Unique connexion physique** : il ne peut exister qu'une seule connexion entre deux noeuds,

— **Priorités déclarées** : la liste des priorités doit être composée de tous les messages dans les cas où le protocole réseau l'exige,

— **Tâche écrivaine unique** : dans le cas de réseaux spécifiques, seule une tâche doit être autorisée à envoyer un message , de multiples tâches étant autorisées dans le méta-modèle MoSaRT

— **Chemin complet** : le chemin complet d'une tâche doit être indiqué, i.e. il ne peut y avoir de lacunes dans le chemin de la source à l'une des destinations.

Dans la suite, on présentera la modélisation des systèmes si le système comprend un réseau de type bus ou bien de type commuté.

Implémentation des invariants Utiliser le langage Ecore permet l'ajout de règles OCL [OMG06b] avec l'éditeur *OCLinEcore* permettant ainsi d'imposer des règles structurelles ne pouvant être exprimées dans le méta-modèle. En effet, les cardinalités seules ne sont pas suffisantes à exprimer la plupart des invariants structurels. Par exemple, pour le premier invariant, l'objet `HpFlowCarrier` doit avoir les noeuds d'origine et de destination différents. La règle OCL correspondant à cette règle est présentée sur le listing 5.1 et dans la figure 5.9.

Listing 5.1 – Instanciation de la règle **Pas de boucle** sur le réseau

```
context HpFlowCarrier
invariant
   PasDeBoucles : self.originNode.oclIsTypeOf(HpCommunicationSwitch)
           and self.targetNode.oclIsTypeOf(HpCommunicationSwitch)
   implies self.originNode <> self.targetNode;
```

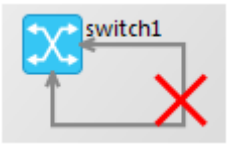


FIGURE 5.9 – Invariant sur les connexions pour éviter les boucles matérielles

Il s'agit aussi d'éviter d'avoir deux connexions reliant les mêmes noeuds pour éviter les doublons de connexion et assurer le déterminisme du réseau. Dans le listing 5.2, en considérant un nœud, on recherche tous les nœuds connectés à celui-ci et doivent être distincts deux à deux.

Listing 5.2 – Instanciation de la règle OCL d'**unique connexion physique**

```
context HpNetworkNode
invariant
   UniqueConnexion : self.nextConnection.targetNode->
           union(self.previousConnection.originNode)->forAll(a,b|a<>b);
```

Le listing 5.3 permet d'assurer l'instanciation de priorités lorsque le protocole l'exige.

Listing 5.3 – Instanciation de la règle OCL **Priorités déclarées**

```
context HpCommunicationChannel
invariant
   PrioriteCAN : self.rtpNetworkProtocol.oclIsTypeOf(RealTimeProperties
```

FIGURE 5.10 – Invariant sur les connexions pour éviter les doubles connexions

```
                                    ::RtpProtocolAndPolicies::CANNetworkProtocol)
  implies self.SoStreamedRessourcePriorities -> notEmpty();
```

Le listing 5.4 assure de n'avoir qu'une seul tâche autorisé à écrire sur une ressource lorsque cette ressource est un message transmise sur un réseau, ici, AFDX.

Listing 5.4 – Instanciation de la règle OCL **Tâche écrivaine unique**

```
context HpCommunicationChannel
invariant
  TacheUnique : self.rtpNetworkProtocol.oclIsTypeOf(RealTimeProperties
                                    ::RtpProtocolAndPolicies::AFDXNetworkProtocol)
  implies self.SoStreamedRessourcePriorities ->
      forAll(ressource|ressource.writerTasks -> size()=1);
```

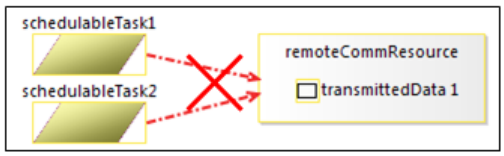


FIGURE 5.11 – Invariant pour éviter les doubles connexions

Le listing 5.5 partage la règle **Chemin Complet** en plusieurs règles. Les deux premières assure que les processeurs d'arrivées et de départ instanciés par le chemin matériel correspondent bien aux tâches d'écriture et de lecture instanciés dans la partie logicielle (règles `Origine` et `Dest`) La dernière règle assure que le chemin soit complet (règle `CheminComplet`).

Listing 5.5 – Instanciation de la règle OCL de la règle **CheminComplet**

```
context SoRemoteCommResource
invariant
  Origine : self.hpFlowCarrierSet->select(a|a.originNode->union(targetNode)
                                ->select(b|b.TerminalNode
                                    =self.writingTask.runOnProcessors))->notEmpty();
  Dest : self.hpFlowCarrierSet->select(a|a.originNode->union(targetNode)
                                ->select(b|b.TerminalNode
                                    =self.writingTask.runOnProcessors))->notEmpty();

   CheminComplet : self.hpFlowCarrierSet->exists(a,b|(a<>b and
       a.targetNode.oclIsType(HpCommunicationSwitch)) implies
           (a.targetNode = b.targetNode or
            a.targetNode = b.originNode )) or
                  self.hpFlowCarrierSet->exists(a,b|(a<>b and
                      a.originNode.oclIsType(HpCommunicationSwitch) ) implies
           (a.originNode = b.originNode or
            a.originNode = b.targetNode ))
```

Ces invariants inclus dans le méta-modèle permettent de modéliser des systèmes en évitant

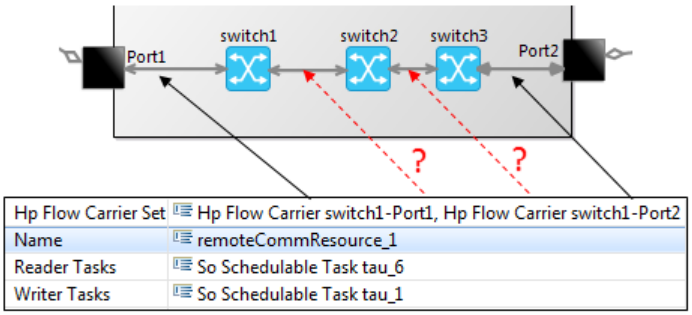


FIGURE 5.12 – Invariant pour assurer que toutes les connexions d'un chemin sont instanciées

les erreurs de modélisation des réseaux abordés dans cette section.

5.5.3 Architectures de type "bus"

Dans ce type d'architecture, un bus est représenté par un ensemble de processeurs connectés au réseau sans connexion "point à point". Les processeurs sont connectés à un bus commun.

5.5.3.1 Exemple

La figure 5.13 ne présente que l'architecture de type bus d'un système distribué se composant de 9 tâches réparties sur trois processeurs A, B et C. Les tâches communiquent entre elles par le biais de plusieurs messages. Les messages m_1, m_2, m_3 sont transmis sur le réseau et le message m_4 est transmis de façon interne entre deux tâches qui appartiennent au même processeur. Les processeurs A et C sont ordonnancés avec un protocole à priorités fixes selon les priorités indiqués dans la figure. Le processeur B est ordonnancé avec le protocole dynamique EDF.

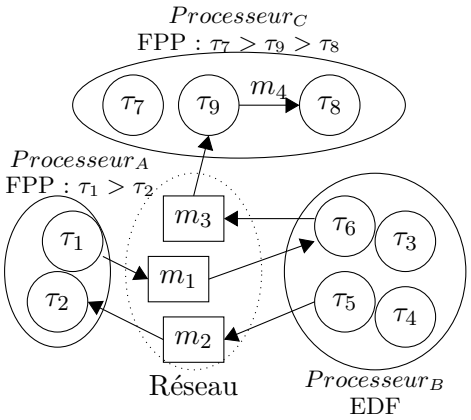


FIGURE 5.13 – Exemple de configuration réseau

Les caractéristiques des tâches telles que les périodes, les échéances et le temps d'exécution sont indiquées dans le tableau 5.3.

Tâche	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9
Périodes (ms)	100	160	40	60	160	100	60	100	100
Échéances (ms)	200	280	60	85	150	220	60	320	250
Temps d'exécution (ms)	52	52	10	20	52	52	28	25	14

TABLEAU 5.3 – Caractéristiques temporelles des tâches de l'exemple

5.5.3.2 Instanciation de l'exemple avec MoSaRT

La figure 5.14 présente la partie architecture logicielle du système. Dans cette figure, les tâches sont localisées sur des processus et peuvent lire ou écrire sur une ressource distante, a priori inconnue. C'est en observant les propriétés de la ressource distante que l'on peut voir si un chemin `Hp_Flow_Carrier_Set` pour le message est déclaré. On observe aussi les ressources utilisées en écriture ou en lecture pour chaque tâche dans les propriétés, les ressources étant les messages transmis sur un réseau. Il est ainsi possible de suivre les chaînes d'exécution. A titre d'exemple, la tâche 1 écrit sur la ressource 1 (notée `remoteCommResource_1`) et cette ressource sera lue par la tâche 6 (notée tau_6).

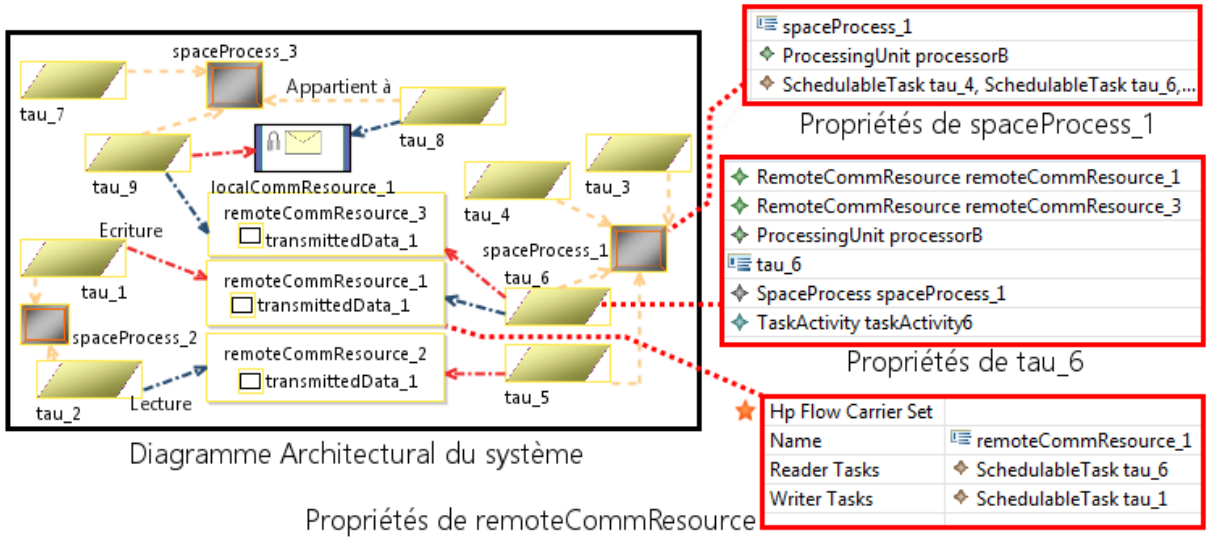


FIGURE 5.14 – Diagramme MoSaRT de l'architecture système

La figure 5.15 présente la partie comportementale du système dans la version graphique du langage MoSaRT. Le comportement de chaque tâche se trouvant dans l'architecture logicielle du système est représenté par l'artéfact `taskActivity` dans la partie comportementale. Dans cette figure, les tâches sont activées de façon périodique et le cas échéant, sont activées une fois que les messages sont reçus. Ce sera dans les propriétés des tâches que les temps d'exécutions seront indiqués ainsi que les relations entres les tâches dans ce diagramme.

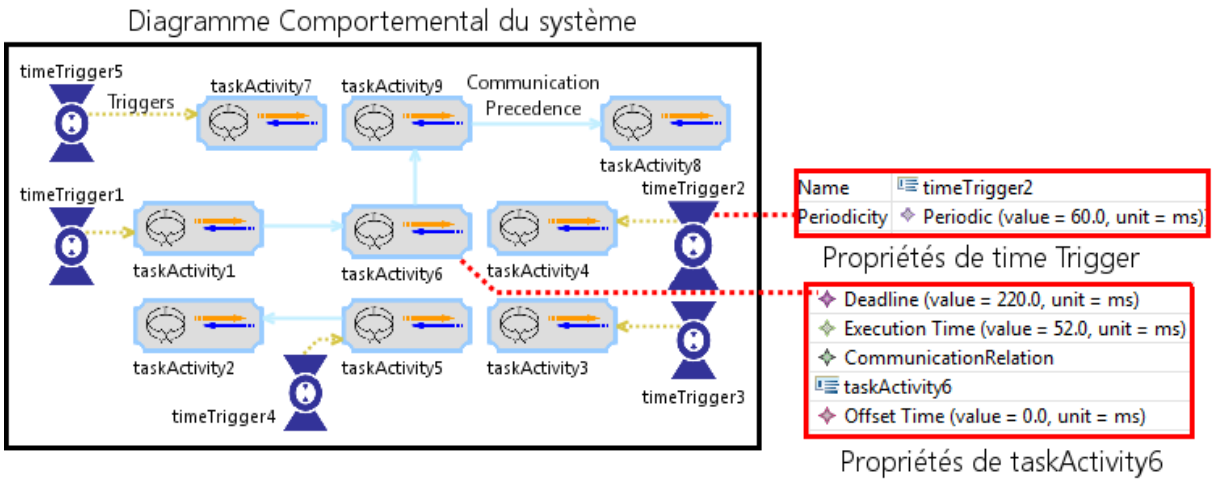


FIGURE 5.15 – Diagramme MoSaRT du comportement temporel du système

La figure 5.16 présente la partie matérielle de l'architecture, modélisé selon le langage MoSaRT. Le réseau choisi étant un bus CAN, les commutateurs `HpCommunicationSwitch` n'apparaissent pas, étant donné que ce n'est pas un réseau commuté.

Nous voyons ici la philosophie de modélisation de MoSaRT : d'une part une représentation purement logicielle (figure 5.14), ayant des besoins de ressources. D'autre part, une modélisation des rythmes d'activation des tâches (et donc des messages envoyés), ce que MoSaRT appelle le comportement (figure 5.15), enfin une modélisation des ressources et de leur arbitrage (figure 5.16).

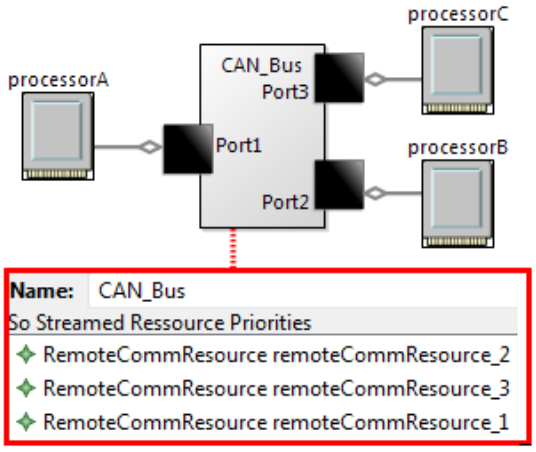


FIGURE 5.16 – Modélisation du réseau dans MoSaRT

Dans un réseau CAN (et sur un bus en général), tous les noeuds (que l'on peut assimiler aux processeurs) présents sont destinataires, les connexions entre les noeuds et le chemin que parcourt un message sur le bus ne sont pas primordiales. Les messages sont classés par ordre de priorités dans une liste ordonnée lors de la déclaration dans le bus des messages y transitant, comme montré dans la figure 5.16. Le méta-modèle proposé permet de facilement changer de type de réseau, et de passer par exemple, d'un bus CAN à un bus AFDX avec peu d'efforts de modélisation. Ce changement est facilement géré dans le framework MoSaRT grâce à la modularité et l'indépendance assurée par les différents diagrammes. En effet, en cas de changement de réseau ou de protocole, seul le diagramme d'architecture physique est impacté.

5.5.4 Architectures commutées

Dans le cadre d'une modélisation incrémentale, l'architecture en "bus CAN" peut être transformée facilement en un réseau commuté dans le cas d'une analyse insatisfaisante par exemple. Dans ce cadre, pour raffiner le modèle, il suffit d'ajouter dans le réseau des commutateurs et de modifier la propriété de l'objet réseau en un réseau commuté. La figure 5.17 présente la modélisation qui est faite pour le même système modélisé précédemment en considérant un réseau commuté. Sur cette figure, les processeurs sont reliés entre eux par le biais des commutateurs. Sur les commutateurs, la latence peut être fixée, (à 16 μs dans un réseau AFDX) ainsi que les connexions avec les autres nœuds. Les `HpFlowCarrier`, les connexions entre deux noeuds, référencent les VL transitant par celle-ci. Réciproquement, les `SoRemoteCommResource` assimilés ici à des VL, recensent les connexion qui doivent être utilisés par le VL. Les processeurs sont connectés au réseau par le biais d'un port `HpCommunicationPort` référencé dans le port comme un `TerminalNode`. Ce port permet de faire la connexion vers le réseau et présente les mêmes propriétés pour connecter les nœuds du réseau.

Il reste toujours possible d'utiliser des priorités dans le réseau pour une architecture le requérant comme ATM, ou bien une version d'AFDX avec priorités. La déclaration des priorités

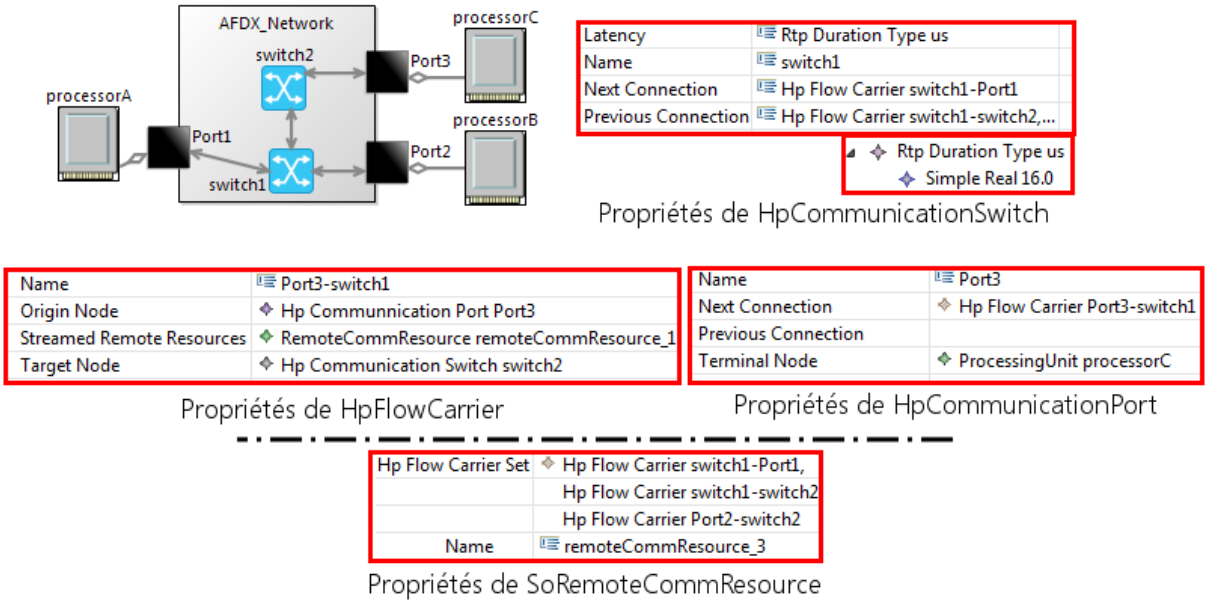


FIGURE 5.17 – Paramètres de modélisation MoSaRT pour AFDX

serait similaire à celle présentée dans la figure 5.16.

La modélisation de réseaux est relativement aisée, grâce à la modélisation incrémentale proposée par le langage, avec des objets polyvalents dans leur utilisation, permettant un changement relativement simple de type de réseau dans un modèle. Par exemple, un message réseau `SoRemoteCommResource` dans CAN deviendra ainsi un Virtual Link dans AFDX. Lors de la transformation vers le modèle d'analyse, les éléments non nécessaires dans l'analyse seront simplement ignorés.

5.5.5 Enrichissement d'un référentiel d'analyse pour les modèles d'analyse de réseau

MoSaRT offre la possibilité de créer un référentiel d'analyse pour guider l'architecte logiciel dans la démarche d'analyse temporelle. Ce référentiel d'analyses permet la détermination d'une analyse avec l'aide de règles structurelles sur le modèle (cf. section 3.6.3).

Nous souhaitons ajouter les analyses réseau dans le référentiel d'analyse de MoSaRT. Le référentiel d'analyse rassemble toutes les analyses possibles ainsi que les règles d'identification du contexte d'analyse. Un contexte d'analyse est défini par un ensemble de contraintes (comme celles définies dans la section 2.3.3) que le modèle d'un système doit vérifier pour appliquer cet analyse.

Pour étendre ce référentiel d'analyse, de nouvelles règles doivent être définies en ajoutant les règles concernant les réseaux dans le référentiel. L'objectif de cet ajout est de représenter les contextes d'analyse qui supportent l'analyse temporelle des systèmes distribués (comme présenté dans la section 2.4).

Le tableau 5.4 présente de façon synthétique les contraintes d'analyses des principales analyses pour les protocoles CAN et AFDX, protocoles particulièrement utilisés dans les systèmes embarqués. Les marqueurs ✓ dénotent les conditions qu'il faut respecter pour appliquer l'analyse, les marqueurs ✗ les conditions qui ne doivent pas exister pour l'analyse et les points d'interrogations signifient que les conditions sont sans incidence sur l'analyse.

Contrainte	CAN		AFDX		
	RTA	**Holistique**	**Network Calculus**	**Méthode des Trajectoires**	**Forward Analysis**
Est un réseau CAN	✓	✓	✗	✗	✗
Est un réseau AFDX	✗	✗	✓	✓	✓
Modélisation statique	✓	✓	✓	✓	✓
Systèmes monoprocesseurs	✓	?	?	?	?
Priorité fixe sur tous les messages	✓	✓	✗	✗	✗
Pas de priorité sur les messages	✗	✗	✓	✓	✓
Messages à échéances contraintes	✓	?	?	?	?
Messages périodiques uniquement	✗	✗	✓	✓	✗
Présence de messages apériodiques	✓	✓	✗	✗	✓
Délai de commutation non nul sur switch	✗	✗	✗	✓	✓

TABLEAU 5.4 – Synthèse des contextes dans lesquels s'appliquent les calculs de temps de traversée

Les éléments utilisés dans le tableau sont décrits ci-après :

1. Est un réseau CAN : le réseau est instancié comme étant un réseau CAN,
2. Est un réseau AFDX : le réseau est instancié comme étant un réseau AFDX,
3. Modélisation statique : les chemins des messages sont statiques et les tâches de destination ou d'arrivée ne peuvent migrer sur le système,
4. Systèmes monoprocesseurs : les tâches de destination ou d'arrivée se trouvent localement sur des monoprocesseurs,
5. Priorité fixe sur tous les messages : les messages ne peuvent changer leur priorité,
6. Pas de priorité sur les messages : les messages n'ont pas de priorité,
7. Messages à échéances contraintes : l'échéance est inférieure à la période du message,
8. Messages périodiques uniquement : les messages sont tous périodiques,
9. Présence de messages apériodiques : des messages apériodiques sont introduits dans le système,
10. Délai de commutation non nul sur commutateur : sur les commutateurs le délai physique de commutation est non négligeable,

Ces contraintes doivent être traduites dans le langage MoSaRT afin que le référentiel d'analyses associé puisse orienter le modèle vers l'analyse la plus adaptée. Les contraintes sont exprimées dans le langage OCL. Pour la contrainte de modélisation statique, on peut considérer que les messages sont liés à des tâches et ces tâches ne peuvent migrer ce qui donne, par exemple, la règle OCL donnée en listing 5.6.

Listing 5.6 – Instanciation de la règle OCL de la connexion unique

```
RemoteCommResource.readerTasks->forall(t|t.runOnProcessors->size()=1) and
RemoteCommResource.writerTasks->forall(t|t.runOnProcessors->size()=1)
```

5.6 Exemple de processus d'analyse

Avec la modélisation que nous avons vue de l'exemple en figure 5.13, il est possible d'analyser maintenant le système selon un bus CAN comme illustré par la figure 5.16.

L'application du référentiel d'analyse au modèle complet donne comme proposition de test d'analyse l'analyses holistique [TC94] (voir figure 5.18). Cette fenêtre montre les règles qui sont vérifiées dans le modèle, celles qui doivent être fausses et celles qui n'ont aucune incidence sur l'analyse comme le protocole utilisé localement sur les processeurs. Les nombres correspondants aux règles qui doivent être vraies et qui doivent être fausses font références aux régles déjà figurant dans le référentiel d'analyses.

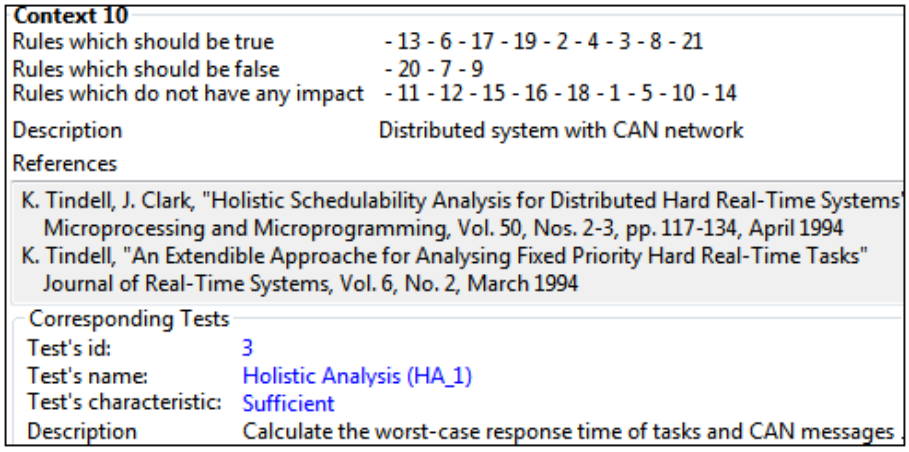


FIGURE 5.18 – Résultat d'un processus d'identification

L'analyse holistique sur un réseau de type CAN permet de calculer les temps de transmission sur le réseau. Cette analyse est implémentée dans MAST. Cet outil d'analyses peut être utilisé à l'aide d'une transformation de modèles. Cette transformation n'étant pas l'objet de ces travaux, elle ne sera pas détaillée.

Pour l'analyse, on considère que le temps de transmission sur le réseau dure $3ms$ soit le temps qu'une trame soit émise complètement. Le tableau 5.5 résume les pires temps de réponse (WCRT) pour chaque tâche obtenus suite à l'utilisation de MAST.

Tâche/Message	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	m_1	m_2	m_3
WCRT (ms)	52	266	10	33	98	16	28	337	219	58	110	177

TABLEAU 5.5 – Résultats d'une analyse holistique pour un bus CAN

5.7 Conclusion

Dans ce chapitre, nous avons présenté des méta-modèles existants pour représenter des systèmes distribués temps réel et ainsi effectuer l'analyse à partir de ces méta-modèles. Ces langages s'avéreraient insuffisants pour analyser les réseaux, puisqu'ils ne permettaient pas de représenter les propriétés nécessaires à un calcul de temps de traversée réseau. Nous souhaitions étendre MoSaRT, tout en gardant la même philosophie de modélisation (logiciel, comportement, ressources), et en ne représentant que les propriétés nécessaires à une analyse temporelle. Nous souhaitions aussi qu'un changement de type de réseau nécessite le moins d'effort de modélisation possible, en mutualisant les concepts entre réseau de type bus, représentant typiquement un espace de collision, et réseau commuté. Nous avons dû cependant introduire certaines propriétés spécifiques au réseau utilisé lorsque celui-ci était nécessaire. Nous avons alors proposé un

méta-modèle, sous forme d'une syntaxe abstraite et d'une syntaxe concrète, qui permet l'analyse de certains réseaux temps réel. Cette structure a enrichi le méta-modèle de MoSaRT, langage orienté analyse temporelle.

À partir d'un exemple simple, la structure a été testée avec des topologies différentes de systèmes distribués temps réel. Cette structure possède des éléments permettant ainsi de modéliser les systèmes distribués et d'analyser le temps de traversée des réseaux.

Cette nouvelle extension a été implémentée dans un langage permettant une modélisation incrémentale : le modèle peut évoluer en fonction des résultats d'analyses. L'extension du méta-modèle MoSaRT garde cet aspect de modélisation. Ainsi, le changement de réseau au cours de la modélisation n'impacte pas le diagramme d'architecture logicielle.

Le *framework* MoSaRT propose également un référentiel d'analyses : ce référentiel a été enrichi avec des analyses adaptés aux systèmes distribués telles l'analyse holistique pour un bus CAN ou le *Forward Analysis* pour un réseau de type AFDX.

Les contextes associés aux analyses réseaux sont également transcrits dans MoSaRT afin de les identifier. Étant donné que le référentiel d'analyses se base sur des règles structurelles pour identifier les contextes d'analyse, de nouvelles règles adaptées pour les réseaux temps réel ont été définies. De plus, le référentiel d'analyses complété ici permet l'orientation de l'utilisateur selon les méthodes possibles pour son modèle voire même la méthode la plus adaptée au modèle.

Ce chapitre comble des lacunes dans la représentation des systèmes distribués temps réel. Cela a permis d'améliorer l'analyse de ces systèmes et ainsi représenter un réseau sans utiliser un langage d'analyses évitant ainsi la formation sur ce logiciel.

Chapitre 6

Extraction conservative et patrons de modèles d'analyses

Chacun a raison de son propre point de vue, mais il n'est pas impossible que tout le monde ait tort.

— Gandhi

Sommaire

Les langages de modélisation présentent un fort intérêt pour l'ingénierie dirigée par les modèles comme vu dans le chapitre 3. En revanche, les langages étant destinés à une large communauté informatique, ils deviennent de plus en plus lourds avec la complexité croissante des systèmes. Dans ce chapitre, on propose d'adapter les méta-modèles aux besoins requis par l'utilisateur. Il s'agit ici de créer des points de vue adaptables aux besoins du concepteur temps réel.

6.1 Introduction

Les systèmes deviennent de plus en plus complexes avec l'évolution des technologies. La possibilité d'extraire un sous-modèle d'un plus grand modèle est une méthode permettant de distribuer les analyses sur plusieurs logiciels. Nous verrons qu'il n'est pas possible, en général, d'avoir une équivalence entre modèle complet et ensemble de sous-modèles le composant, sous peine de nécessiter un type d'analyse de complexité exponentielle. S'agissant de systèmes temps réel, il est primordial que ce processus d'extraction soit conservatif, et que l'analyse menée sur le sous-modèle ne puisse pas être optimiste par rapport au modèle global. Ce principe sera présenté dans la section 6.2.

Ensuite, dans le chapitre 5, la construction d'un système distribué a été présentée. Un système distribué sous-entend la création de multiples sous systèmes, chacun avec ses caractéristiques. Or, les langages de modélisation actuels même s'ils se focalisent sur un domaine précis se composent d'un grand nombre d'artefacts alors que comme vu dans la section 3.7 certains sous-systèmes ont besoin de peu d'artefacts de modélisation. De même, dans un contexte industriel, la démarche proposée par MoSaRT, qui consiste, à partir du modèle d'entrée, à proposer des analyses applicables, peut se révéler infructueuse, notamment pour des ingénieurs peu expérimentés avec la conception logicielle de systèmes pouvant être validés.

En effet, si un modèle MoSaRT ne peut pas être analysé, dans l'outillage MoSaRT, il est possible d'envisager d'informer au mieux l'utilisateur de la cause, par exemple en expliquant quels contextes s'approchant au mieux du modèle d'entrée sont analysables, espérant ainsi guider l'architecte système dans ses modifications. Une autre possibilité de permettre à l'architecte système de concevoir un système analysable est de restreindre ses choix de conception. Dans un contexte temps réel, l'une des premières démarches dans ce sens pourrait être attribuée à la proposition du profil Ravenscar [Bur99]. On trouve aussi cette philosophie dans les choix opérés statiquement dans l'outil Tempo, ou encore, toujours statiquement, dans les démarches de type patrons de conception. Nous pourrions choisir, statiquement, de restreindre MoSaRT, en offrant ainsi, statiquement, un ensemble de patrons de conception, cependant, nous pensons plus intéressant de généraliser la démarche en proposant une méthode de création dynamique de patrons de conception. Ceci pourrait par exemple être utilisé dans des services de conception industrielle, en fonction des outils qualifiés existants, ou de la philosophie de conception d'applications utilisée par l'entreprise ou le service.

Par exemple, un système d'actionneurs demandant peu de ressources sera localisé sur un monoprocesseur tandis qu'un calculateur embarqué nécessitant plus de puissance utilisera plutôt un système distribué local ou bien un multiprocesseur. Ces sous-systèmes cohabitent dans un même système, parfois de la taille d'un avion : les parties-prenantes au système sont donc nombreuses.

La manipulation de tels langages exige une formation poussée sur tous les éléments du langage même si, au final, l'architecte logiciel n'utilisera qu'une petite partie de ce langage. Cela peut même aller jusqu'à générer une dérive en terme d'ambiguïtés sémantiques et une non-correspondance avec la spécification du cahier des charges en cas d'utilisation par un expert. Ainsi, par exemple, dans Time4sys, ce n'est qu'une petite partie de UML - MARTE qui est utilisée comme base du langage pivot, et dans les langages de conception basés sur UML - MARTE, qui propose plus d'une centaine de concepts, seule une à deux dizaines de concepts sont repris. Ce type d'inconvénient se trouve dans tout langage très riche et très large. En enrichissant MoSaRT, nous l'avons aussi rendu plus complexe, même s'il n'adresse toujours que deux métiers : l'architecture système et l'analyse temporelle. UML - MARTE quant à lui est très riche, puisqu'on peut l'utiliser pour la modélisation des NoC, ou même la synchronisation d'horloge, ou bien le calcul de durées d'exécution, comme pour la modélisation des systèmes temps réel en vue de l'analyse d'ordonnançabilité. Cette complexité intrinsèque au langage est donc encore plus grande sur UML - MARTE que sur MoSaRT, mais nous pensons que les méthodes que nous proposons dans ce chapitre pour créer dynamiquement des patrons de conception peuvent être généralisées

à d'autre langage que MoSaRT, sur lequel nous appliquons dans cette thèse ces méthodes.

6.2 Extraction conservative d'un sous-système

Les systèmes deviennent de plus en plus grands et donc de plus en plus complexes. L'analyse de ces systèmes peut prendre ainsi beaucoup de temps.

6.2.1 Contexte et objectifs

La complexité des systèmes est très souvent liée à l'ajout de fonctionnalités supplémentaires dans un système. L'analyse de grands systèmes est aussi rendue compliquée comme dans le cadre d'affectations de tâches sur un système multiprocesseur, problème NP complet et implique ainsi un temps de calcul élevé pour déterminer l'affectation de tâches. L'analyse holistique, présentée dans la section 2.4.2.3, permet de calculer les temps de transmission sur les réseaux et plus globalement sur les systèmes en entier. Cette analyse calcule les temps de réponse des tâches de façon locale – sur un processeur par exemple –, et ainsi le résultat est réutilisé pour les systèmes dépendants de cette analyse locale. Les analyses locales peuvent ainsi être faites par différents logiciels et au final, il est possible de combiner les analyses pour donner le résultat global pour le système. Le principe d'analyse compositionnelle étudiée par Richter *et al.* [RZJE02, Ric05] ne permet que la distribution de l'analyse : les analyses des sous-système sont déterminées dans un seul logiciel, SymTA/S ou pyCPA.

Lors du développement d'un nouveau système, il serait également possible de partir d'un système conçu auparavant sur lequel on ajoute un sous-système supplémentaire. Par exemple, dans le cas d'un drone, on ajoute un processeur communiquant avec le système déjà analysé temporellement. C'est le travail de l'ingénieur intégrateur : il fait l'intégration de plusieurs sous-systèmes créés par autant de parties prenantes.

Il s'agit donc de créer des sous-systèmes à partir d'un modèle de système dans lesquels l'analyse pourrait être faite de façon indépendante des autres sous-systèmes comme le montre la figure 6.1. Sur cette figure, cinq tâches sont présentes sur un système réparti sur deux processeurs. Aussi, la tâche τ_5 est activée par la tâche τ_2 et la tâche τ_4 active la tâche τ_3. Arbitrairement, on suppose que la priorités des tâches est fixée et que les tâches ne migrent pas entre les processeurs.

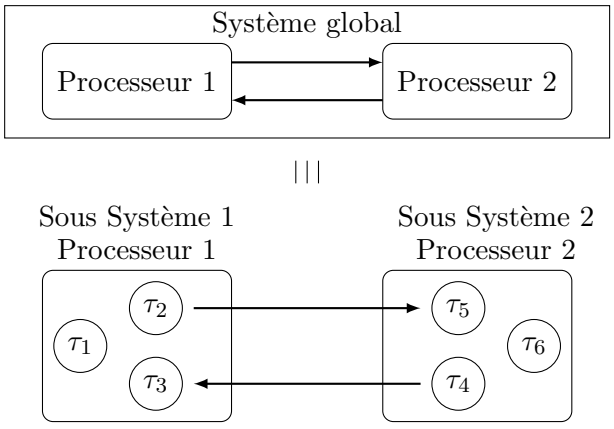


FIGURE 6.1 – Principe de sous-systèmes

6.2.2 Extraction d'un sous-modèle

Comme vu dans la section 3.7.1, il est possible de découper des modèles afin de conduire des analyses de façon séparée. Dans le domaine temporel, on peut envisager de diviser les modèles de système de différentes manières :

- par processeur,
- par multi-processeur,
- par sous-système fonctionnel qui peut comprendre plusieurs processeurs.

Pour conduire l'analyse correctement, une fois la division de modèles terminée il faut qu'un processus n'appartienne qu'à un seul sous-modèle. Autrement dit, soit $\mathcal{P}$ l'ensemble des processeurs du système tel que $\mathcal{P} = p_1, p_2, ..., p_n$ et $\mathcal{M} = m_1, m_2, ..., m_3$ l'ensemble des sous-modèles du système. Soit une fonction f injective telle que $f : \mathcal{P} \rightarrow \mathcal{M}$. On a alors $\forall i f(p_i) = m_j$ mais aussi $f^{-1}(m_j) = p_i, ..., p_k$.

Processus de découpage Pour extraire un sous-modèle, il faut s'assurer que cette extraction ne va pas entraîner une analyse optimiste. On propose donc de diviser un modèle de façon conservative. Pour ce faire, il faut, lors de l'analyse, récupérer le pire cas de chaque sous-système : le pire temps de réponse est d'abord déterminé pour toutes les tâches, en supposant une gigue nulle pour les messages en entrée. Ensuite, il faut propager les temps de réponse de la même manière que lors de l'analyse holistique présentée dans la section 2.4.2.3. En effet, cette analyse permet le calcul du temps de réponse de bout en bout avec l'aide d'analyses "locales" sur les sous-systèmes. Cette analyse permet de déterminer localement avec les analyses les moins pessimistes possibles et ainsi rendre l'analyse globale moins pessimiste.

Dans l'exemple de la figure 6.1, on propose de partager le système par processeur. Sur la figure, on assimile les processeurs aux sous-systèmes, il est possible qu'un sous-système contienne plusieurs processeurs.

Lors de l'analyse locale sur le sous-système 1, il ne nous reste que les tâches τ_1, τ_2, τ_3. Le pire temps de réponse de la tâche τ_2 sera utilisé comme gigue dans le sous-système 2, sur le même principe que la figure 2.8. Inversement, la tâche τ_3 aura une gigue d'activation provenant de l'analyse du sous-système 2, lors du calcul du pire temps de réponse de la tâche τ_4. Les éléments en entrée et sortie du sous-système 1 sont résumés dans la figure 6.2.

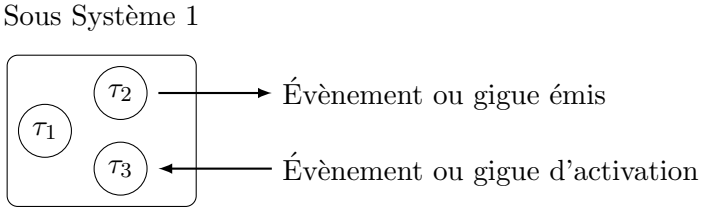


FIGURE 6.2 – Entrées-sorties du sous-système 1

6.2.3 Application sur MoSaRT

Pour appliquer notre démarche de découpage de modèles, on la considère sur un exemple simple. Cet exemple est représenté dans le langage MoSaRT. Le schéma de l'exemple est présenté dans la figure 6.3. On considère un ensemble de six tâches réparties sur deux processus, chacune localisé sur un processeur. Ces deux processeurs peuvent communiquer à l'aide d'un réseau. Les

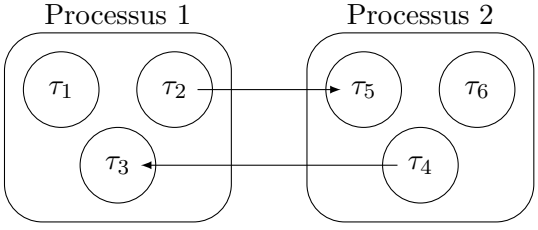


FIGURE 6.3 – Schéma de l'exemple considéré

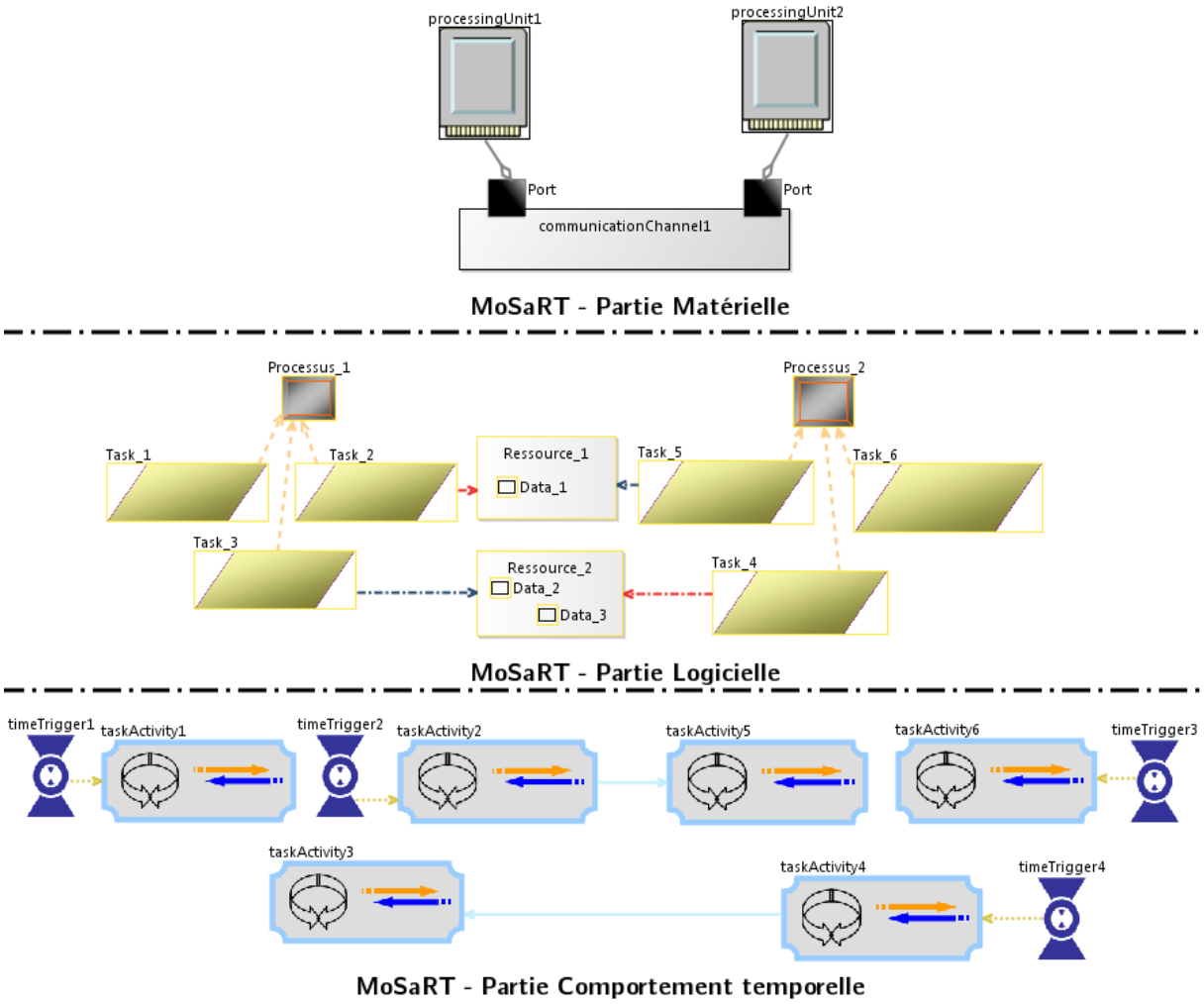


FIGURE 6.4 – Exemple modélisé sous MoSaRT

tâches 1, 2, et 3 sont localisées sur le processus 1 et les tâches 4, 5 et 6 sont localisées sur le processus 2. La tâche 2 envoie une donnée à la tâche 5 qui attend la réception de la donnée pour s'exécuter. Dans le sens inverse, la tâche 4 envoie une donnée à la tâche 3, cette dernière attend le message pour s'exécuter. Les tâches 1, 2, 3, et 4 sont activées périodiquement.

La modélisation sous MoSaRT est présentée sur la figure 6.4. Pour tirer des sous-modèles du modèle MoSaRT, on exécute le processus de découpage développé sous forme de code Java. L'utilisateur doit déterminer les sous-systèmes et donc ces sous-systèmes sont composés comme on a vu dans la section 6.2.2. L'algorithme sous forme de pseudo code est présenté sur le listing 6.1. Avec cet algorithme, les classes contenues dans les processeurs retenus par l'utilisateur sont renvoyés dans un nouveau modèle, conforme à MoSaRT. Les tâches qui ont un prédécesseur contenu dans un autre processeur que ceux retenus par l'utilisateur sont activées de la même façon que le prédécesseur, en tenant compte d'une gigue d'activation comme vu dans la section précédente.

Listing 6.1 – Pseudo-code de l'algorithme du découpage de modèles MoSaRT

```
retainedProcessor = {processor1, processor2,...,processor_n}
model = new Model();
for k in 1..n loop
   nb_classes = processor(k).schedulableTasks.size();
   for i in 1..nb_classes loop;
```

```
     if (class(i).hasAPredecessor()
        and not class(i).previousProcess.includedIn(retainedProcesses) ) then
        tempTrigger = getPreviousTrigger();
        trigger = new SbTimeTrigger();
        trigger = tempTrigger;
        class(i).representedActivity.SbSequencingRelation.replaceBy(trigger);
     end if;
      addInModel(model,class(i));
   end for;
end for;
```

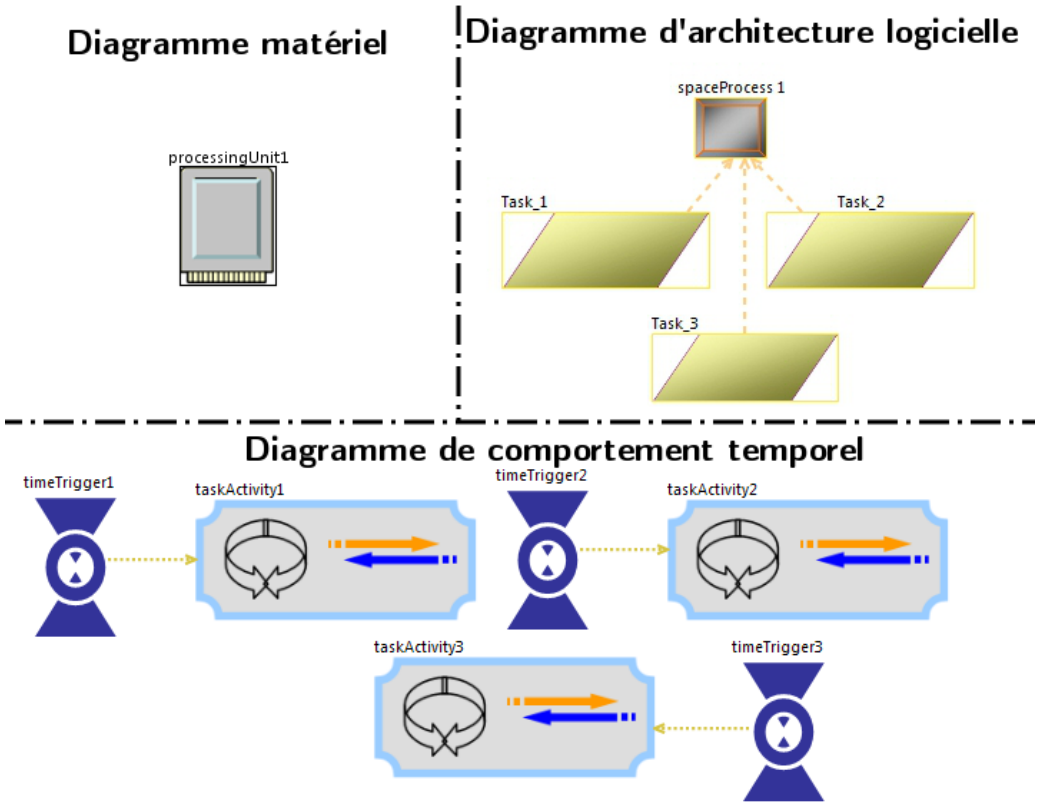


FIGURE 6.5 – Sous modèle du processus 1 extrait sous MoSaRT

6.3 Contexte de découpage de méta-modèles

La suite de ce chapitre présente notre contribution basée sur les *Feature models* [CN02] afin de créer dynamiquement des patrons de conception extraits du méta-modèle MoSaRT. L'idée est ici de permettre à des utilisateurs avancés, de définir des patrons de conception correspondant aux habitudes, et aux outils (potentiellement qualifiés), utilisables dans le service.

6.3.1 Constats sur le développement actuel

Les langages de conception sont très souvent manipulés par le biais de la syntaxe concrète du langage. En effet, cette syntaxe est la plus facile à manipuler : les objets ainsi que les relations entre ceux-ci sont gérés par l'environnement de développement. Cet environnement est souvent développé par le créateur du méta-modèle, celui-ci connaissant le méta-modèle et ses subtilités. Le concepteur du logiciel temps réel ne connaît pas forcément la syntaxe abstraite du langage. En effet, cette syntaxe est difficile à comprendre, voire à mettre en place, car il faut connaître

les outils utilisés pour établir le méta-modèle. Par exemple, le langage MoSaRT utilise les outils Ecore, OCL, Sirius, EEF, des outils pas nécessairement connus de tous.

6.3.2 Objectifs du découpage de méta-modèle

Plus un langage est vaste, plus il est difficile de le manipuler. Il s'agit donc de permettre de restreindre les éléments du méta-modèle à un sous-ensemble afin de contraindre les utilisateurs à un certain type de conception d'architecture.

L'architecte logiciel temps réel est spécialiste dans un domaine particulier. Ce domaine sera privilégié par l'architecte et il fera en sorte de faire correspondre les éléments du langage de modélisation avec son domaine. Réduire le champ des possibilités pour l'architecte permet d'éviter les erreurs dans la modélisation comme les non-sens et ainsi éviter l'allongement du temps de développement. Il permet aussi de limiter les risques de se trouver face à une conception de système non analysable, ou bien non analysable par les outils disponibles dans le service.

L'analyste temps réel connaît à l'avance les analyses qu'il peut appliquer, et les outils d'analyse existants. Il peut exprimer son besoin par rapport au test d'ordonançabilité en exprimant les conditions d'applicabilité du test. Par exemple, il peut exprimer des contraintes sur les relations entre les tâches, les caractéristiques temporelles des tâches, la présence d'un réseau ou non, etc.

Il s'agit donc, que ce soit du côté de l'architecture logicielle ou bien du côté de l'analyse temporelle, que l'utilisateur puisse être en mesure de définir ses besoins facilement pour obtenir les artefacts de modélisation qui répondent à ces besoins, et seulement eux.

6.3.3 Positionnement par rapport aux travaux existants

Des outils d'élagage de méta-modèles existent pour réduire le champ d'action des méta-modèles. Ces outils ont été présentés dans la section 3.7.2. Or, ils exigent que l'utilisateur connaisse parfaitement le méta-modèle. Pour établir un sous-méta-modèle, il est donc nécessaire de connaître la signification de chaque objet et la conséquence de la suppression de chacun de ceux-ci. À moins que l'utilisateur ne soit expert dans le langage à élaguer, l'utilisateur ne peut créer lui-même le sous-méta-modèle en évitant de façon sûre les interprétations erronées du langage.

6.4 Approche de réduction de méta-modèles

Dans cette section, l'approche utilisée pour établir des patrons de conception est présentée. Même si nous proposons cette approche pour adresser la construction dynamique de patrons de conception dans le contexte de MoSaRT, cette approche, si appliquée par exemple à UML - MARTE, pourrait permettre de restreindre un sous-ensemble du langage à un point de vue particulier, comme le point de vue analyse temporelle par exemple. C'est pour cette raison que nous parlons par la suite de points de vue. Ceux-ci, dans le cadre applicatif choisi dans cette thèse, correspondent à des patrons de conception. De façon générale, il s'agit de proposer des points de vue adaptés aux besoins de l'utilisateur. Dans la suite, on appellera **point de vue**, un sous-ensemble d'un langage de modélisation.

6.4.1 Démarche de création de sous-méta-modèles

Dans le domaine temporel, il existe de multiples paramètres disponibles pour construire un système temps réel (cf. section 2.3.3). Il s'agit de construire un lien entre les éléments de contexte voulu par l'architecte système et le langage. Pour ce faire, on propose à l'utilisateur un langage pour exprimer ses besoins. Dans le cadre des systèmes temps réel critiques, ses besoins s'expriment en termes d'architecture logicielle, d'architecture matérielle ainsi qu'en termes d'aspects de comportement temporel. Comme on a vu dans la section précédente, la syntaxe

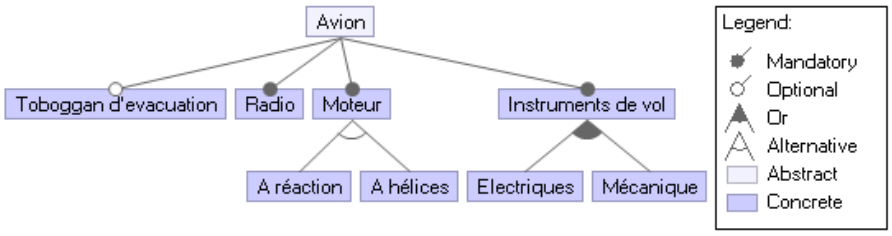


FIGURE 6.6 – Exemple de *Feature Model* composant un avion

abstraite est difficile à manipuler pour le novice. Le langage d'expression des besoins doit également être capable de vérifier la cohérence des choix effectués par l'utilisateur, à l'image des règles structurelles OCL dans le langage Ecore. Cette vérification de la cohérence évite ainsi une mauvaise modélisation du système.

Les besoins exprimés dans ce langage doivent être transposés dans le langage de modélisation. Une fois les besoins exprimés, cette transposition indique les classes du méta-modèle qui doivent être retenues dans le sous-méta-modèle. Ainsi, il est possible de construire un sous-méta-modèle, dérivé du méta-modèle initial, contenant uniquement les éléments dont l'utilisateur a besoin.

6.4.2 Aspect temporel de l'élagage de méta-modèles

Dans un langage, chaque objet implique l'instanciation d'autres objets. Par exemple, un réseau exige plusieurs processeurs pour être utile ou bien un commutateur doit être instanciée dans un réseau. Réciproquement, plusieurs processeurs peuvent communiquer sans réseau (dans le cadre d'un système multiprocesseur) ou encore un bus n'a pas besoin de contenir des commutateurs. Pour représenter ces besoins, on propose d'utiliser la composante *Feature model* tiré des méthodes *Software Product Line* (SPL) [CN02].

6.4.3 Présentation du *feature model*

Inspiré des lignes de production industrielles, le SPL est un ensemble de méthodes d'ingénierie logicielle pour créer de multiples logiciels qui partagent un socle commun de composants. Ce socle commun permet de gérer les différentes versions logicielles, par exemple, entre les différentes plateformes Android, Apple et Windows Phone ou bien entre deux versions comprenant différentes fonctionnalités. Ce processus se place donc lors de la conception du logiciel.

On tire du SPL la méthode des *Feature Model*. Cette méthode permet de tirer les similitudes ainsi que les différences entre les systèmes. Elle se présente sous la forme d'un arbre sur lequel un nœud représente une caractéristique du système. Ses feuilles sont sélectionnées selon les caractéristiques voulues par l'architecte logiciel.

Un nœud père peut comporter plusieurs nœuds enfants. Un nœud peut être soit obligatoire, soit facultatif, soit en "ou" exclusif (un et un seul nœud parmi les enfants doit être sélectionné), soit en "ou" (au moins un nœud doit être sélectionné). Si un nœud enfant est sélectionné, tous les nœuds parents doivent être sélectionnés. Un exemple simple d'arbre est présenté en figure 6.6.

Le nœud père racine, un avion, est nécessairement composé d'une radio, de moteurs et d'instruments de vol. Le toboggan d'évacuation n'est qu'optionnel, en particulier pour de petits avions. Le moteur peut être soit à réaction soit à hélices mais pas les deux en même temps. Les instruments de vol peuvent être mécanique (ou analogique) et/ou électriques, les deux types d'instruments peuvent coexister. Enfin, il n'est pas possible de sélectionner un moteur seul et des instruments de vol seuls : ce sont des éléments abstraits.

6.4.4 Application aux systèmes temps réel

À l'aide de ce langage, on propose de créer un dictionnaire de besoin adapté à l'analyse des systèmes temps réel. Pour les systèmes temps réel, on propose d'utiliser la composante *Feature model* pour déterminer les variantes logicielles avant la modélisation. Ainsi, lors de la spécification du système, l'architecte logiciel doit déterminer les besoins non-fonctionnels du logiciel.

Dans de grands systèmes, cela se traduit par la construction de plusieurs sous-modèles créés par autant de parties-prenantes, spécialistes dans leur domaine. Donc, contrairement à l'extraction conservative de sous-modèles de la section 6.2, il s'agit ici de déterminer les sous-méta-modèles pour que l'utilisateur construise les modèles à partir de ce modèle.

Dans la suite, nous allons donc déterminer les caractéristiques communes ainsi que les caractéristiques variables des systèmes temps réel. À partir de ces caractéristiques, un *Feature model* est construit et selon la sémantique requise par le concepteur, celui-ci sera utilisé pour élaguer le méta-modèle. Nous proposons donc un **élagage de méta-modèle dirigé par la sémantique des systèmes temps réel**.

Chaque nœud présent dans le *Feature model* correspondra à un ensemble de classes du méta-modèle nécessaire à la modélisation du système. Enfin, le sous-méta-modèle obtenu servira ensuite de base pour le logiciel de modélisation.

6.5 Construction du *feature model*

Dans cette section, nous allons formaliser l'utilisation du modèle descriptif de besoins à l'aide d'un *Feature model*. Ensuite, à partir du modèle de besoin, nous allons faire correspondre un langage de modélisation aux besoins exprimés par le modèle de besoin (i.e. dictionnaire).

6.5.1 Formalisation

Afin de représenter les systèmes, cette section définit les concepts que nous allons utiliser dans la suite.

On note un nœud du modèle de besoin n_i. Le nœud correspond à un ensemble de classes du méta-modèle, notées chacune c_i. On obtient alors : $n_i = \{c_1, c_2, ..., c_N\}$. Ces classes correspondent à celles qui doivent rester dans le méta-modèle pour le nœud considéré. Chaque nœud doit répondre à un besoin particulier exprimé par l'utilisateur. Deux besoins différents ne doivent pas se recouper, c'est-à-dire, qu'aucune classe représentée par ces besoins ne sont identiques entre elles. Si une classe est représentée par deux besoins, cette classe doit être représenté par un besoin père à ces deux besoins. On a alors pour chaque nœud : $\forall i \neq j, n_i \cap n_j = \emptyset$. Le méta-modèle complet sans élagage est lorsque tous les nœuds doivent être sélectionnés et dans le cas de nœuds avec alternative, un nœud doit être spécifié comme étant le cas général.

Pour la suite, si un nœud est sélectionné sans que les nœuds enfants ne soient sélectionnés, aucune des classes liées avec les nœuds enfants ne sont instanciées.

On note $\mathcal{S}$ l'ensemble des nœuds correspondant aux besoins de l'utilisateur. $\mathcal{S}$ est donc tel que $\mathcal{S} = \{n_1, n_2, ..., n_k\}$. Le sous-méta-modèle $\mathcal{M}$ correspondant à l'ensemble des nœuds est alors $\mathcal{M} = \cup_{i \in \{1..k\}}\ n_i = \{c_1, ..., c_M\}$

6.5.2 Développement et généralisation

Dans cette section, nous allons détailler les différentes parties exprimées en *feature model*. On utilisera le plug-in Eclipse *FeatureIDE* [KTS+09] pour créer le modèle de besoins temps réel.

Un système temps réel peut se décomposer en trois grandes parties :

— La partie logicielle, composée, entre autres, de tâches, de processus, et de ressources critiques,

— La partie matérielle sur laquelle les tâches s'exécutent et transmettent leurs messages,

— La partie comportement temporel, comportant les paramètres temporels du système, comme ceux décrits dans la section 2.3.3. Cette partie permet de déterminer le comportement temporel du système.

La figure 6.7 propose un *feature model* pour les systèmes temps réel. Sur cette figure apparaissent les trois grandes parties d'un système.

Dans la partie matérielle, un réseau peut être instancié, sous la forme d'un bus ou bien sous la forme d'un réseau commuté (noté *Switched* sur la figure). Les processeurs du système peuvent être tous des monoprocesseurs ou des multiprocesseurs homogènes, hétérogènes ou uniformes.

Dans la partie logicielle, il est possible de choisir le nombre de processus dans le système. Notons au passage que suivant la plateforme cible, il n'est pas toujours possible de créer plusieurs processus sur un unique processeur. Par exemple, le système d'exploitation temps réel VxWorks offre la possibilité de créer plusieurs processus sur un même processeur. Cependant, POSIX, dans son profil minimaliste PSE 51 [IEE03] souvent utilisé pour le temps réel, ne propose qu'un processus, de même qu'AUTOSAR/Osek [C$^+$13] très utilisé dans le domaine automobile.

Dans le cas de plusieurs processus, les processus peuvent être hiérarchisés ou non. Les tâches peuvent soit interagir entre elles, soit être indépendantes entre elles. Les ressources peuvent être protégées par un sémaphore d'exclusion mutuelle, ou bien ne pas exister. Les tâches peuvent transmettre des messages de façon locale ou par le biais d'un réseau. Dans ce dernier cas, le réseau doit être sélectionné, et réciproquement un message réseau doit être transmis si un réseau est présent.

Dans la partie ordonnancement, le comportement temporel des tâches peut être décrit en utilisant une date de réveil, un délai critique, une période. Les tâches peuvent être préemptibles ou non.

Dans ce modèle descriptif de besoins, on note également la présence de contraintes non exprimables directement dans l'arbre. Ces contraintes lient des nœud se trouvant sur des branches différentes. Ici, lorsqu'un réseau est demandé dans la partie matérielle (nœud *Network*), les messages réseau doivent également apparaître dans la partie logicielle (nœud *NetworkMessage*) et réciproquement.

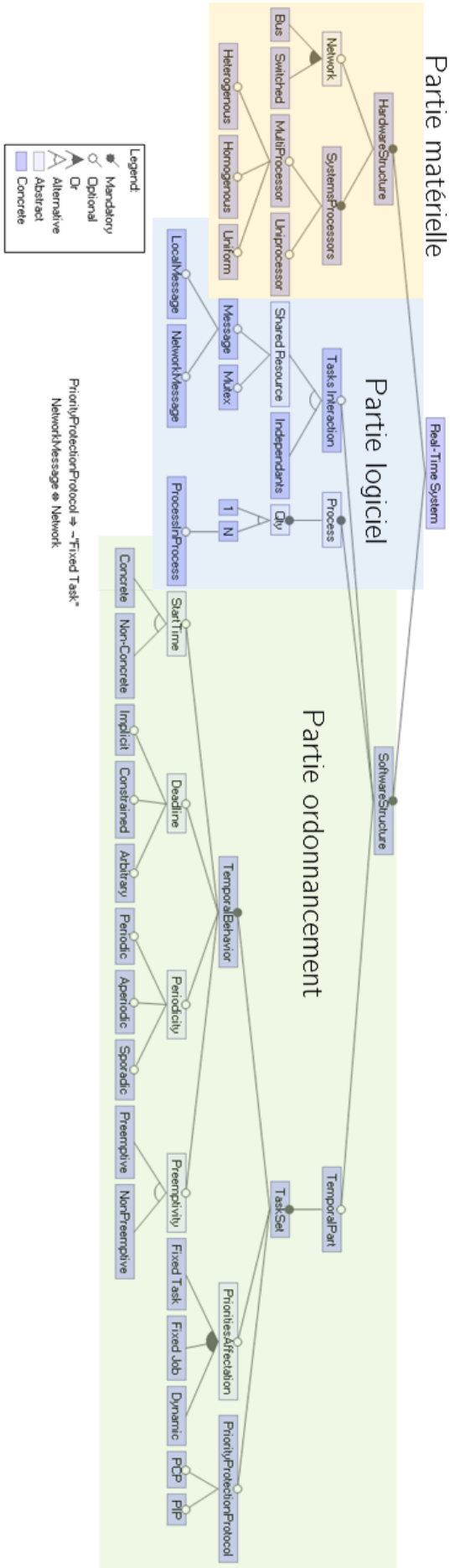


FIGURE 6.7 – Modèle de besoins (ou *Feature Model*) pour les systèmes temps réel

6.5.3 Transpositions des besoins dans le langage de modélisation MoSaRT

Une fois que les besoins sont définis, il est nécessaire de faire une correspondance avec le langage de modélisation. On a proposé un modèle de besoins permettant d'identifier les besoins du système temps réel conçu. Ces besoins doivent être utilisés pour créer un point de vue du langage de modélisation. On propose une plateforme permettant d'élaguer les langages adaptés au temps réel à l'aide du modèle de besoin. Nous allons élaguer le langage MoSaRT.

A chaque nœud, on fait correspondre un ensemble de classes à garder. Le *feature Model* s'inspire ainsi fortement des contextes du référentiel d'analyses. Pour représenter le besoin exprimé dans le méta-modèle, on propose de rassembler toutes les correspondances entre le nœud et le méta-modèle dans un **référentiel de mapping**. Le référentiel consiste à stocker toutes les correspondances dans un modèle dépendant du méta-modèle cible. Ainsi, on propose dans la figure 6.8, un méta-modèle correspondant à ce référentiel. Sur cette figure, le `MappingDatabase` est la classe racine du modèle. Cette figure exprime les relations entre le besoin (noté `Feature` sur la figure) et les classes, attributs et références Ecore (notées respectivement `Eclass`, `EAttribute`, et `EReference` sur la figure. Les éléments du méta-modèle Ecore liés au besoin doivent être dans le point de vue extrait. Chaque `Feature` décrit le besoin exprimé dans la classe `Relation`.

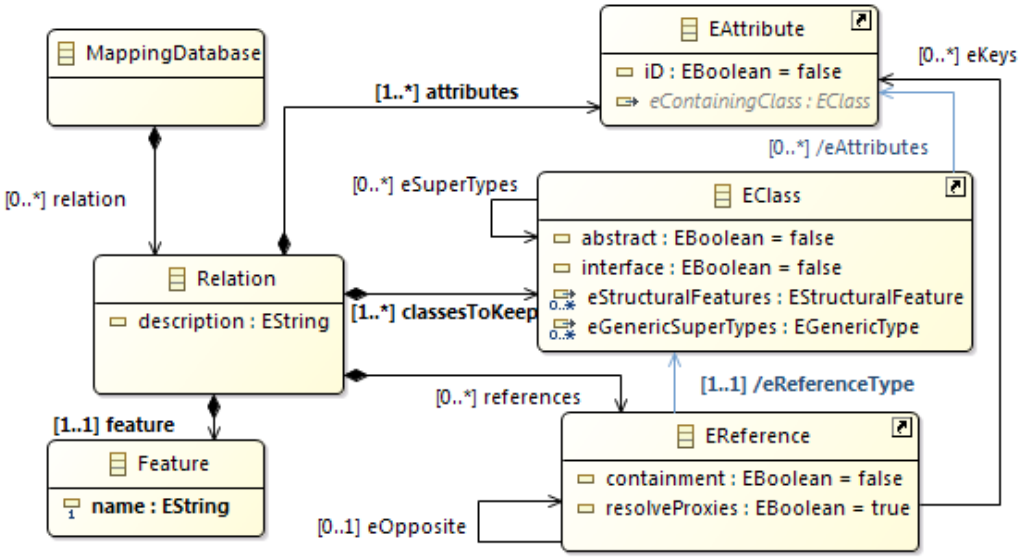


FIGURE 6.8 – Méta-modèle du mapping

Une instance de correspondance est montrée sur la figure 6.9. Sur cette figure, on ne présente que la partie correspondant à l'extraction de tâches périodiques, préemptives sur le méta-modèle MoSaRT.

En sélectionnant les nœuds *Periodic* et *Preemptive*, les nœuds pères (*Periodicity*, *Preemptivity*, *TemporalBehavior*, *TaskSet* et *TemporalPart*) sont automatiquement sélectionnées. À chacun de ces nœuds, on doit affecter des classes qui doivent être retenues dans le sous-méta-modèle (le point de vue). Le nœud *TaskSet* indique la présence de tâches : la classe `SoSchedulableTask` (cf. figure 3.13) est donc sélectionnée. Le nœud *TemporalBehavior* indique que des les tâches ont un comportement temporel : le conteneur des éléments comportementaux et l'activation des tâches doivent être retenus (respectivement les classes `GlobalBehavior` et `SbTimeTrigger` de la figure 3.14).

Une activation peut être soit périodique, sporadique ou apériodique. Le nœud *Periodic* caractérise uniquement l'activation périodique d'une tâche, lorsqu'elle est sélectionnée, Donc, si les tâches sont uniquement périodiques, seul le nœud *Periodic* doit être sélectionné. En revanche, si les tâches peuvent être activée de différentes façons, les autres nœuds doivent être sélectionnés.

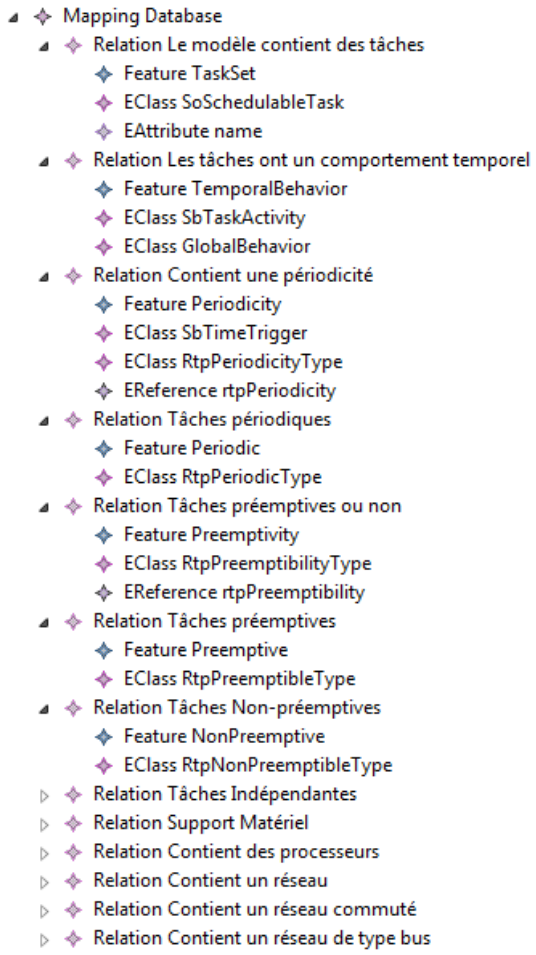


FIGURE 6.9 – Extrait d'une instance de *mapping*

6.6 Intégration - Exemple

Dans cette section, nous intégrons cette plateforme à l'aide d'outils existants pour permettre la génération et l'utilisation de points de vue sur un langage. Ce processus est ensuite appliqué sur un cas simple de point de vue.

6.6.1 Intégration dans un processus de génération d'un point de vue

Le processus appliqué pour construire un point de vue est présenté sur la figure 6.10. À partir du modèle de besoins (c'est-à-dire à partir de l'ensemble des éléments du dictionnaire que nous avons proposé sour forme de *Features Model*), l'expert crée les correspondances à l'aide du langage créé dans la figure 6.8. Cela donne un *mapping* similaire à la figure 6.9. Ce mappint est fait une seule fois pour chaque langage. Ensuite, le concepteur logiciel peut définir ses besoins à l'aide du *Feature Model*. Une fois que les besoins sont exprimés, on utilise l'outil d'élagage de méta-modèles. Cet outil donnera un méta-modèle élagué pour créer un sous-méta-modèle à partir du mapping créé par l'expert du méta-modèle et aussi des besoins exprimés par le concepteur logiciel.

L'outil du processus d'élagage que nous avons créé est détaillé dans la figure 6.11. À partir des besoins de l'utilisateur et du mapping, il est possible de déterminer les classes à extraire du méta-modèle, et inversement à retirer du méta-modèle, à l'aide d'un parseur Java, dans l'optique d'utiliser ATL car il est doté d'une option de raffinement ne possédant que des fonctions de suppression de classes (c'est-à-dire les classes qui n'ont pas été sélectionnées).

Une fois la liste des classes à retirer du méta-modèle établie, l'outil Acceleo [MJL+12] permet

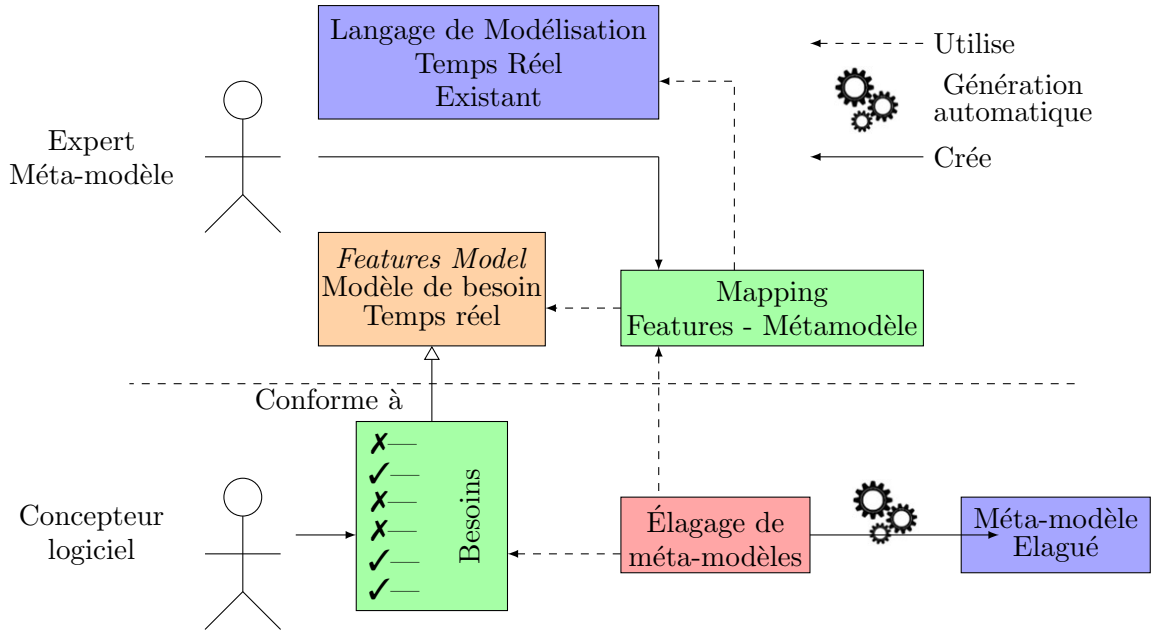


FIGURE 6.10 – Processus global du système

de générer automatiquement du code à partir de ce modèle. Le code Acceleo utilisé pour générer le code ATL est donné dans le listing 6.2. Un extrait du code ATL généré est donné dans le listing 6.3.

Listing 6.2 – Instanciation Acceleo

```
[template public generateATL{counter : Integer = 1;}]
-- nsURI Ecore =uri:http://www.eclipse.org/emf/2002/Ecore
-- @atlcompiler atl2010
module UserDefined_Pruning;
create OUT: Ecore refining IN: Ecore;
[for () {counter : Integer = counter + 1;} ]
rule DeleteClass[counter/]{
   from s : Ecore!ENamedElement (s.name.startsWith('
   [class.name/]
   '))
   to drop
}
[/template]
```

Listing 6.3 – Extrait de code ATL généré

```
rule DeleteClass1{
   from s : Ecore!ENamedElement (s.name.startsWith('HpProcessorInterconnector'))
   to drop}
rule DeleteClass2{
   from s : Ecore!ENamedElement (s.name.startsWith('HpFlowCarrier'))
   to drop}
rule DeleteClass3{
   from s : Ecore!ENamedElement (s.name.startsWith('HpCommunicationSwitch'))
   to drop}
```

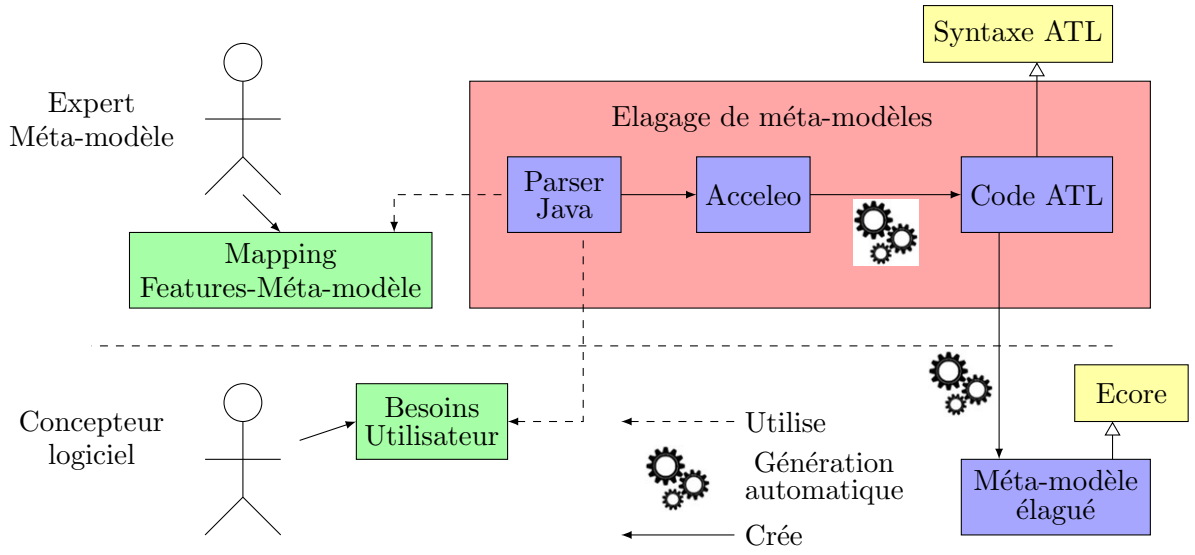


FIGURE 6.11 – Processus d'élagage détaillé du système

À partir de ce code ATL, il est possible de générer un sous-méta-modèle par le biais d'une transformation endogène de Ecore vers Ecore. Le sous-méta-modèle Ecore peut ensuite servir de base pour une génération de code Java pour la création d'un plug-in adapté au point de vue demandé par l'utilisateur.

6.6.2 Exemple d'application

Pour appliquer ce processus, on utilise le modèle de tâches de Liu & Layland. Ce modèle se compose d'un monoprocesseur sur lequel des tâches indépendantes s'exécutent, à l'intérieur d'un unique processus. En appliquant ces hypothèses sur le modèle de besoins, la configuration demandée est résumée dans la figure 6.12. Sur cette figure, on ne considère qu'un seul processus pour le système, l'activation des tâches est connue (*concrete* sur la figure), les tâches sont périodiques, à échéances implicites, préemptibles et les priorités sont fixes. Enfin, après traitement des choix sémantiques retenus, le méta-modèle MoSaRT résultant du processus est donné en figure 6.13.

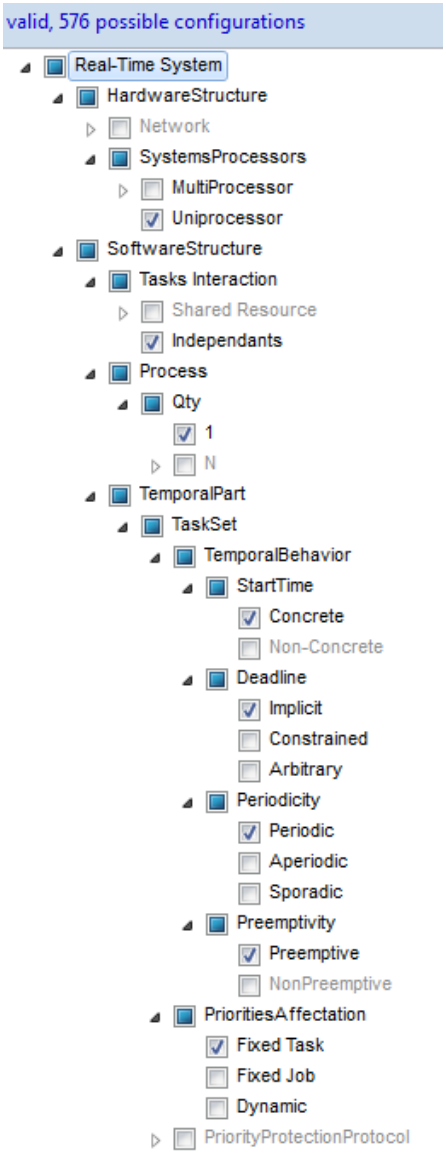


FIGURE 6.12 – Exemple de choix pour le modèle de tâche de Liu et Layland

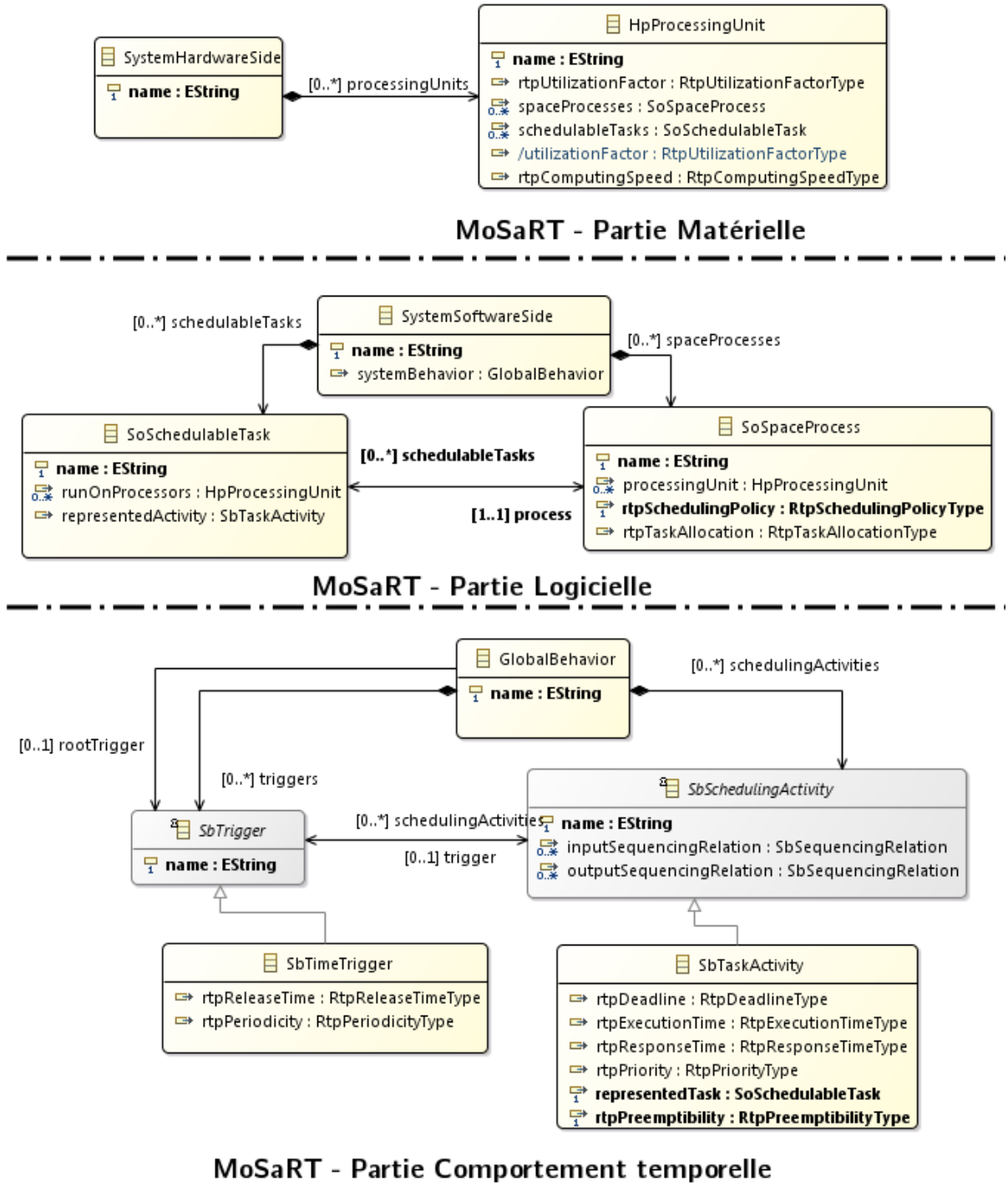


FIGURE 6.13 – Exemple de sous-méta-modèle pour le modèle de tâche de Liu et Layland

6.7 Conclusion

Dans ce chapitre, nous avons vu le principe de découpage de modèles au sens général du terme. Ce principe est appliqué pour des modèles MoSaRT existants pour étudier, par morceaux, le comportement temporel de sous-modèles. La distribution des analyses peut permettre d'affiner une analyse globale menée avec un seul logiciel d'analyse temporel. Ce découpage de modèles doit être effectué en conservant le pessimisme du modèle original sur les sous-modèles. Nous avons été amenés pour cela à nous baser sur l'analyse holistique.

Ensuite, le principe de découpage est appliqué aux méta-modèles afin de créer des points de vue liés au besoins du concepteur. Cette élagage de méta-modèle est adaptable grâce aux travaux de l'expert méta-modèles. En effet, le concepteur exprime ses besoins à l'aide d'une représentation simple lui évitant la manipulation de méta-modèles. À partir des besoins, l'outil d'élagage de méta-modèle, permet l'établissement d'un environnement de développement basé sur le sous-méta-modèle. Ainsi, le modèle créé sur la base du sous-méta-modèle (patron de conception) est conforme par construction au méta-modèle original.

Troisième partie

Conclusions

Chapitre 7

Conclusions et perspectives

Sommaire

Ce dernier chapitre résume les contributions apportées à l'ingénierie dirigée par les modèles appliquée à l'analyse des systèmes temps réel. En particulier, cette thèse contribue à rapprocher l'expertise nécessaire à l'analyse de performances des pratiques industrielles de conception d'architectures logicielles et matérielles, ceci afin de réduire les coûts de développement ainsi que la détection tardive des erreurs de conception, en permettant de se rapprocher de l'idéal qui serait une conception correcte par construction. Nous nous basons largement sur les technologies issues de l'ingénierie dirigée par les modèles, qui offrent un ensemble de méthodes et outils performants, en particulier lorsque l'on souhaite avoir une approche pluri-disciplinaire d'un problème.

Afin de mettre l'expertise de l'analyse de performances à disposition de l'architecte système, en charge de concevoir cette architecture de façon éclairée par l'analyse, cette thèse a proposé trois contributions : (1) l'*adaptation conservative de modèles* de tâches trouvés dans des cas concrets mais pas dans les modèles académiques ; (2) la modélisation de *réseaux* de terrain filaires en vue de leur analyse ; (3) la création de sous-modèles qui permettent (a) soit d'analyser un système par parties (b) soit en passant au niveau méta, de créer un *point de vue* particulier, ce qui s'est traduit sur MoSaRT par la possibilité de créer dynamiquement des patrons de conception.

7.1 Contributions au projet WARUNA

Cette thèse s'est déroulée dans le cadre du projet collaboratif FUI WARUNA entre 2015 et fin 2018, qui regroupait des partenaires industriels, et deux partenaires académiques. Ce projet a donné lieu à la proposition d'un langage basé UML - MARTE, étendant ce dernier par divers concepts manquants, ainsi que d'un ensemble d'outils, et de transformations de modèles endogènes et exogènes, dont CONSERT fait déjà partie. Nous avons mené les travaux de la thèse de la façon suivante : MoSaRT étant un langage totalement développé au laboratoire, nous avions la possibilité de le faire évoluer de façon unilatérale. De plus, MoSaRT est strictement plus riche sémantiquement sur les concepts liés à l'analyse temporelles que ses concurrents. Il nous a donc servi de laboratoire d'expérimentation et de prototypage pour nos propositions (telles que la première implémentation de CONSERT, la modélisation des réseaux, l'extraction de sous-modèles et la création de points de vue). Il faut noter que chaque proposition a été implémentée d'abord dans MoSaRT, mais que les méthodes et concepts proposés sont indépendants du langage de modélisation choisi. Elles peuvent donc facilement être transposées dans d'autres langages de description d'architecture orientés temps réel basés sur les technologies de l'ingénierie dirigée par les modèles, tels que Time4Sys, UML - MARTE, ou AADL par exemple. Nos implémentations dans MoSaRT avaient pour but de servir d'expérimentation laboratoire avant de pousser les extensions, ou méthodes, vers Time4Sys, pour lequel chaque nouvel artefact de modélisation ne pouvait être introduit qu'avec l'accord du consortium. Nous avons pu assez simplement intégrer CONSERT à la plateforme Time4Sys, et certains éléments réseau ont pu être repris, sous la contrainte forte de réutiliser autant que possible les artefacts de modélisation pré-existant dans UML - MARTE. Enfin, la partie extraction de sous-modèles pourra vraisemblablement être intégrée dans la chaîne d'outils développée pour Time4Sys. La dernière contribution du manuscrit sur la création de points de vue, qui s'avère aussi être la dernière de la thèse chronologiquement, arrive en fin de projet, et au jour de la rédaction de ce manuscrit, n'est pas intégrée dans Time4Sys.

7.2 Adaptation conservative de modèles

7.2.1 Évolutions proposées

Les modèles industriels différent beaucoup des modèles d'analyse d'ordonnancement et ces différences ne sont surmontées qu'avec l'aide de l'expert en analyses temporelles. Le *plug-in* CONSERT présenté dans le chapitre 4 propose la création d'un référentiel de transformations endogènes proposant automatiquement ces transformations. Ces transformations permettent le rapprochement du modèle industriel vers un modèle analysable exprimé dans le même langage tout en vérifiant que toutes les hypothèses nécessaires sont présentes dans le modèle d'entrée pour que la transformation soit conservative. La plateforme CONSERT permet ainsi la conservation des transformations pour les appliquer de façon correcte lors de la détection d'un contexte de transformation. Cette détection de contexte s'appuie sur des règles structurelles définies dans CONSERT.

Ce principe de transformation endogène permet la présentation du modèle adapté à l'analyse temporelle à l'utilisateur pour vérifier la transformation exécutée. Le modèle adapté présenté dans le même langage permet à l'utilisateur de comprendre les modifications effectuées sur le modèle.

L'implémentation réalisée de CONSERT l'a été dans le contexte de Time4Sys, bien qu'il soit possible simplement de la porter dans MoSaRT ou UML - MARTE.

7.2.2 Perspectives

L'une des perspectives immédiates de ce travail consiste en un enrichissement de CONSERT par d'autres transformations endogènes. Une autre perspective consiste à l'intégrer plus en profondeur dans le processus d'analyse en se basant notamment sur l'approche proposée dans [BNH17], qui consiste à orchestrer automatiquement des tests. Dans l'implémentation actuelle, c'est au concepteur de lancer la transformation et de vérifier les résultats de celle-ci, alors qu'elle pourrait être directement intégrée à une chaîne de transformations et d'analyses orchestrées.

7.3 Analyse de réseaux temps réel

7.3.1 Évolutions proposées

Nous sommes partis du constat que la représentation du réseau dans les systèmes distribués proposés par les différents langages de description d'architecture, y compris MoSaRT, n'était pas suffisamment expressive en vue de l'analyse de temps de traversée, dont nous avons dans l'état de l'art identifié les paramètres d'entrée.

Nous avons alors proposé une extension de MoSaRT dans le chapitre 5 respectant sa philosophie de modélisation, qui soit guidée par les analyses, c'est-à-dire qu'elle permet de ne modéliser que ce qu'il est nécessaire de modéliser en vue de l'analyse temporelle. De plus, nous avons proposé des artefacts de modélisation qui étaient communs, que ce soit pour les bus ou les réseaux commutés, de façon à permettre de réduire l'effort de re-modélisation lorsque le concepteur change de réseau.

Cette représentation a donné lieu à une extension du méta-modèle de MoSaRT (syntaxe abstraite) et à l'enrichissement de sa syntaxe concrète, en vue de faciliter la modélisation et la compréhension de la lecture d'un modèle. Nous avons appliqué cela à la modélisation d'un système distribué communiquant par un réseau CAN, que nous avons changé pour un réseau AFDX. Enfin, afin d'illustrer la démarche complète, de la modélisation à la validation, nous avons traité une étude de cas utilisant un réseau CAN qui a nécessité une analyse holistique.

Cette structure a été appliquée dans un langage de modélisation orienté analyses : MoSaRT. MoSaRT vise à être adaptable à tous les modèles d'analyse temporelle existants. Le langage MoSaRT donne la possibilité de conduire une modélisation incrémentale. La représentation réseau autorise toujours cette modélisation incrémentale en autorisant la réutilisation des éléments existants pour un autre type de réseau.

7.3.2 Perspectives

Pour le moment, peu d'outils d'analyse de temps de traversée réseau sont connectés à la plateforme MoSaRT. Il conviendra donc par la suite d'enrichir le référentiel d'analyse et d'ajouter des transformations vers les nouveaux outils d'analyse à connecter à MoSaRT comme les analyses internes des laboratoires. De plus, nous nous sommes limités aux réseaux de terrain filaires, alors qu'il est probable que les réseaux sans fil soient de plus en plus utilisés dans les systèmes temps réel. Cependant, étant donné qu'à notre connaissance, peu ou pas de réseau sans fil permettent un calcul de pire temps de traversée, nous n'avons pas souhaité, pour le moment, proposer de modèle. En effet, nos modèles sont proposés en fonction des analyses pouvant être appliquées, et en attendant l'émergence de méthodes d'analyse de temps de traversée dans les réseaux sans fil, nous nous sommes limités aux réseaux filaires.

7.4 Points de vues de systèmes temps réel

Étant donné que les outils d'analyse peuvent se limiter à certains contextes, il est parfois impossible, sur des systèmes complexes, de valider tout le système avec un unique outil existant.

En nous basant sur le principe de l'analyse holisitique, nous avons proposé dans le chapitre 6 une décomposition de modèle en sous-modèles, chacun pouvant être adressé par un ou plusieurs outils d'analyse, l'analyse holistique servant de glue conservative à l'analyse globale.

Ensuite, en partant du constat que pour un industriel, en particulier s'il était soumis à la conception de systèmes critiques devant être certifiés, la conception devait s'adapter à un modèle généralement restreint de conception d'une part, et devait d'autre part pouvoir être analysée par des outils qualifiés existants, nous avons conclu que la flexibilité et la richesse d'un langage tel que MoSaRT, ou MAST, pouvait être un inconvénient. Plutôt que de fournir des patrons de conception statiques, qui devraient être créés pour chaque usage, et donc potentiellement chaque entreprise et même service d'entreprise amené à utiliser ces langages, nous avons proposé une méthode de création dynamique de points de vue (qui peuvent exprimer des patrons de conception) basée sur le paradigme de variabilité qui peut être exprimé par les *Feature models*. Cela permet de créer simplement, sans maîtrise du méta-modèle sous-jacent, des points de vue qui extraient de façon consistante des éléments du méta-modèle initial, en créant un méta-modèle restreint. Nous espérons permettre aux utilisateurs, grâce à cela, de créer leurs propres points de vue, en fonction de leurs besoins particuliers.

7.4.1 Perspective : vers une ontologie des systèmes temps réel ?

Une ontologie de domaine est un modèle de données représentant les concepts de ce domaine ainsi que les relations entre eux. C'est une représentation qui doit être sémantiquement complète et sans ambiguïtés. Dans le domaine des systèmes temps réel, aucun travail de ce type n'a été répertorié malgré des travaux s'en approchant [1]. Un modèle de ce type dans le domaine temporel contribuerait à harmoniser le vocabulaire des langages orientés temps réel par exemple.

7.4.2 Vers de la génération de code logiciel

La génération de points de vues à l'aide d'un *feature model* permet de réduire les artefacts de modélisation qui sont superflus pour un objectif donné. En cela, en s'inspirant des restrictions du profil Ravenscar, ou en créant des restrictions spécifiques liées à la ou les cibles choisies, il serait possible d'étendre les possibilités de MoSaRT vers la génération de code, en faisant ainsi un langage orienté architecture permettant non seulement de mener l'analyse, mais aussi de générer du code. En effet, le risque d'ajouter des éléments nécessaires à la génération de code est de rendre le langage MoSaRT plus complexe, mais comme on pourra le restreindre simplement à l'aide de points de vue, soit à son rôle d'analyse, soit à son rôle de conception en vue de générer du code, le point de vue analyse pourra conserver une complexité similaire à la complexité actuelle du langage.

1. Avionique modulaire étendue - https ://aerorecherchecorac.com/

Bibliographie

[AB99] L. Abeni and G. Buttazzo. Qos guarantee using probabilistic deadlines. In *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99*, pages 242–249, June 1999. (Cité en page 18)

[ABJ01] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, pages 193–202, Dec 2001. (Cité en page 19)

[ABRW91] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Real-time scheduling : the deadline-monotonic approach. In *in Proc. IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137, 1991. (Cité en page 15), (Cité en page 31)

[AHSW62] James P. Anderson, Samuel A. Hoffman, Joseph Shifman, and Robert J. Williams. D825 - a multiple-computer system for command & control. In *Proceedings of the December 4-6, 1962, Fall Joint Computer Conference*, AFIPS '62 (Fall), pages 86–96, New York, NY, USA, 1962. ACM. (Cité en page 18)

[AJ00] Björn Andersson and Jan Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *Proc. of the IEEE Real-Time Systems Symposium, Work-in-Progress Session*, 2000. (Cité en page 19)

[ARI01] ARINC 429. Arinc specification 429. *Aeronautical Radio Inc*, 2001. (Cité en page 22)

[ARI03] ARINC. *ARINC 664, Aircraft Data Network, Parts 1,2 7. Technical report, ARINC specification 664.*, 2003. (Cité en page 22), (Cité en page 25), (Cité en page 30)

[Aud91] Neil Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, The Univetrsity of York, Novembre 1991. (Cité en page 17), (Cité en page 19), (Cité en page 20)

[Bar98] S. K. Baruah. Feasibility analysis of recurring branching tasks. In *Proceeding. 10th EUROMICRO Workshop on Real-Time Systems (Cat. No.98EX168)*, pages 138–145, June 1998. (Cité en page 44), (Cité en page 45)

[Bar03] Sanjoy K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1) :93–128, Jan 2003. (Cité en page 44), (Cité en page 45)

[Bar10] S. Baruah. The non-cyclic recurring real-time task model. In *2010 31st IEEE Real-Time Systems Symposium*, pages 173–182, Nov 2010. (Cité en page 44), (Cité en page 45)

[BB97] G. Bernat and A. Burns. Combining (/sub m//sup n/)-hard deadlines and dual priority scheduling. In *Proceedings Real-Time Systems Symposium*, pages 46–57, Dec 1997. (Cité en page 17)

[BB98] I. Bate and A. Burns. Investigation of the pessimism in distributed systems timing analysis. In *Proceeding. 10th EUROMICRO Workshop on Real-Time Systems (Cat. No.98EX168)*, pages 107–114, Jun 1998. (Cité en page 24)

[BB06] Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, RTSS '06, pages 159–168, Washington, DC, USA, 2006. IEEE Computer Society. (Cité en page 31)

[BBB03] Enrico Bini, Giorgio C Buttazzo, and Giuseppe M Buttazzo. Rate monotonic analysis : the hyperbolic bound. *Computers, IEEE Transactions on*, 52(7) :933–942, 2003. (Cité en page 17)

[BC06] T. P. Baker and M. Cirinei. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 178–190, Dec 2006. (Cité en page 19)

[BC08] Jung Ho Bae and Heung Seok Chae. Umlslicer : A tool for modularizing the uml metamodel using slicing. In *2008 8th IEEE International Conference on Computer and Information Technology*, pages 772–777, July 2008. (Cité en page 57)

[BCBB11] Arnaud Blouin, Benoît Combemale, Benoit Baudry, and Olivier Beaudoux. Modeling model slicers. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems*, pages 62–76, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cité en page 57)

[BCBB15] Arnaud Blouin, Benoît Combemale, Benoit Baudry, and Olivier Beaudoux. Kompren : modeling and generating model slicers. *Software & Systems Modeling*, 14(1) :321–337, Feb 2015. (Cité en page 57)

[BCGM99] Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1) :5–22, Jul 1999. (Cité en page 44), (Cité en page 45)

[BCL05] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of edf on multiprocessor platforms. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 209–218, July 2005. (Cité en page 19)

[BF05] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 9 pp.–329, Dec 2005. (Cité en page 19)

[BMR90] Sanjoy K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *[1990] Proceedings 11th Real-Time Systems Symposium*, pages 182–190, Dec 1990. (Cité en page 15)

[BNH17] Guillaume Brau, Nicolas Navet, and Jérôme Hugues. Heterogeneous models and analyses in the design of real-time embedded systems - an avionic case-study. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, RTNS '17, pages 168–177, New York, NY, USA, 2017. ACM. (Cité en page 48), (Cité en page 128)

[Bos91] Robert Bosch. Can specification version 2.0. *Rober Bousch GmbH*, 1991. (Cité en page 16), (Cité en page 22)

[Bur99] Alan Burns. The ravenscar profile. *Ada Letters*, XIX(4) :49–52, December 1999. (Cité en page 58), (Cité en page 105)

[But11] Giorgio Buttazzo. *Hard real-time computing systems : predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011. (Cité en page 17)

[C+13] AUTOSAR Consortium et al. The autosar standard, 2013. (Cité en page 113)

[CB97] M. Caccamo and G. Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In *Proceedings Real-Time Systems Symposium*, pages 330–339, Dec 1997. (Cité en page 17)

[CG07] Liliana Cucu and Joël Goossens. Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '07, pages 1635–1640, San Jose, CA, USA, 2007. EDA Consortium. (Cité en page 19)

[CGGM14] Liliana Cucu-Grosjean, Adriana Gogonel, and Dorin Maxim. *Ordonnancement dans les systèmes*, chapter Focus sur l'ordonnancement probabiliste, pages 155–188. ISTE Wiley, iste edition, 2014. (Cité en page 67)

[CHD14] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. SimSo : A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms. In *5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, page 6 p., Madrid, Spain, July 2014. (Cité en page 45), (Cité en page 76)

[CN02] Paul Clements and Linda Northrop. *Software product lines : practices and patterns*, volume 3. Addison-Wesley Reading, 2002. (Cité en page 109), (Cité en page 111)

[Cru91a] Rene L. Cruz. A calculus for network delay. i. network elements in isolation. *IEEE Transactions on Information Theory*, 37(1) :114–131, Jan 1991. (Cité en page 26)

[Cru91b] Rene L. Cruz. A calculus for network delay. ii. network analysis. *IEEE Transactions on Information Theory*, 37(1) :132–141, Jan 1991. (Cité en page 26)

[DAE12] Jonas Diemer, Philip Axer, and Rolf Ernst. Compositional performance analysis in python with pycpa. *Proc. of WATERS*, 2012. (Cité en page 46), (Cité en page 57)

[DB11a] Robert I Davis and Alan Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1) :1–40, 2011. (Cité en page 19)

[DB11b] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4) :35 :1–35 :44, October 2011. (Cité en page 3), (Cité en page 19)

[DBBL07] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (can) schedulability analysis : Refuted, revisited and revised. *Real-Time Systems*, 35(3) :239–272, 2007. (Cité en page 17), (Cité en page 23), (Cité en page 30)

[Der74] Michael L. Dertouzos. Control robotics : The procedural control of physical processes. In *IFIP Congress*, 1974. (Cité en page 14)

[DK04] Martin Dodge and Rob Kitchin. Flying through code/space : The real virtuality of air travel. *Environment and Planning A : Economy and Space*, 36(2) :195–211, 2004. (Cité en page 3)

[DL78] Sudarshan K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations research*, 26(1) :127–140, February 1978. (Cité en page 18)

[DM89] M. L. Dertouzos and A. K. Mok. Multiprocessor online scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15(12) :1497–1506, Dec 1989. (Cité en page 16), (Cité en page 19)

[dNLR09] D. d. Niz, K. Lakshmanan, and R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *2009 30th IEEE Real-Time Systems Symposium*, pages 291–300, Dec 2009. (Cité en page 18), (Cité en page 31)

[DoD78] Norme DoD. Aircraft internal time division – command resoponse multiplex data bus. Technical Report MIL–STD–1553A, US Departement of Defense, Geneva, CH, 1978. (Cité en page 25)

[DRSK89] A. Damm, J. Reisinger, W. Schwabl, and H. Kopetz. The real-time operating system of mars. *SIGOPS Oper. Syst. Rev.*, 23(3) :141–157, July 1989. (Cité en page 16), (Cité en page 31)

[EHA00] Johan Eker, Per Hagander, and Karl-Erik Arzén. A feedback scheduler for real-time controller tasks. *Control Engineering Practice*, 8(12) :1369 – 1378, 2000. (Cité en page 17)

[EHS97] A. Ermedahl, H. Hansson, and M. Sjodin. Response-time guarantees in atm networks. In *Proceedings Real-Time Systems Symposium*, pages 274–284, Dec 1997. (Cité en page 28)

[EN04] D Exertier and V Normand. Mdsyse : A model-driven systems engineering approach at thales. In *INCOSE Mid-Atlantic Regional Conference*, 2004. (Cité en page 38)

[Eur00a] Norme Euro. Applications ferroviaires - spécification et démonstration de la fiabilité, de la disponibilité, de la maintenabilité et de la sécurité (fdms) - partie 1 : exigences de base et procédés génériques. Technical Report NF EN 50126, AFNOR, Paris, FR, 2000. (Cité en page 36)

[Eur00b] Norme Euro. Applications ferroviaires - systèmes de signalisation, de télécommunication et de traitement - logiciels pour systèmes de commande et de protection ferroviaire. Technical Report NF EN 50128, AFNOR, Paris, FR, 2000. (Cité en page 36)

[Eur00c] Norme Euro. Applications ferroviaires - systèmes de signalisation, de télécommunications et de traitement - systèmes électroniques de sécurité pour la signalisation. Technical Report NF EN 50129, AFNOR, Paris, FR, 2000. (Cité en page 36)

[EUR11] EUROCAE. Ed-12c, software considerations in airborne systems and equipment certification, 2011. (Cité en page 36), (Cité en page 37)

[Fei04] Peter Feiler. Open source aadl tool environment (OSATE). In *AADL Workshop, paris*, pages 1–40, 2004. (Cité en page 36), (Cité en page 42)

[FG12] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL : An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 1st edition, 2012. (Cité en page xi), (Cité en page 42)

[FGB01] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, pages 183–192, Dec 2001. (Cité en page 19)

[FGC+06] Partick Farail, Pierre Gaufillet, Agusti Canals, Christophe Le Camus, David Sciamma, Pierre Michel, Xavier Crégut, and Marc Pantel. The topcased project : a toolkit in open source for critical aeronautic systems design. In *Proceedings of Embedded Real Time Software and Sytems (ERTS 2006)*, 2006. (Cité en page 57)

[FPE+89] Henry Fuchs, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebbs, and Laura Israel. Pixel-planes 5 : A heterogeneous multiprocessor graphics system using processor-enhanced memories. *SIGGRAPH Comput. Graph.*, 23(3) :79–88, July 1989. (Cité en page 18)

[Fuc12] Christian M. Fuchs. The evolution of avionics networks from arinc 429 to afdx. *Innovative Internet Technologies and Mobile Communications (IITM), and Aerospace Networks (AN)*, 65 :1551–3203, 2012. (Cité en page 22)

[FW99] Christian Ferdinand and Reinhard Wilhelm. Efficient and precise cache behavior prediction for real-time systems. *Real-Time Systems*, 17(2) :131–181, Nov 1999. (Cité en page 18), (Cité en page 31)

[Gar99] Mark K. Gardner. Probabilistic analysis and scheduling of critical soft real-time systems. Technical report, Champaign, IL, USA, 1999. (Cité en page 18)

[Gau14] Vincent Gaudel. *Des patrons de conception pour assurer l'analyse d'architectures : un exemple avec l'analyse d'ordonnancement*. PhD thesis, Université de Bretagne Occidentale, Novembre 2014. (Cité en page 48)

[GGH97] J. C. Palencia Gutierrez, J. J. Gutierrez Garcia, and M. Gonzalez Harbour. On the schedulability analysis for distributed hard real-time systems. In *Proceedings Ninth Euromicro Workshop on Real Time Systems*, pages 136–143, Jun 1997. (Cité en page 30)

[GGKL14] Antonio Garmendia, Esther Guerra, Dimitrios S. Kolovos, and Juan De Lara. *EMF Splitter : A Structured Approach to EMF Modularity*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. (Cité en page 57)

[GH95] J. J. G. Garcia and M. G. Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems*, WPDRTS '95, pages 124–, Washington, DC, USA, 1995. IEEE Computer Society. (Cité en page 17)

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979. (Cité en page 19)

[GL99] Mark K. Gardner and Jane W. S. Liu. Analyzing stochastic fixed-priority real-time systems. In W. Rance Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 44–58, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. (Cité en page 18)

[GLMR05] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier (H.). Roux. Romeo : A tool for analyzing time petri nets. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification*, pages 418–423, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. (Cité en page 45), (Cité en page 67)

[GPGdL15] Antonio Garmendia, Ana Pescador, Esther Guerra, and Juan de Lara. Towards the generation of graphical modelling environments aided by patterns. Languages, Applications and Technologies, pages 160–168. Springer International Publishing, 2015. (Cité en page 58)

[Gri04] Jérôme Grieu. *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques.* PhD thesis, Institut National Polytechnique de Toulouse, 2004. (Cité en page 26), (Cité en page 30)

[GRR11] Emmanuel Grolleau, Michael Richard, and Pascal Richard. Scheduling in distributed real-time systems. In Serge Haddad, Fabrice Kordon, Laurent Pautet, and Laure Petrucci, editors, *Distributed Systems : Design and Algotrithms*, chapter 7, pages 117–158. ISTE, Wiley, 2011. (Cité en page 28), (Cité en page 29), (Cité en page 30)

[GTPH01] N. Gandhi, D. M. Tilbury, S. Parekh, and J. Hellerstein. Feedback control of a lotus notes server : modeling and control design. In *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, volume 4, pages 3000–3005 vol.4, 2001. (Cité en page 17)

[Hac18] Kahina Hacid. *Handling domain knowledge in system design models. An ontology based approach.* PhD thesis, 2018. Thèse de doctorat dirigée par Aït-Ameur, Yamine Sureté de Logiciel et Calcul à Haute Performance Toulouse, INPT 2018. (Cité en page 58)

[Har84] Paul K. Harter, Jr. Response times in level-structured systems. Technical report, Department of Computer Science, University of Colorado, 1984. (Cité en page 15)

[Har87] Paul K. Harter, Jr. Response times in level-structured systems. *ACM Trans. Comput. Syst.*, 5(3) :232–248, August 1987. (Cité en page 15)

[HGGM01] M. Gonzalez Harbour, J. J. Gutierrez Garcia, J. C. Palencia Gutierrez, and J. M. Drake Moyano. Mast : Modeling and analysis suite for real time applications. In *Proceedings 13th Euromicro Conference on Real-Time Systems*, pages 125–134, 2001. (Cité en page 46)

[HHHKaL91] Michael González Harbour, Mark Harbour, and John P H. Klein and. Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. In *In Proceedings, IEEE Real-Time Systems Symposium*, pages 116–128. IEEE Computer Society Press, 1991. (Cité en page 16), (Cité en page 31)

[HHJ+05] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis – the symta/s approach. *IEE Proceedings - Computers and Digital Techniques*, 152 :148–166(18), March 2005. (Cité en page 57), (Cité en page 66)

[Hor74] WA Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1) :177–185, 1974. (Cité en page 14)

[HRS13] Rafik Henia, Laurent Rioux, and Nicolas Sordon. Tempo : Performance viewpoint for component-based design of real-time systems. *SIGBED Rev.*, 10, July 2013. (Cité en page 48), (Cité en page 58), (Cité en page 76)

[HS97] Chao-Ju Hou and K. G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Transactions on Computers*, 46(12) :1338–1356, Dec 1997. (Cité en page 19)

[HSD+04] Bill Haskins, Jonette Stecklein, Brandon Dick, Gregory Moroney, Randy Lovell, and James Dabney. Error cost escalation through the project life cycle. *INCOSE International Symposium*, 14(1) :1723–1737, 2004. (Cité en page xiii), (Cité en page 38)

[HSF+04] Harald Heinecke, Klaus-Peter Schnelle, Helmut Fennel, Jürgen Bortolazzi, Lennart Lundh, Jean Leflour, Jean-Luc Maté, Kenji Nishikawa, and Thomas Scharnhorst. Automotive open system architecture-an industry-wide initiative to manage the complexity of emerging automotive e/e-architectures. Technical report, SAE Technical Paper, 2004. (Cité en page 36)

[HZPK08] Jerome Hugues, Bechir Zalila, Laurent Pautet, and Fabrice Kordon. From the prototype to the final embedded system using the ocarina aadl tool suite. *ACM Trans. Embed. Comput. Syst.*, 7(4) :42 :1–42 :25, August 2008. (Cité en page 48)

[IEE03] IEEE. 1003.13-2003 - standard for information technology - standardized application environment profile - posix realtime and embedded application support, 2003. (Cité en page 113)

[ISO94] Norme ISO. Véhicules routiers – communication en série de données à basse vitesse. Technical Report ISO/IEC 11519 :1994, ISO, Geneva, CH, 1994. (Cité en page 22), (Cité en page 30)

[ISO11] Norme ISO. Véhicules routiers – sécurité fonctionnelle. Technical Report ISO/IEC 26262 :2011, ISO, Geneva, CH, 2011. (Cité en page 35)

[ISO12] Norme ISO. Information technology — programming languages — ada. Technical Report ISO/IEC 8652 :2012, ISO, Geneva, CH, 2012. (Cité en page 58)

[ISO13] Norme ISO. Véhicules routiers – système de communications flexray. Technical Report ISO/IEC 17458 :2013, ISO, Geneva, CH, 2013. (Cité en page 25), (Cité en page 30)

[ISO15] Norme ISO. Ingénierie des systèmes et du logiciel – processus du cycle de vie du système. Technical Report ISO/IEC 15288 :2015, ISO, Geneva, CH, 2015. (Cité en page 37)

[ISO16a] Norme ISO. Véhicules routiers – gestionnaire de réseau de communication (can). Technical Report ISO/IEC 11898 :2016, ISO, Geneva, CH, 2016. (Cité en page 22), (Cité en page 30)

[ISO16b] Norme ISO. Véhicules routiers – réseau internet local (lin). Technical Report ISO/IEC 17987 :2016, ISO, Geneva, CH, 2016. (Cité en page 24), (Cité en page 30)

[ITU99] Norme ITU. General description of asynchronous transfer mode. Technical Report I 150, International Telecommunication Union, Geneva, CH, 1999. (Cité en page 27), (Cité en page 30)

[JABK08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl : A model transformation tool. *Science of Computer Programming*, 72(1) :31 – 39, 2008. Special Issue on Second issue of experimental software and toolkits (EST). (Cité en page 40)

[Jac57] James R. Jackson. Simulation research on job shop production. *Naval Research Logistics Quarterly*, 4(4) :287–295, 1957. (Cité en page 14)

[JBF11] Jean-Marc Jézéquel, Olivier Barais, and Franck Fleurey. *Model Driven Language Engineering with Kermeta*, pages 201–221. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. (Cité en page 40)

[JP86] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5) :390–395, 1986. (Cité en page 15)

[Kai82] Claude Kaiser. *Exclusion mutuelle et ordonnancement par priorité*. Technique et Science Informatiques, 1982. (Cité en page 16)

[KBR+15] G. Kemayo, N. Benammar, F. Ridouard, H. Bauer, and P. Richard. Improving afdx end-to-end delays analysis. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sept 2015. (Cité en page 27)

[KDN11] D. A. Khan, R. I. Davis, and N. Navet. Schedulability analysis of can with non-abortable transmission requests. In *ETFA2011*, pages 1–8, Sept 2011. (Cité en page 24)

[KMG11] Pierre Kelsen, Qin Ma, and Christian Glodt. *Models within Models : Taming Model Complexity Using the Sub-model Lattice*, pages 171–185. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. (Cité en page 57)

[KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0 : Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, pages 585–591, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cité en page 67)

[KPP06] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. The epsilon object language (eol). In Arend Rensink and Jos Warmer, editors, *Model Driven Architecture – Foundations and Applications*, pages 128–142, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. (Cité en page 40)

[KRBR14] Georges Kemayo, Frédéric Ridouard, Henri Bauer, and Pascal Richard. A forward end-to-end delays analysis for packet switched networks. In *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 65 :65–65 :74, New York, NY, USA, 2014. ACM. (Cité en page 27), (Cité en page 30), (Cité en page 46)

[KS86] E. Kligerman and A. D. Stoyenko. Real-time euclid : A language for reliable real-time systems. *IEEE Transactions on Software Engineering*, SE-12(9) :941–949, Sept 1986. (Cité en page 16)

[KS95] G. Koren and D. Shasha. Skip-over : algorithms and complexity for overloaded systems that allow skips. In *Proceedings 16th IEEE Real-Time Systems Symposium*, pages 110–117, Dec 1995. (Cité en page 17)

[KTS+09] Christian Kastner, Thomas Thum, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. Featureide : A tool framework for feature-oriented software development. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 611–614, Washington, DC, USA, 2009. IEEE Computer Society. (Cité en page 112)

[Lam77] S. Lam. Delay analysis of a time division multiple access (tdma) channel. *IEEE Transactions on Communications*, 25(12) :1489–1494, December 1977. (Cité en page 30)

[Leh90] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *[1990] Proceedings 11th Real-Time Systems Symposium*, pages 201–209, Dec 1990. (Cité en page 16), (Cité en page 31)

[Leh96] J. P. Lehoczky. Real-time queueing theory. In *17th IEEE Real-Time Systems Symposium*, pages 186–195, Dec 1996. (Cité en page 18), (Cité en page 31)

[Leh98] J. P. Lehoczky. Scheduling communication networks carrying real-time traffic. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 470–479, Dec 1998. (Cité en page 17)

[Lip68] John S. Liptay. Structural aspects of the system/360 model 85 : Ii the cache. *IBM Systems Journal*, 7(1) :15–21, March 1968. (Cité en page 31)

[LKR10] Kevin Lano and Shekoufeh Kolahdouz-Rahimi. Slicing of uml models using model transformations. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, *Model Driven Engineering Languages and Systems*, pages 228–242, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. (Cité en page 57)

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1) :46–61, January 1973. (Cité en page 14), (Cité en page 15), (Cité en page 20), (Cité en page 31), (Cité en page 44), (Cité en page 45)

[LM80] Joseph Y.-T. Leung and M.L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3) :115 – 118, 1980. (Cité en page 15), (Cité en page 31)

[LS86] John P. Lehoczky and Lui Sha. Performance of real-time bus scheduling algorithms. *SIGMETRICS Perform. Eval. Rev.*, 14(1) :44–53, May 1986. (Cité en page 16)

[LSD89] John P. Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm : exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171, Dec 1989. (Cité en page 17)

[LSS87] John P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environment. *Proceedings of IEEE Real-Time Systems Symposium'87*, pages 261–270, 1987. (Cité en page 16)

[LSTS99] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Design and evaluation of a feedback control edf scheduling algorithm. In *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No.99CB37054)*, pages 56–67, 1999. (Cité en page 17)

[LTE+09] Agnes Lanusse, Yann Tanguy, Huascar Espinoza, Chokri Mraidha, Sebastien Gerard, Patrick Tessier, Remi Schnekenburger, Hubert Dubois, and François Terrier. Papyrus uml : an open source toolset for mda. In *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*, pages 1–4, 2009. (Cité en page 57)

[LW82] Joseph Y-T Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4) :237–250, 1982. (Cité en page 15), (Cité en page 31)

[MAC+87] Aloysius K Mok, Prasanna Amerasinghe, Moyer Chen, Supoj Sutanthavibul, and Kamtorn Tantisirivat. Synthesis of a real-time message processing system with data-driven timing constraints. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pages 133–143, 1987. (Cité en page 16)

[MACT89] Aloysius Mok, P. Amerasinghe, M. Chen, and K. Tantisirivat. Evaluating tight execution time bounds of programs by annotations. *IEEE Real-Time Syst. Newsl.*, 5(2-3) :81–86, May 1989. (Cité en page 16), (Cité en page 31)

[Mar91] E.P. Markatos. Multiprocessor synchronization primitives with priorities. *IFAC Proceedings Volumes*, 24(2) :1 – 6, 1991. IFAC/IFIP Workshop on Real Time Programming, Atlanta, GA, USA, 15-17 May 1991. (Cité en page 16), (Cité en page 19)

[MC97] Aloysius K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10) :635–645, Oct 1997. (Cité en page 31), (Cité en page 44), (Cité en page 45)

[MGC11] Julio L. Medina and Alvaro Garcia Cuesta. Model-based analysis and design of real-time distributed systems with ada and the uml profile for marte. In Alexander Romanovsky and Tullio Vardanega, editors, *Reliable Software Technologies - Ada-Europe 2011*, pages 89–102, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cité en page 55)

[MH92] Brendan P Mahony and Ian J Hayes. A case-study in timed refinement : A mine pump. *IEEE transactions on Software Engineering*, 18(9) :817–826, 1992. (Cité en page 77)

[MJL+12] Jonathan Musset, Étienne Juliot, Stéphane Lacrampe, William Piers, Cédric Brun, Laurent Goubet, Yvan Lussaud, and Freddy Allilaire. Acceleo user guide. 2, 2012. (Cité en page 41), (Cité en page 116)

[MMTS14] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Mps-can analyzer : Integrated implementation of response-time analyses for controller area network. *Journal of Systems Architecture*, 60(10) :828 – 841, 2014. (Cité en page 46)

[MNLM10] N. Tchidjo Moyo, E. Nicollet, F. Lafaye, and C. Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *2010 22nd Euromicro Conference on Real-Time Systems*, pages 271–278, July 2010. (Cité en page 44), (Cité en page 45)

[Mok83] Aloysius K. Mok. *Fundamental design problems of distributed systems for the hard-real-time environment*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983. (Cité en page 15), (Cité en page 44), (Cité en page 45)

[MP15] Frédéric Madiot and Marc Paganelli. Eclipse sirius demonstration. In *P&D@ MoDELS*, pages 9–11, 2015. (Cité en page 49)

[Mue99] F. Mueller. Priority inheritance and ceilings for distributed mutual exclusion. In *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No.99CB37054)*, pages 340–349, 1999. (Cité en page 16)

[MV07] Eric Maes and Nicolas Vienne. Marte to cheddar transformation using atl. Technical Document 61565546-179, Thales Report/Technical Documents number 61565546-179 and 61565546 108, Octobre 2007. 2007. (Cité en page 55)

[OB98] Dong-Ik Oh and T.P. Baker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(2) :183–192, Sep 1998. (Cité en page 19)

[OGR+15] Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, Pascal Richard, and Frédéric Madiot. *MoSaRT Framework : A Collaborative Tool for Modeling and Analyzing Embedded Real-Time Systems*, pages 283–295. Springer International Publishing, Cham, 2015. (Cité en page xi), (Cité en page 54)

[OMGa] OMG. Uml profile for telecommunication services specification. `https://www.omg.org/spec/TelcoML/`. (Cité en page 43)

[OMGb] OMG. Uml profile for voice-based applications specification. `https://www.omg.org/spec/VOICP/`. (Cité en page 43)

[OMG00] OMG. Model driven architecture. `https://www.omg.org/cgi-bin/doc?omg/00-11-05.pdf`, November 2000. (Cité en page 38)

[OMG06a] OMG. Object constraint language, omg available specification, version 2.0, 2006. (Cité en page 40)

[OMG06b] OMG. Object constraint language, omg available specification, version 2.0, 2006. (Cité en page 93)

[OMG11] OMG. Uml profile for marte specification. `https://www.omg.org/spec/MARTE/`, 2011. (Cité en page 43)

[OMG15] OMG. UML version 2.5. *OMG Specification*, 2015. (Cité en page 38), (Cité en page 39)

[OMG17a] OMG. Omg systems modeling language. `https://www.omg.org/spec/SysML/`, 2017. (Cité en page 36)

[OMG17b] OMG. Query/view/transformation QVT. `http://www.omg.org/spec/QVT/`, 2017. Accessed : 2017-07-17. (Cité en page 40)

[Ouh13] Yassine Ouhammou. *Model-based Framework for Using Advanced Scheduling Theory in Real-Time Systems Design*. PhD thesis, ISAE - ENSMA, dec 2013. (Cité en page xi), (Cité en page 3), (Cité en page 4), (Cité en page 5), (Cité en page 48), (Cité en page 49), (Cité en page 50), (Cité en page 51), (Cité en page 52), (Cité en page 69)

[PGA+11] H. Perez, J. J. Gutierrez, E. Asensio, J. Zamorano, and J. A. d. l. Puente. Model-driven development of high-integrity distributed real-time systems using the end-to-end flow model. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 209–216, Aug 2011. (Cité en page 55)

[PH98] J. C. Palencia and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 26–37, Dec 1998. (Cité en page 44), (Cité en page 45)

[Pha16] Guillaume Phavorin. *Hard Real-Time Scheduling subjected toCache-Related Preemption Delays*. PhD thesis, Université de Poitiers, September 2016. (Cité en page 18)

[PK89] P. Puschner and Ch. Koza. Calculating the maximum execution time of real-time programs. *Real-Time Systems*, 1(2) :159–176, Sep 1989. (Cité en page 16)

[PPE+08] Traian Pop, Paul Pop, Petru Eles, Zebo Peng, and Alexandru Andrei. Timing analysis of the flexray communication protocol. *Real-Time Systems*, 39(1) :205–235, Aug 2008. (Cité en page 25)

[PRG+18] Guillaume Phavorin, Pascal Richard, Joël Goossens, Claire Maiza, Laurent George, and Thomas Chapeaux. Online and offline scheduling with cache-related preemption delays. *Real-Time Systems*, 54(3) :662–699, Jul 2018. (Cité en page 18)

[Pri92] P. J. Prisaznuk. Integrated modular avionics. In *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference NAECON 1992*, pages 39–45 vol.1, May 1992. (Cité en page 56)

[Raj90] Ragunathan Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Distributed Computing Systems, 1990. Proceedings., 10th International Conference on*, pages 116–123. IEEE, 1990. (Cité en page 16)

[Ray87] Asok Ray. Performance evaluation of medium access control protocols for distributed digital avionics. *Journal of dynamic systems, measurement, and control*, 109(4) :370–377, 1987. (Cité en page 30)

[RBJ17] James Rumbaugh, Grady Booch, and Ivar Jacobson. *The unified modeling language reference manual*. Addison Wesley, 2017. (Cité en page 43)

[RGRR12] Ahmed Rahni, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Feasibility analysis of real-time transactions. *Real-Time Systems*, 48(3) :320–358, May 2012. (Cité en page 44), (Cité en page 45)

[Ric05] Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models : The SymTA-S Approach*. PhD thesis, 2005. (Cité en page 57), (Cité en page 106)

[RS84] Krithi Ramamritham and John A Stankovic. Dynamic task scheduling in hard real-time distributed systems. *IEEE software*, 1(3) :65, 1984. (Cité en page 16)

[RSL88] Ragunathan Rajkumar, Lui Sha, and John P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, pages 259–269, Dec 1988. (Cité en page 16), (Cité en page 19)

[RSS90] K. Ramamritham, J. A. Stankovic, and P. F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(2) :184–194, Apr 1990. (Cité en page 19)

[RSZ89] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8) :1110–1123, Aug 1989. (Cité en page 16)

[RTC12] RTCA. Do-178c, software considerations in airborne systems and equipment certification, 2012. (Cité en page 36), (Cité en page 37)

[RZJE02] Kai Richter, Dirk Ziegenbein, Marek Jersak, and Rolf Ernst. Model composition for scheduling analysis in platform design. In *Proceedings of the 39th Annual Design Automation Conference*, DAC '02, pages 287–292, New York, NY, USA, 2002. ACM. (Cité en page 106)

[SAÅ+04] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K Mok. Real time scheduling theory : A historical perspective. *Real-time systems*, 28(2-3) :101–155, 2004. (Cité en page 3)

[SBPM08a] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF : Eclipse Modeling Framework*. Pearson Education, 2008. (Cité en page 40)

[SBPM08b] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF : Eclipse Modeling Framework*. Pearson Education, 2008. (Cité en page 49)

[Sch06] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2) :25, 2006. (Cité en page 38)

[SEGY11] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 71–80, April 2011. (Cité en page 18), (Cité en page 31), (Cité en page 44), (Cité en page 45)

[SFR05] Arnor Solberg, Robert France, and Raghu Reddy. Navigating the metamuddle. In *Proceedings of the 4th Workshop in Software Model Engineering, Montego Bay, Jamaica*, 2005. (Cité en page 57)

[SH98] Mikael Sjodin and Hans Hansson. Improved response-time analysis calculations. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 399–408. IEEE, 1998. (Cité en page 17)

[Sha89] A. C. Shaw. Reasoning about time in higher-level language software. *IEEE Transactions on Software Engineering*, 15(7) :875–889, July 1989. (Cité en page 16)

[Sim14] Simso, 2014. (Cité en page 45)

[SLNM04] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar : A flexible real time scheduling framework. *Ada Lett.*, XXIV(4) :1–8, November 2004. (Cité en page 18), (Cité en page 45), (Cité en page 55)

[SLR86] Lui Sha, John P. Lehoczky, and Ragunathan Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. *Proc. 7th IEEE Real-Time Systems Symposium*, 1986. (Cité en page 16)

[SLR87] Lui Sha, J. P. Lehoczky, and Ragunathan Rajkumar. Task Scheduling In Distributed Real-Time Systems. In *Proceedings of the IEEE Industrial Electronics Conference*, pages 909–917, October 1987. (Cité en page 16)

[SLS88] Brinkley Sprunt, John Lehoczky, and Lui Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proceedings. Real-Time Systems Symposium*, pages 251–258, Dec 1988. (Cité en page 16)

[SLS95] J. K. Strosnider, J. P. Lehoczky, and Lui Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1) :73–91, Jan 1995. (Cité en page 16)

[SLST99] J. A. Stankovic, Chenyang Lu, S. H. Son, and Gang Tao. The case for feedback control real-time scheduling. In *Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on*, pages 11–20, 1999. (Cité en page 17)

[SM09] Philip Samuel and Rajib Mall. Slicing-based test case generation from uml activity diagrams. *SIGSOFT Softw. Eng. Notes*, 34(6) :1–14, December 2009. (Cité en page 57)

[SMBJ09] Sagar Sen, Naouel Moha, Benoit Baudry, and Jean-Marc Jézéquel. Meta-model pruning. In Andy Schürr and Bran Selic, editors, *Model Driven Engineering Languages and Systems*, pages 32–46, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. (Cité en page 57)

[Som06] Ian Sommerville. *Software Engineering : (Update) (8th Edition)*. Addison Wesley, 8 edition, 2006. (Cité en page 35)

[SRC85] J. A. Stankovic, K. Ramamritham, and S. Cheng. Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. *IEEE Transactions on Computers*, C-34(12) :1130–1143, Dec 1985. (Cité en page 16)

[SRL90] Lui Sha, Ragunathan Rajkumar, and John P Lehoczky. Priority inheritance protocols : An approach to real-time synchronization. *Computers, IEEE Transactions on*, 39(9) :1175–1185, 1990. (Cité en page 16)

[SRTC14] Daniel Struber, Julia Rubin, Gabriele Taentzer, and Marsha Chechik. *Splitting Models Using Information Retrieval and Model Crawling Techniques*, pages 47–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. (Cité en page 57)

[SS09] K. Schmidt and E. G. Schmidt. Message scheduling for the flexray protocol : The static segment. *IEEE Transactions on Vehicular Technology*, 58(5) :2170–2179, Jun 2009. (Cité en page 30)

[SSL89] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1) :27–60, Jun 1989. (Cité en page 16)

[Str88] Jay Kurt Strosnider. *Highly Responsive Real Time Token Rings*. PhD thesis, Pittsburgh, PA, USA, 1988. AAI8905263. (Cité en page 16)

[SY15] Martin Stigge and Wang Yi. Graph-based models for real-time workload : a survey. *Real-Time Systems*, 51(5) :602–636, 2015. (Cité en page 44), (Cité en page 45)

[TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks : An np-hard problem made easy. *Real-Time Systems*, 4(2) :145–165, Jun 1992. (Cité en page 19)

[TBW95] Ken Tindell, A. Burns, and A. Wellings. Calculating controller area network (can) message response times. In J.A. DE LA PUENTE and M.G. RODD, editors, *Distributed Computer Control Systems 1994*, IFAC Postprint Volume, pages 29 – 34. Pergamon, Oxford, 1995. (Cité en page 17), (Cité en page 23)

[TC94] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2) :117 – 134, 1994. Parallel Processing in Embedded Real-time Systems. (Cité en page 17), (Cité en page 24), (Cité en page 30), (Cité en page 100)

[THW94] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing real-time communications : controller area network (can). In *1994 Proceedings Real-Time Systems Symposium*, pages 259–263, Dec 1994. (Cité en page 17)

[Tin94] Ken Tindell. *Adding time-offsets to schedulability analysis*. Citeseer, 1994. (Cité en page 16), (Cité en page 24), (Cité en page 30), (Cité en page 31)

[US 94] US Departement Of Defense. MIL-STD-498,application and reference guidebook, 1994. (Cité en page 37)

[VK00] Steve Vestal and Jonathan Krueger. Technical and historical overview of metah. *Honeywell Technology Center*, 2000. (Cité en page 42)

[Voi18] Jean-Luc Voirin. *Model-based System and Architecture Engineering with the Arcadia Method*. ISTE Press, 2018. (Cité en page 36)

[WDD+12] V. Wiels, R. Delmas, D Doose, P.L. Garoche, J. Cazin, and G. Durrieu. Formal Verification of Critical Aerospace Software. *AerospaceLab*, (4) :p. 1–8, May 2012. (Cité en page 3)

[Wil92] Maurice V Wilkes. Edsac 2. *IEEE Annals of the History of Computing*, (4) :49–56, 1992. (Cité en page 31)

[XP90] Jia Xu and David Lorge Parnas. Scheduling processes with release times, deadlines, precedence and exclusion relations. *Software Engineering, IEEE Transactions on*, 16(3) :360–369, 1990. (Cité en page 17)

[ZA04] Omar U. Pereira Zapata and Pedro Mejia Alvarez. Edf and rm multiprocessor scheduling algorithms : Survey and performance evaluation. Technical report, Sección de Computación, 2004. (Cité en page 3)

[ZR85] Wei Zhao and Krithi Ramamritham. Distributed scheduling using bidding and focused addressing. In *Proceedings of RTSS 1985*, 1985. (Cité en page 16)

[ZR87] Wei Zhao and K. Ramamritham. Virtual time csma protocols for hard real-time communication. *IEEE Transactions on Software Engineering*, SE-13(8) :938–952, Aug 1987. (Cité en page 16)

Annexe A

Liste des acronymes

AADL *Architecture Analysis & Design Language*. 42, 48, 53, 55, 59, 86

ABS *Anti-lock Breaking System*. 12

AFDX *Avionics Full DupleX*. 25, 27, 31, 84, 92

ATL *Atlas Transformation Language*. 40, 58, 71, 116

ATM *Asynchronous Transfert Mode*. 27

BAG *Bandwidth Allocation Gap*. 25, 26

CAN *Controler Area Network*. 16, 17, 22, 87, 92

CONSERT *CONServative Endogenous Repository based Transformations*. 69–71, 75–78, 80, 127, 128

CPIOM *Core Processing & I/O Module*. 25

CRPD *Cache Related Preemption Delay*. 18, 31

CSMA-CA *Carrier Sense Multiple Access - Collision Avoidance*. 16, 22

CSMA-CD *Carrier Sense Multiple Access - Collision Detection*. 16

DBF *Demand Bound Function*. 15

DSML *Domain-Specific Modeling Language*. 41, 53, 57, 59

E/S *End System*. 25

EDF *Earliest Deadline First*. 14, 15, 19, 95

EMF *Eclipse Modeling Framework*. 40

FIFO *First In, First Out*. 27, 84, 88

IMA *Integrated Modular Avionics*. 56

LIN *Local Interconnect Network*. 24, 25

LLF *Least Laxity First*. 15

M2M *Model-to-Model*. 40

M2T *Model-To-Text*. 40, 41

MAST *Modeling and Analysis Suite for Real-time applications*. 46–48, 65, 66, 76, 78, 79, 100, 129

MOF *Meta Object Facility*. 39, 40

MoSaRT *Modeling oriented Scheduling analysis of Real Time systems*. 48, 49, 52–55, 76, 87, 101, 107, 108, 115, 128

NoC *Network on Chip*. 18, 105

OCL *Object Constraint Language*. 40, 71, 93, 99, 111

OMG *Object Management Group*. 38–40

OPA *Optimal Priority Assignement*. 17, 19

RBF *Request Bound Function*. 15, 20, 68, 69

RM *Rate Monotonic*. 14, 17

RTA *Response Time Analysis*. 23

SPL *Software Product Line*. 111

TDMA *Time-Division Multiple Access*. 25

UML *Unified Modeling Language*. 40, 43, 57

UML - MARTE *Modeling and Analysis of Real-time and Embedded systems*. 43, 48, 49, 53, 56, 58, 59, 76, 84, 86, 87, 105, 110, 127

URI Uniform Resource Identifier. 73

VL *Virtual Link*. 25–27, 84, 92, 97

WCET *Worst Case Execution Time*. 13, 16, 21, 67

Annexe B

Liste des symboles

T_i Dans le cadre d'une tâche périodique, le temps entre deux activations. Pour une tâche sporadique, c'est le temps minimum entre deux activations.. 13, 78

C_i Temps d'exécution d'une tâche. 13, 78

D_i Le délai maximum donné à une tâche pour s'exécuter après son réveil. 13

r_i Dans le cadre d'une tâche périodique, c'est la première date d'activation de la tâche τ_i. 13

Vous êtes encore là ?

— Netflix

Résumé

La validation temporelle des systèmes temps réel est nécessaire dans le cadre d'applications critiques tels l'aéronautique, le spatial ou l'automobile. Il s'agit, dans ces systèmes, de garantir les temps de réponse des tâches ainsi que le déterminisme de leurs communications. En raison de la complexité des systèmes actuels ainsi que de leur criticité, il est nécessaire de mettre en place une démarche de conception réduisant le temps de développement et ainsi le temps de mise en marché (*time-to-market*), tout en réduisant les risques d'erreurs de conception.

Ce contexte rend l'ingénierie dirigée par les modèles particulièrement adaptée au développement de ce type de système. Les contributions de cette thèse partent des constats suivants. Premièrement, malgré la multitude des modèles d'analyses existants les modèles actuels ne peuvent retranscrire de nombreux cas rencontrés en industrie. Pour pouvoir analyser ces cas, il est nécessaire de les adapter à l'analyse. Le deuxième constat porte sur l'adaptation qui n'est toujours pas simple surtout quand il existe une disparité sémantique entre les langages de description d'architecture et les modèles d'analyse ce qui nécessite de les rapprocher. Le dernier constat porte sur les difficultés dans la modélisation des systèmes distribués complexes car à moins de connaître le système complet, la représentation globale et sa validation temporelle reste coûteuse.

Cette thèse propose des outils et méthodes pour améliorer le processus de modélisation et d'analyses temps-réel. La première contribution consiste en la mise en place d'un référentiel de transformation de modèles endogène pour effectuer une adaptation conservative des modèles industriels aux modèles d'analyses. Dans l'optique de réduire l'écart sémantique entre les langages de description et les modèles d'analyse, cette thèse propose également une modélisation incrémentale des réseaux temps réel en vue de leur validation temporelle car les langages existants les considèrent de façon limitée. La troisième contribution de la thèse porte sur la réduction des artéfacts de modélisation par extraction et élagage de méta-modèles afin d'obtenir les éléments nécessaires par rapport au points de vue d'analyse souhaités.

Toutes ces contributions sont implémentées dans des *frameworks* intégrant les processus d'analyses temps réel tels *Time4Sys* et *MoSaRT* et utilisées, dans le cadre d'un projet collaboratif, par des partenaires industriels.

Mots-clés : Ingénierie dirigée par les modèles ; Systèmes critiques ; Temps réel (informatique) ; Ordonnancement dynamique ; Analyse temporelle ; Systèmes à paramètres répartis ; Systèmes, Conception de ; Systèmes, Analyse de ; Modélisation des données ; Tolérance aux fautes (informatique) ; Langages de modélisation

---

Abstract

The timing validation of real-time systems is mandatory for critical applications such as aeronautics, aerospace or automotive systems. The aim is to guarantee tasks response time and messages transmission time on networks. As for the criticality of these complex systems, it is necessary to implement a design process that reduces the development time therefore the time-to-market while reducing design errors risks.

This context makes model-driven engineering well adapted for the development of critical real-time systems. The contributions of this thesis rely on the following observations. First of all, despite the existence of various analysis models, they often cannot represent perfectly some industrial cases. To analyze these cases, an adaptation is required to make them analyzable with existing tests. However, the adaptation is not quite easy especially in case of a semantic gap between systems description languages and analysis models. Also, several difficulties have been noticed to design and analyze an entire distributed complex system in one-shot unless knowing well the full system.

In this PhD thesis, tools and methods are proposed to ease and improve the modeling and analysis processes of real-time systems. The first thesis contribution consists of implementing a rule-based endogenous transformation repository dedicated to adapt conservatively industrial models to the analysis models. The second contribution is focused on real-time networks and is dedicated to reduce the semantic gap between description languages and analysis models by proposing artefacts allowing to design networks on an incremental way. Moreover, this thesis proposes to reduce modeling artifacts using extraction and meta-models pruning in order to retrieve useful elements referring to chosen analysis viewpoints.

All these contributions are implemented in modeling frameworks integrating real-time analyses processes such as Time4Sys and MoSaRT, and used in a collaborative project by industrial partners.

Keywords : Real-time data processing ; Time-domain analysis ; Distributed parameter systems ; System design ; System analysis ; Fault-tolerant computing

---

Secteur de recherche : Informatique et applications

Laboratoire d'Informatique et d'Automatique pour les Systèmes
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Téléport 2 – 1 avenue Clément Ader – BP 40109 – 86961 FUTUROSCOPE CHASSENEUIL CEDEX
Tél : 05.49.49.80.63 – Fax : 05.49.49.80.64