Aide à la validation temporelle et au dimensionnement de systèmes temps réels dans une démarche dirigée par modèles

Thanh Dat Nguyen

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 :

Thanh Dat NGUYEN

****

Aide à la validation temporelle et au dimensionnement de systèmes temps réel dans une démarche dirigée par les modèles

****

Directeur de thèse : **Emmanuel GROLLEAU**
Co-encadrant de thèse : **Yassine OUHAMMOU**

****

Soutenue le Juillet 2020
devant la Commission d'Examen

****

JURY

Président :

Annie GENIET	Professeur, Université de Poitiers, Poitiers

Rapporteurs :

Olivier H. ROUX	Professeur, École Centrale de Nantes, Nantes
Chokri MRHAIDA	Chercheur HDR, CEA LIST, Saclay-Palaiseau

Membres du jury :

Nicolas NAVET	Professeur, Université du Luxembourg, Luxembourg
Laurent RIOUX	Docteur-Ingénieur, Thalès R&T, Saclay-Palaiseau
Yassine OUHAMMOU	Maître de Conférences, ISAE-ENSMA, Chasseneuil-du-Poitou
Emmanuel GROLLEAU	Professeur, ISAE-ENSMA, Chasseneuil-du-Poitou

Remerciements

Je tiens tout d'abord à remercier mon directeur de thèse monsieur Emmanuel GROLLEAU, ainsi que mon encardrant monsieur Yassine OUHAMMOU pour m'avoir guidé pendant ces années de thèse, leur patience et leurs conseils toujours précieux, sans votre aide je ne peux pas terminer la thèse littéralement.

Je tiens également à remercier tous les membres de jury pour avoir accepté de participer dans ma soutenance et d'évaluer ce travail de thèse. Je tiens à remercier particulièrement monsieur Olivier H. ROUX et monsieur Chokri MRAIDHA d'avoir accepté lourde charge d'être rapporteurs de cette thèse. Je remercie également madame Annie Geniet, monsieur Nicolas NAVET et monsieur Laurent RIOUX d'avoir accepté d'être mes examinateurs de ma soutenance.

J'aimerais adresser un remerciement à madame Annie CHOQUET-GENIET d'avoir accepté d'être la présidente de ma soutenance.

Un grand merci à monsieur Laurent GUITTET et monsieur Ladjel BELLATRECHE ainsi que les autres permanents pour leur bonne humeur, et leur soutien technique particulièrement monsieur Mickael BARON, monsieur Frédéric CARREAU. Je tiens aussi à remercier madame Bénédicte Boinot pour son aide administratif.

Mes vifs remerciements vont également à tous les amis doctorants Ibrahim, Matheus, Soulimane, Jorge, Ishaq,... pour leurs encouragements. Je tiens aussi à remercier mes amis vietnamiens pour le temps partagé entre nous.

Mes derniers remerciements vont vers ma famille pour son soutien inconditionnel tout au long de ma vie, particulièrement ces années difficiles de thèse.

Table des matières

Liste des figures

Liste des tableaux

Chapitre 1

Introduction générale

Sommaire

1.1 Contexte et motivation

Les systèmes embarqués temps réel critiques sont très utilisés dans plusieurs domaines (e.g., avionique, nucléaire, automobile, etc.) où le cycle de développement prend plusieurs mois, voire plusieurs années. Cela peut générer des coûts élevés relatifs à la durée de développement qui pourrait être allongée si l'on s'aperçoit tardivement que la conception retenue ne permet pas de respecter les exigences. Dans le domaine des exigences liées à la performance temporelle, la phase de conception, en particulier la répartition des fonctions dans des tâches, et des tâches sur des calculateurs, a un fort impact sur les performances du système développé. Par conséquent, dès la phase de conception des systèmes embarqués temps réel, il faudrait pouvoir analyser le potentiel de la conception retenue vis-à-vis la satisfaction des exigences et en particulier, dans le cadre de cette thèse, les exigences temporelles (e.g., échéances, délais de bout en bout).

Actuellement, bien que plusieurs méthodes et outils de conception existent et permettent aux architects d'automatiser certains processus, il n'existe aucune méthodologie d'ingénierie outillée permettant de traiter l'aspect d'aide à l'analyse de manière homogène et différents niveaux de granularité tout au long du cycle de développement des systèmes temps réel critiques. Les outils permettant de modéliser le comportement temporel d'un système ne couvrent en général qu'une seule phase du développement, ils sont restreints à une technologie, ils ne s'intègrent pas dans l'environnement de l'ingénieur et ils ne permettent pas de combiner les résultats complémentaires des différentes analyses réalisées.

1.1.1 Le projet Waruna

Afin d'aider les architectes systèmes à concevoir correctement des systèmes complexes tout en satisfaisant non-seulement les exigences fonctionnelles mais aussi les exigences non-fonctionnelles (appelées aussi extra-fonctionnelles) issus des différents domaines et qui peuvent être complémentaires ou contradictoires nécessitant des compromis, le paradigme "ingénierie système à base de modèles (model-based system engineering en anglais)" est devenu de plus en plus utilisé dans les pratiques industrielles [BKT+06, PHAB12]. Il s'agit d'une méthodologie interdisciplinaire qui permet aux ingénieurs de différents domaines et cultures d'avoir un terrain d'entente en travaillant autour d'un modèle principal comme moyen d'échange d'informations plutôt que les échanges basés sur des documents. Ces derniers peuvent être sources de mal-interprétation et/ou d'ambiguïtés structurelles et sémantiques. Bien entendu, un ensemble de modèles peuvent être dérivés à partir d'un modèle central pivot selon le type et la complexité du système. La dérivation peut être dirigée par étape de conception dans le cycle de vie (modèle de besoins, modèle d'analyse, modèle de déploiement) et/ou et par domaine (modèle stabilité du contrôleur, modèle d'analyse temporelle, modèle de consommation énergétique).

Vu le caractère critique des systèmes temps réel abordés dans cette thèse, plusieurs méthodes et outils, y compris les cycles de développement, ont été proposés afin d'assurer une conception rigoureuse. Les cycles définissent un ensemble d'activités, leurs artefacts d'entrée/de sortie et leurs rôles. Bien qu'il existe plusieurs cycles comme le cycle en cascade (*Waterfall*) [Dav90] et le cycle en spirale [Boe88], le cycle en V [MR84] est l'un des cycles de développement les plus utilisés pour concevoir les systèmes temps réel critiques.

Les étapes du cycle en V sont décrites dans la figure 1.1. Le cycle en V met l'accent sur les activités de vérification et de validation. Le développement d'un système suit la branche descendante (le côté gauche) tandis que les activités de vérification et de validation suivent la branche montante (le côté droit). Ainsi, le système mis en oeuvre est vérifié par rapport à chaque artefact produit. Ce modèle permet en cas d'anomalie de limiter un retour aux étapes précédentes. Les phases de la pente montante doivent renvoyer de l'information sur les phases en vis-à-vis lorsque des défauts sont détectés afin d'améliorer le système. Le cycle en V permet de construire les plans de test dès la phase de développement.

Le cycle en V a quelques avantages : facile à comprendre et à utiliser et le périmètre des

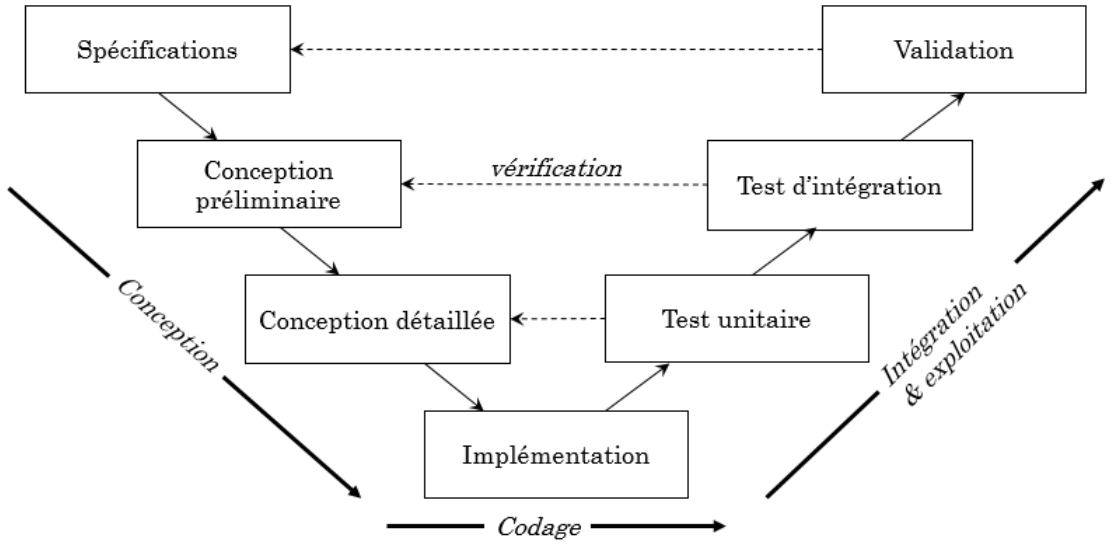

FIGURE 1.1 – Cycle en V

rôles et des étapes est défini avec clarté, cependant il reste très rigide. Depuis des décennies, le cycle en V et ses variantes est au cœur des cycles de développement de systèmes critiques. En effet, bien qu'à l'opposé des autres méthodes existantes ce cycle ne soit pas du tout propice au changement d'exigences ou de cahier des charges, il permet d'aboutir à un logiciel sûr.

Dans cette perspective le projet FUI Waruna [1] vise à construire un atelier de modélisation et de vérification de propriétés temporelles, qui couvre toutes les étapes de développement (voir la figure 1.2) et qui permet d'évaluer au plus tôt les impacts des choix d'architecture en matière de temps de réponse, qui fusionne les résultats des outils d'analyse obtenus à différents niveaux du développement et qui s'intègre dans un environnement de modélisation moderne de type ingénierie système à base de modèles.

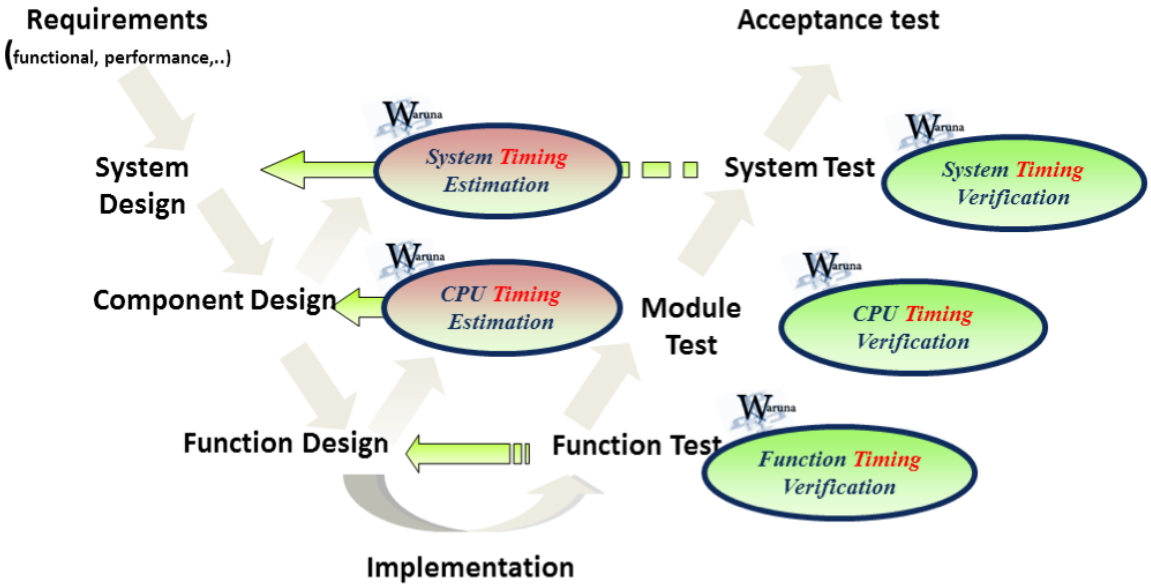

FIGURE 1.2 – Cycle de conception prévue pour Waruna

Le projet Waruna s'intéresse donc à l'interopérabilité et à l'intégration des outils d'analyses temporelles de façon à réaliser des analyses de manière incrémentale, pour toutes les étapes de

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

conception d'un système, au sein d'un environnement métier. Il permettra de tenir compte des informations disponibles au fur et à mesure que les détails d'implémentation sont connus et de comparer des choix d'architectures différentes alors même que tous les éléments d'architecture ne sont pas encore connus. La conception sera alors mieux maîtrisée.

1.1.2 Positionnement de la thèse

L'analyse temporelle incrémentale telle qu'elle est souhaitée dans le cadre du projet WARUNA nécéssite de mettre en place une modélisation dirigée par l'analyse temporelle comme le montre la figure 1.3. En effet, la figure est un focus sur le processus de modélisation des systèmes temps réel critiques et ses différentes étapes. Le passage d'une étape à l'autre est le résultat d'une activité d'analyse qui varie selon l'étape et le type de modèles. Une activité d'analyse peut être de type exploration, de dimensionnement ou de validation. De plus, quelque soit le type et la méthode d'analyse utilisée, l'ordonnançabilité est une caractéristique primordiale qui doit être toujours vérifiée.

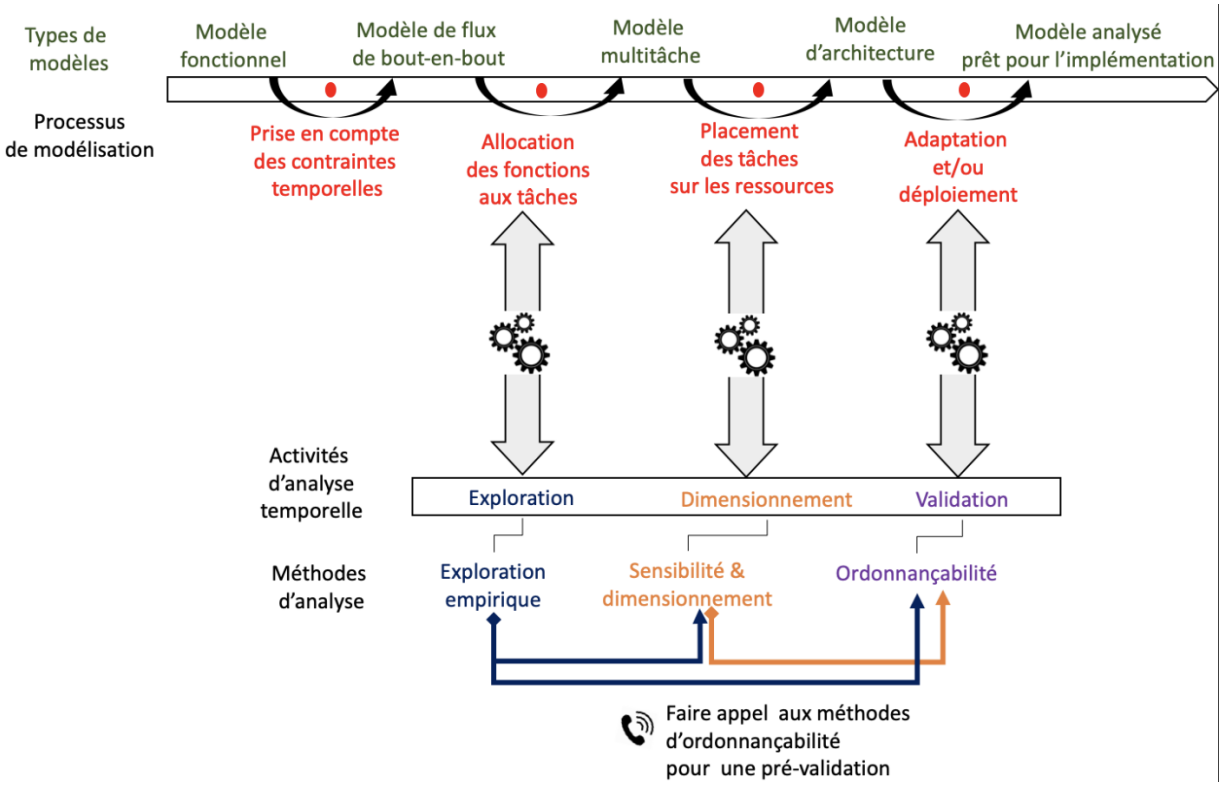

FIGURE 1.3 – Modélisation dirigée par l'analyse temporelle

Tout un pan de la littérature a proposé depuis cinquante ans de nombreux modèles et méthodes d'analyse basés sur la théorie de l'ordonnancement. Cependant, l'une de difficultés principales rencontrées dans la vérification des exigences temporelles est que celle-ci nécessite à l'heure actuelle l'expertise des analystes, alors que la phase de conception est généralement menée par l'architecte système dont l'analyse temporelle n'est pas le cœur de son métier. Bien que cette thèse s'est déroulée dans un contexte lié à l'ingénierie système à base de modèles, les contributions ne portent pas seulement sur le rapprochement entre les activités d'analyse et le processus de modélisation mais aussi sur la proposition des nouveaux tests d'analyse d'ordonnançabilité.

1.2 Contributions de la thèse

Cette thèse propose trois contributions afin de répondre à trois objectifs principaux.

Le premier objectif de la thèse s'inscrit dans le cadre du problème de l'ordonnancement de tâches temps réel périodiques à contraintes de précédence multi-périodiques. Ainsi par exemple, une tâche de période 100 millisecondes envoie à chaque période un messages à une tâche de période 75 millisecondes, comment cela fonctionne-t-il, et surtout comment l'analyser ? Nous avons été amenés, afin d'intégrer cette possibilité dans un langage de modélisation pivot, à définir la sémantique des contraintes de précédence multi-périodiques en étendant et outillant les contraintes de précédence de type producteur/consommateur (*Semaphore Precedence Constraint* - SPC). Nous les avons étendues en lui permettant de supporter des cycles. Un graphe de dépliage a été également proposé afin de présenter la cyclicité des systèmes utilisant SPC et garantir le non-blocage. Etant donné que ces travaux se sont déroulés en phase amont du projet WARUNA, nous avons implémenté le *pattern* SPC pour le langage de conception AADL ainsi que le test d'ordonnançabilité correspondant.

La plupart des outils existants traitant de performances sont basés sur des heuristiques conservatives. En effet, la plupart des analyses temporelles étant NP-difficiles au sens fort dans les cas autres qu'académiques, les analyses implémentées sont de complexité temporelle polynomiale ou pseudo-polynomiale. Par conséquent, elles ne sont pas exactes, mais pessimistes. Sur certains systèmes de taille raisonnable, une méthode exhaustive peut permettre d'analyser de façon exacte un système de tâches. La seconde contribution consiste en la proposition d'un calcul exact de temps de réponse de bout en bout de tâches sporadiques dépendantes (partageant des ressources critiques et avec des contraintes de précédence) exécutées par un ordonnanceur global à priorités fixes (G-FP) sur une architecture multiprocesseur identique. Pour ce faire, nous utilisons le formalisme des réseaux de Petri temporels paramétriques.

La troisième contribution porte sur la capitalisation des efforts du processus d'analyse. En effet, de nombreux tests d'analyse ont été proposés, notamment par des chercheurs académiques, basés sur la théorie d'ordonnancement et dédiés aux différentes architectures logicielles et matérielles. Cependant, l'une des principale difficultés rencontrées par les concepteurs est de choisir le test d'analyse le plus approprié permettant de valider et/ou de dimensionner correctement leurs conceptions. Cette difficulté se concrétise, dans le milieu industriel, par le peu de tests d'analyse utilisés malgré la multitude de tests proposés. Cette thèse vise donc à faciliter le processus d'analyse, réduire le fossé sémantique entre le modèle métier et les entrées d'analyse. La thèse propose un référentiel d'analyse jouant le rôle d'un dictionnaire de tests, leur contextes pour une utilisation correcte, les outils les implémentant, ainsi qu'un mécanisme pour le choix des tests selon le modèle conçu.

1.3 Organisation de la thèse

Le reste du manuscrit s'organise comme suit :

— Le chapitre 2 introduit les bases sur les systèmes embarqués temps réel. Il présente les définitions, la classification et la structure des systèmes embarqués temps réel ainsi qu'une vue d'ensemble sur les tests d'ordonnançabilité. Ce chapitre présente aussi l'ingénierie dirigée par les modèles et son usage dans le cycle de développement.

— Le chapitre 3 présente les principes généraux de l'ordonnancement temps réel pour les systèmes monoprocesseur, multiprocesseur et les systèmes distribués. Ce chapitre présente également certains outils d'analyse.

— Le chapitre 4 introduit la première contribution sur les contraintes de précédence à base de sémaphore (i.e., *Semaphore Precedence Constraint* - SPC). Il présente l'extension de SPC pour supporter les cycles et présente aussi une nouvelle approche pour visualiser les contraintes basées sur les graphes de précédence.

— Le chapitre 5 présente un test exact des tâches dépendantes sur l'architecture multiprocesseur identique sous G-FP. Le test est basé sur une modélisation par réseau de Petri temporel à stopwatches.

— Le chapitre 6 présente nos contributions quant à la généralisation de l'utilisation de référentiel d'analyse pour l'aide au choix de méthode d'analyse en fonction du modèle de conception à analyser.

— Le dernier chapitre présente la conclusion et les perspectives de cette thèse.

1.4 Publications scientifiques

Les travaux de cette thèse ont donné lieu aux publications suivantes :

— Thanh Dat NGUYEN, Yassine OUHAMMOU et Emmanuel GROLLEAU. **PARAD Repository : On the Capitalization of the Performance Analysis Process for AADL Designs**. In : *European Conference on Software Architecture (ECSA*). Springer, LNCS, 2017.p. 22-39.

— Thanh Dat NGUYEN, Yassine OUHAMMOU, Emmanuel GROLLEAU *et al.* **Design ans analysis of semaphore precedence constraints**. In : *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018. P231-236.

— Thanh Dat NGUYEN, Yassine OUHAMMOU et Emmanuel GROLLEAU. **Towards a model-based framework for prototyping performance analysis tests**. In : *3th Euromicro Conference on Real-Time Systems (ECRTS),WIP Session, 2018*

— Thanh Dat NGUYEN, Yassine OUHAMMOU et Emmanuel GROLLEAU. **Towards a Descriptive Language to Explicitly Define the Applicability of Timing Verification Tests of Critical Real-Time Systems**. In : *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019. p. 450-457.

Première partie

État de l'art

Chapitre 2

Systèmes embarqués temps réel

Sommaire

Les systèmes embarqués temps réel se constituent de différents composants logiciels exécutés sur des ressources matérielles. L'utilisation des ressources matérielles est arbitrée par des algorithmes d'ordonnancement ou des protocoles d'arbitrage. Dans ce chapitre, nous présentons les définitions, les notions de base autour des systèmes temps réel pour avoir une vue globale sur l'architecture d'un système et son fonctionnement d'un point de vue temporel. Le chapitre présente aussi brièvement la théorie de l'ordonnancement temps réel. Le paradigme de l'ingénierie dirigée par les modèles ainsi que quelques langages de modélisation sont également présentés et discutés.

2.1 Introduction

Un système embarqué est un ensemble de matériel et logiciel coopérant pour accomplir des missions spécifiques. Les systèmes embarqués sont désormais devenus omniprésents dans notre vie quotidienne : les téléphones portables, baladeurs, montres numériques, systèmes d'assistance au freinage dans l'automobile ou systèmes de pilote automatique en avionique. Les ressources des systèmes embarqués sont souvent limitées notamment en énergie, taille, mémoire, etc. Un système embarqué peut être un système indépendant ou un sous-système faisant partie d'un grand système.

Définition 2.1.1 *Un système temps réel est un système informatique qui doit réagir avec un comportement correct à des événements d'entrée dans des délais spécifiés [BW01].*

Les systèmes temps réel sont caractérisés non seulement par leur exactitude fonctionnelle, mais également par l'exactitude temporelle [Dav14]. Par conséquent, un système temps réel doit satisfaire les deux types d'exigence suivants [Sta88] :

— Exactitude logique : le résultat généré et calculé par le système doit être correct.

— Exactitude temporelle : le comportement correct du système est défini par le respect du délai des sorties.

Dans les applications critiques, un résultat obtenu après le délai est non seulement tardif mais faux. En fonction des conséquences pouvant survenir du fait d'un délai non respecté, un système temps réel était classifié en trois catégories dans le monde académique [But97] :

— Strict : un système temps réel est dit "strict" si la production des résultats après leur délai peut avoir des conséquences catastrophiques sur le système sous contrôle.

— Ferme : un système temps réel est dit "ferme" si la production des résultats après leur délai est inutile pour le système, mais ne provoque aucun dommage.

— Mou : un système temps réel est dit "mou" si les résultats produits après leur délai ont encore une utilité pour le système, bien que causant une dégradation des performances.

Dans le monde industriel, les systèmes temps réel sont classifiés suivant les normes de sûreté logicielle dépendant du domaine. Dans le domaine électrotechnique, l'IEC 61508 est une norme internationale publiée par la commission électrotechnique internationale (*International Electrotechnical Commission*-IEC) appliquée pour traiter de la sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables. L'analyse de sécurité détermine un niveau d'intégrité de la sécurité (*Safety Integrity Level*-SIL) classé suivant quatre niveaux : SIL 1, le niveau le plus faible, à SIL 4 le niveau le plus élevé. Dans le domaine automobile, la norme ISO 26262 hérite des principes de l'IEC 61508. ASIL (*Automotive Safety Integrity Level*) est une adaptation de SIL pour la norme ISO 26262. Il y a quatre niveaux d'ASIL identifiés par la norme : ASIL A, ASIL B, ASIL C, ASIL D (D correspond au niveau le plus critique). Dans le domaine ferroviaire, il existe aussi des normes qui héritent de la norme IEC 61508 telles que EN 50126, EN 50128 et EN 50129. Dans le domaine de l'avionique civile, les normes DO 178 et DO 254 définissent les niveaux d'assurance de la conception (*Design Assurance Levels*-DAL), suivant cinq niveaux de DAL A (défaillance catastrophique pour la sûreté de fonctionnement) à DAL E (sans effet sur la sûreté de fonctionnement). Dans le cadre de cette thèse, nous nous intéressons aux systèmes temps réel stricts dits critiques.

La suite de ce chapitre est organisée comme suit. La section 2.2 est consacrée à la présentation de la structure des systèmes temps réel en vue de leur ordonnançabilité. La section 2.3 présente une synthèse sur les différents types et familles de tests d'ordonnançabilité.Une présentation du paradigme ingénierie dirigée par les modèles est faite au niveau de la section 2.4. La section 2.5 présente les langages de modélisation abordés lors de cette thèse ainsi qu'une discussion autour de leur usage dans le cycle de développement. Enfin, la section 2.6 conclut ce chapitre.

2.2 Structure des systèmes temps réel

La structure des systèmes temps réel est présentée dans la figure 2.1, composée de trois parties : partie logicielle, partie matérielle et un système d'exploitation temps réel.

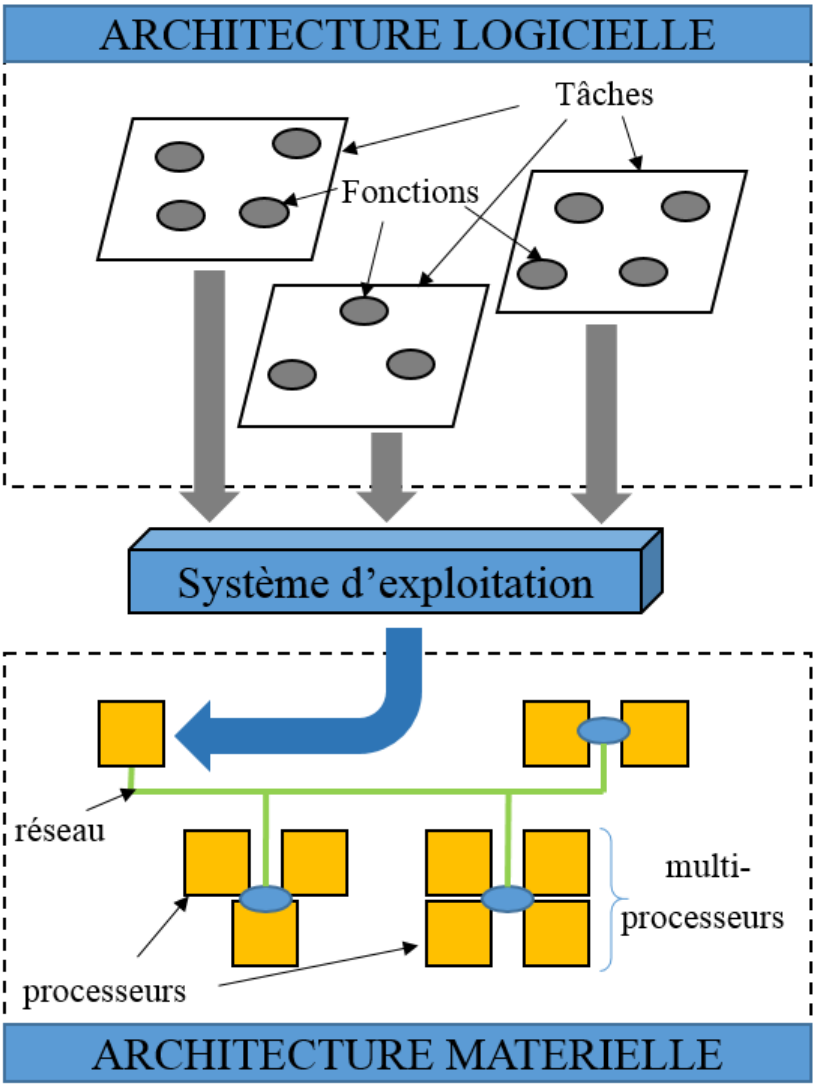

FIGURE 2.1 – Structure des systèmes temps réel

2.2.1 Architecture logicielle

L'architecture logicielle des systèmes temps réel consiste en un ensemble de tâches qui interagissent, généralement en échangeant des messages ou en utilisant des mécanismes de synchronisation (sémaphores, variables conditionnelles, etc.). Les tâches sont les entités logicielles de base d'un système temps réel. Une tâche est un fil d'exécution logique dans un processeur [AB90]. Une tâche représente l'exécution séquentielle d'un ensemble d'instructions. Elle implémente un ensemble de fonctions représentant les besoins du cahier des charges et s'exécutant de façon récurrente avec un certain rythme d'activation.

2.2.1.1 États d'une tâche

Pendant l'exécution d'une application temps réel, une tâche peut passer par différents états (présentés dans la figure 2.2). Le nombre d'états dépend du RTOS, mais au minimum, on trouve

quatre états dans la plupart des RTOS. Une tâche est initialement dans l'état **Endormie**. Quand la tâche se réveille, elle passe à l'état **Prête**, dans cet état la tâche est activée mais en attente d'être choisie par l'ordonnanceur pour être exécutée et donc passer à l'état **En cours d'exécution**. Dans l'état **En cours d'exécution**, une tâche est exécutée par un coeur de processeur. Elle peut au gré de l'ordonnanceur : (i) être préemptée pour retourner à l'état **Prête**, (ii) se mettre en attente d'un message, d'une date, d'un événement ou l'accès à une ressource pour passer à l'état **En attente** ou (iii) finir son exécution ou être suspendue et passer à l'état **Endormie**. A l'état **En attente**, la tâche requiert qu'une condition soit vérifiée pour passer à l'état **Prête**.

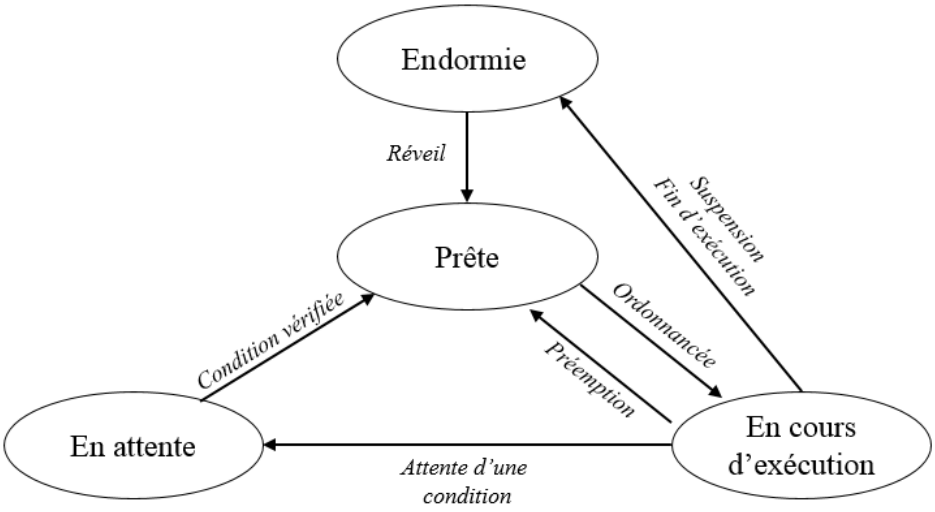

FIGURE 2.2 – États d'une tâche

2.2.1.2 Caractéristiques temporelles d'une tâche

Dans la plupart des modèles académiques de représentation de tâches temps réel celles-ci sont récurrentes. Une tâche est vue comme une suite de travaux, qui représentent chacun une instance d'exécution de la tâche. La figure 2.3 représente le modèle conventionnel de tâche relevant les caractéristiques de base d'une tâche et ses travaux.

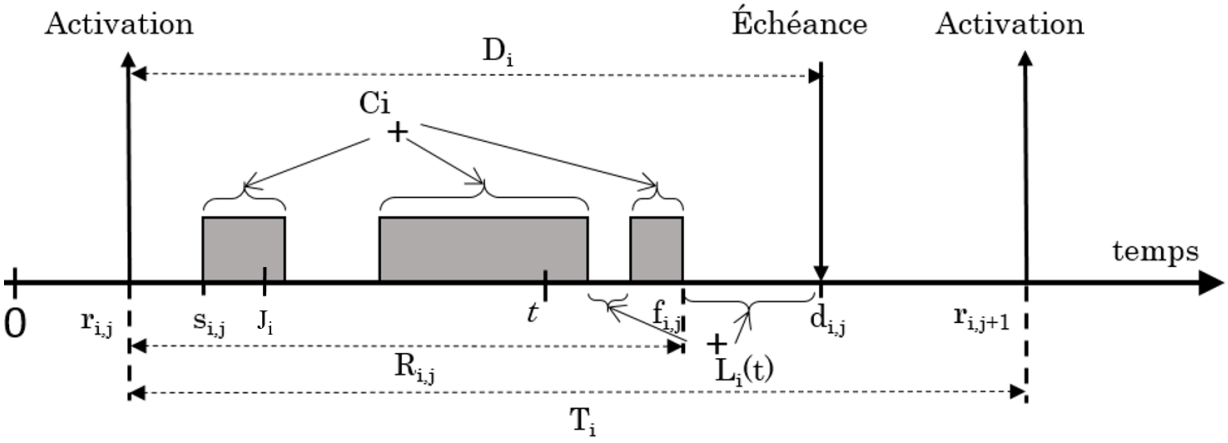

FIGURE 2.3 – Modèle de tâche

Les caractéristiques d'une tâche :

— C_i : (pire durée d'exécution) le temps maximum dont le processeur a besoin pour finir l'exécution d'une instance de la tâche.

— T_i : (période) l'intervalle de temps entre le réveil de deux instances consécutives d'une tâche.

— D_i : (échéance relative) le délai entre réveil d'un travail et son échéance. Si D_i=T_i, la tâche est dite à **échéance sur requête** ou à **échéance implicite**. Les tâches d'un système sont dites à **échéances contraintes** si toutes les délais relatifs sont inférieurs ou égaux aux périodes. S'il n'y a aucune relation entre le délai relatif et la période, le système de tâches est dit à **échéances arbitraires**.

— J_i : la gigue (ou *jitter* en anglais) est le retard maximal d'une activation par rapport à l'activation périodique de celle-ci.

Les caractéristiques des travaux :

— $r_{i,j}$: (date de réveil ou date d'activation) date où la j-ième instance de la tâche τ_i est activée. Si la date de réveil est connue a priori, la tâche est dite **concrète**. Si toutes les dates de réveil des tâches d'un système sont connues et égales, le système de tâches est dit **synchrone**, autrement le système de tâches est dit **différé**. La date de réveil de la première instance correspond à l'*offset* de la tâche.

— $s_{i,j}$: (date de début) date où la j-ième instance de la tâche τ_i commence son exécution.

— $f_{i,j}$: (date de fin) date où la j-ième instance de la tâche τ_i finit son exécution.

— $R_{i,j}$: (temps de réponse) le temps nécessaire pour que la j-ième instance de la tâche finisse $R_{i,j}$=$f_{i,j}$-$r_{i,j}$. Le temps de réponse maximal parmi les instances d'une tâche représente le pire temps de réponse de la tâche.

— $d_{i,j}$: (délai absolu) date avant laquelle la j-ième instance de la tâche τ_i doit finir ses travaux, $d_{i,j}$=$r_{i,j}$+D_i.

— $L_i(t)$: (laxité) le temps (variable selon l'instant de calcul t) pendant lequel une tâche peut être retardée sans manquer son échéance, $L_i(t) = d_i(t) - t - c_i(t)$ où $d_i(t)$ est l'échéance du travail en cours pour la tâche τ_i à l'instant t, et $c_i(t)$ est la durée d'exécution restante pour ce travail.

Selon la périodicité, on peut classifier les tâches en trois types :

— Tâche périodique : la tâche s'active à des intervalles réguliers de temps T_i. Nous avons alors : $r_{i,j+1}$=$r_{i,j}$+T_i.

— Tâche sporadique : la période d'activation n'est pas connue a priori, deux activations consécutives sont séparées par un intervalle minimum de temps. Alors $r_{i,j+1} \geq r_{i,j} + T_i$. La tâche périodique est un cas particulier de tâche sporadique.

— Tâche apériodique : elle est souvent activée par l'arrivée des événements qui peuvent se passer à tout moment. Il n'existe donc pas de période (même l'intervalle minimal séparant deux instances) ni de date de réveil.

2.2.2 Architecture matérielle

La partie matérielle des systèmes temps réel se compose des processeurs, mémoires, réseaux, dispositifs d'entrées/sorties, etc. L'architecture matérielle des systèmes est souvent classifiée selon le nombre de processeurs : monoprocesseur et plusieurs processeurs. Lorsqu'une application s'exécute sur plusieurs processeurs, nous pouvons distinguer deux types de systèmes : le système multiprocesseur et le système distribué (ou réparti).

2.2.2.1 Architecture multiprocesseur

Le système multiprocesseur (fortement couplé dans l'académique) est un système qui a plusieurs processeurs connectés par un bus interne et qui partagent souvent une mémoire commune. Le coût de communication inter-processeur est souvent considéré négligeable, il peut avoir une

horloge commune, voire un ordonnanceur commun (dit global). Les tâches peuvent migrer d'un processeur à l'autre. La figure 2.4 présente l'architecture matérielle des systèmes multiprocesseurs.

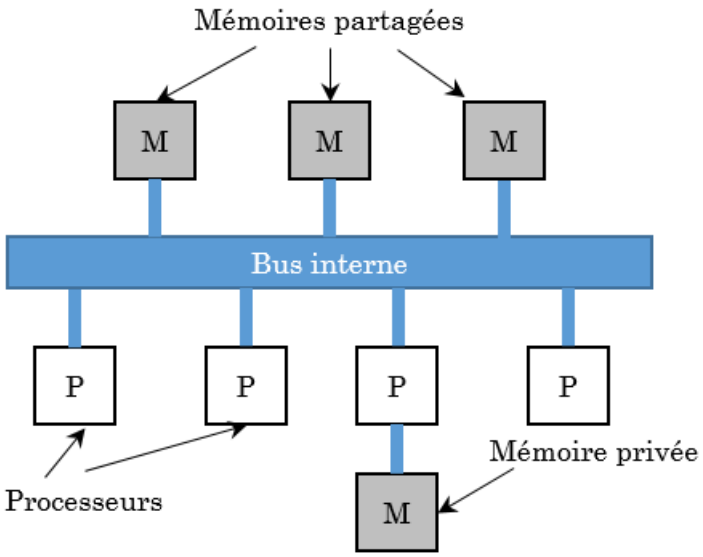

FIGURE 2.4 – Architecture matérielle des systèmes multiprocesseur

Il existe trois types d'architectures multiprocesseurs :

— Architecture à processeurs hétérogènes : les processeurs sont différents donc le temps d'exécution d'une tâche dépend à la fois du processeur et de la tâche.

— Architecture à processeurs homogènes : tous les processeurs sont identiques. Par conséquent, le temps d'exécution de toutes les tâches est le même sur tous les processeurs.

— Architecture à processeurs uniformes : le temps d'exécution d'une tâche dépend de la cadence du processeur. Un processeur de cadence 2xC exécute deux fois plus vite qu'un processeur de cadence C.

2.2.2.2 Architecture distribuée

Le système à architecture distribuée (faiblement couplée) est un système qui est composé de plusieurs noeuds reliés entre eux par un ou plusieurs réseaux. L'architecture de chaque noeud peut être de type monoprocesseur ou multiprocesseur. Généralement, la communication entre les noeuds a un coût de temps non-négligeable et l'arbitrage du réseau est considéré dans les temps de transmission. Les processeurs de différents noeuds fonctionnent de manière indépendante (pas d'horloge commune). Les tâches ne migrent généralement pas entre les processeurs appartenant à différents noeuds. Un exemple d'architecture distribuée est présenté sur la figure 2.5.

L'architecture des systèmes distribués pourrait être classifiée selon les types des réseaux qui peuvent d'être de différentes topologies (e.g., bus, maillée, etc.) et arbitrés par différents protocoles de communication (e.g., TDMA, CSMA, etc.).

2.2.3 Système d'exploitation

Un système d'exploitation temps réel (*Real-Time Operating System*-RTOS) sert d'intermédiaire entre la partie logicielle et la partie matérielle. Le RTOS gère les ressources matérielles et fournit un mécanisme d'arbitrage permettant le partage des ressources. Ce qui nous intéresse le plus dans le cadre de cette thèse est le mécanisme d'arbitrage des ressources de calcul, appelé

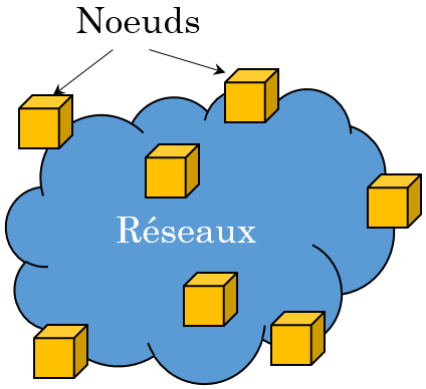

FIGURE 2.5 – Architecture matérielle des systèmes distribués

ordonnanceur (*scheduler* en anglais), qui est responsable de l'ordre d'exécution des tâches sur les processeurs de façon à garantir le déterminisme temporel.

Définition 2.2.1 *La politique d'ordonnancement est la stratégie utilisée par l'ordonnanceur pour établir l'ordre d'exécution des tâches.*

Un système de tâches est **ordonnançable** selon une politique d'ordonnancement donnée si l'ordonnancement produit par cette politique respecte toutes les échéances. Un système de tâches est **faisable** s'il existe une politique d'ordonnancement sous laquelle le système de tâches est ordonnançable [DB11b]. Nous pouvons classifier les politiques d'ordonnancement selon les caractéristiques d'ordonnancement suivantes :

— **Preemptif et Non-preemptif** : sous les politiques d'ordonnancement **préemptif**, l'instance de la tâche en cours d'exécution peut être interrompue à tout moment pour exécuter une autre instance de tâche plus prioritaire conformément à la politique d'ordonnancement choisie. Sous les politiques d'ordonnancement **non-préemptif**, une instance de tâche, une fois commencée son exécution, ne peut pas être interrompue jusqu'à son achèvement. Dans ce cas, toutes les décisions d'ordonnancement sont prises lorsque l'instance de tâche termine son exécution.

— **Statique et Dynamique** : les politiques d'ordonnancement **statique** se basent sur les paramètres fixes (e.g., période, délai relatif) pour affecter les priorités aux tâches avant leur exécution. Dans les politiques d'ordonnancement statique, nous pouvons distinguer deux classes : **priorités fixes aux tâches** (e.g., RM, DM) et **priorités fixes aux travaux** (e.g., EDF [Der74]). L'ordonnancement **dynamique** se base sur les paramètres dynamiques qui peuvent changer au cours d'exécution (e.g., laxité). La politique d'ordonnancement dynamique typique est LLF (*Least Laxity First* [Mok83]).

— **En ligne et Hors ligne** : les politiques **hors ligne** consistent à construire en avance la séquence complète d'ordonnancement des tâches qui se répète après un intervalle de temps spécifique. Les politiques d'ordonnancement **en ligne** sont capables à tout moment de choisir la prochaine tâche à exécuter en fonction des paramètres temporels des tâches. Ces politiques sont basées sur la notion de **priorité**. Les décisions d'affectation de priorité sont prises soit avant l'exécution du système (politiques à priorités fixes aux tâches), soit durant cette exécution (politiques à priorités fixes aux travaux).

Parmi les politiques d'ordonnancement, l'optimalité, la dominance, l'équivalence et l'incomparabilité sont des critères de comparaison à considérer dans le choix des politiques d'ordonnancement.

Définition 2.2.2 *Une politique d'ordonnancement A est dite optimale pour une classe de systèmes dans une catégorie de politiques d'ordonnancement C et selon les hypothèses posées, si et*

seulement si elle peut ordonnancer fiablement tout système ordonnançable par une politique de cette classe.

Par exemple, la politique d'ordonnancement RM (*Rate Monotonic* [LL73a]) est optimale dans la catégorie des algorithmes à priorités fixes aux tâches pour les classes de systèmes de tâches préemptibles, indépendantes, périodiques synchrones ou sporadiques, et à échéance sur requête sur une architecture monoprocesseur.

Définition 2.2.3 *La politique d'ordonnancement A est dominante pour une classe de systèmes par rapport à la politique d'ordonnancement B si tous les systèmes de tâches ordonnançables par B sont ordonnançables par A.*

Par exemple, la politique d'ordonnancement DM (*Deadline Monotonic* [LW82]) domine la politique d'ordonnancement RM (*Rate Monotonic* [LL73a]) pour les classes de systèmes de tâches préemptibles, indépendantes, périodiques synchrones ou sporadiques, et à échéances contraintes sur une architecture monoprocesseur.

Définition 2.2.4 *La politique d'ordonnancement A et la politique d'ordonnancement B sont équivalentes si tous les systèmes de tâches ordonnançables par A sont aussi ordonnançables par B et inversement.*

Définition 2.2.5 *La politique d'ordonnancement A et la politique d'ordonnancement B sont incomparables si A peut ordonnancer fiablement quelques systèmes de tâches qui ne sont pas ordonnançables par B et inversement.*

2.3 Caractéristiques des tests d'ordonnançabilité

Les tests d'ordonnançabilité visent à déterminer si un système de tâches sous une politique d'ordonnancement donnée arbitrant des ressources données respectera ses contraintes temporelles.

Définition 2.3.1 *Un test d'ordonnançabilité est défini comme une* ***condition suffisante*** *si tous les systèmes de tâches qui sont considérés comme ordonnançables selon le test sont en fait ordonnançables. Un test d'ordonnançabilité est défini comme une* ***condition nécessaire*** *si tous les systèmes de tâches qui sont considérés comme non-ordonnançables selon le test sont en fait non-ordonnançables. Un test d'ordonnançabilité est une* ***condition exacte*** *s'il est à la fois nécessaire et suffisant.*

Un test d'ordonnançabilité est utilisé pour vérifier l'ordonnançabilité d'un système de tâches en se basant sur son comportement pire cas. Le résultat d'analyse doit rester valide quand le système en réalité se comporte mieux que dans le pire cas. Pour cela, le test d'ordonnançabilité doit être **viable**.

Définition 2.3.2 *Un test d'ordonnançabilité pour une politique d'ordonnancement est* ***viable*** *si un système considéré comme ordonnançable par le test reste ordonnançable lorsque les paramètres d'une ou de plusieurs instances de tâches sont modifiées d'une, de quelque ou de toutes les manières suivantes : (i) réduction de temps d'exécution, (ii) augmentation de périodes (iii) réduction de gigues et (iv) augmentation de délais relatifs [BB06].*

2.3.1 Complexité

Les tests d'ordonnançabilité sont caractérisés aussi par leur complexité algorithmique. La complexité algorithmique représente le temps (complexité temporelle) ou la taille de l'espace (complexité spatiale) nécessaire pour résoudre le problème. La complexité est estimée en fonction de la taille des entrées du problème. La complexité d'un algorithme f est toujours représentée par la borne supérieure notée O(f) (dite grand O de f). $g(n) \in O(f(n))$ si $lim_{n\to\infty}|\frac{g(n)}{f(n)}| < \infty$, n est la taille des entrées de l'algorithme.

— si f(n) = c (constant), la complexité est appelée **constante**.

— si f(n) = n, la complexité est appelée **linéaire**.

— si f(n) = n^p, la complexité est appelée **polynomiale**.

— si f(n) = 2^n, la complexité est appelée **exponentielle**.

— si f est un polynôme de la valeur numérique de l'entrée (mais pas nécessairement de la taille de l'entrée), la complexité est dite **pseudo-polynomiale**.

2.3.2 Modèles de tâches

Beaucoup de modèles de tâche ont été proposés de sorte à représenter de mieux en mieux le comportement réel des tâches. Dans ce contexte, l'expressivité, la complexité des tests et le pessimisme sont des facteurs à prendre en considération pour le choix de modèles de tâche. Le premier modèle de tâche bien connu de Liu&Layland [LL73a] a été proposé en 1973. Il caractérise le comportement des tâches par seulement deux paramètres : le temps d'exécution et la période d'activation. L'échéance est égale à la période, les tâches sont considérées indépendantes et préemptibles. Ce modèle de tâches a été étendu suivant différentes dimensions. Une des dimensions est la dépendance prenant en compte des interactions particulières que les tâches peuvent avoir (précédences, exclusions mutuelles, etc.). Une autre dimension est la caractérisation plus fine des dates d'activation, par exemple le modèle *multiframe* [MC97] dont la tâche est composée de *frames*, chaque *frame* a son propre temps d'exécution. Formellement, la tâche est représentée par un tuple (E, P) où $E = [E_0, E_1, ..., E_{N-1}]$ est le vecteur de temps d'exécution et P est le temps minimum de séparation entre deux *frames* successives, l'échéance des *frames* est égale à P. Le modèle *multiframe* a été encore généralisé par le modèle *generalized multiframe* (GMF) [BCGM99] dont les échéances des *frames* peuvent différer du temps minimum de séparation. De plus, les *frames* n'ont pas nécessairement les mêmes échéances et le temps minimum de séparation n'est pas identique pour toutes les *frames*. Formellement, une tâche GMF est caractérisée par un tuple (E, D, P) où $E = [E_0, E_1, ..., E_{N-1}]$ est le vecteur des temps d'exécution, $D = [D_0, D_1, ..., D_{N-1}]$ est le vecteur des échéances et $P = [P_0, P_1, ..., P_{N-1}]$ est le vecteur de délais minimum de séparation. Une autre extension du modèle proposé par Liu&Layland est le modèle fonction de distance. Une fonction de distance minimale (respectivement maximale) $\delta^- : \mathbb{N} \to \mathbb{N}$ (respectivement δ^+) retourne, pour chaque nombre d'instance d'une tâche ($q \in \mathbb{N}$), une borne inférieure (respectivement supérieure) sur la longueur de chaque intervalle contenant q activations successives d'une chaîne fonctionnelle. Le modèle de tâches sporadiques proposé par Mok [Mok83] étend le modèle fonction de distance minimale en précisant que le nombre d'activations est deux et la chaîne ne contient qu'une tâche (i.e., $\delta_a^-(2)$). Dans ce modèle, une tâche est caractérisée par le temps d'exécution, l'échéance et le délai minimal inter-arrivée. L'échéance n'est pas nécessairement égale au délai minimal inter-arrivée. Palencia et Harbour [PH98, RGRR12] proposent le modèle de transaction dont le système se compose de transactions périodiques activées par des événements externes non concrets. Chaque transaction se compose de tâches à offset, la date de réveil relative à la date de l'événement déclencheur de la transaction. Les modèles de tâches récurrentes (*Recurring Branching Tasks* [Bar98], *Recurring model* [Bar03], *Non-cyclic Recurring model* [Bar10]) réduisent le pessimisme des anciens modèles utilisant le pire temps d'exécution constant car ils représentent la tâche par un graphe orienté acyclique

(*Directed Acyclique Graph* - DAG). C'est intéressant si la tâche contient du code conditionnel. En effet, comme plusieurs chemins d'exécution sont possibles, chaque chemin correspond à une branche dans le DAG avec un temps d'exécution différent. Le modèle GMF est généralisé par le modèle non-cyclique GMF [MNLM10] dont les *frames* s'exécutent dans n'importe quel ordre. Le modèle de tâches récurrentes est généralisé par le modèle digraphe [SEGY11a, SEGY11b] permettant de modéliser la boucle. La relation hiérarchique des modèles représentée dans la figure 2.6 montre que plus le modèle est expressif, plus l'analyse de ce modèle devient complexe.

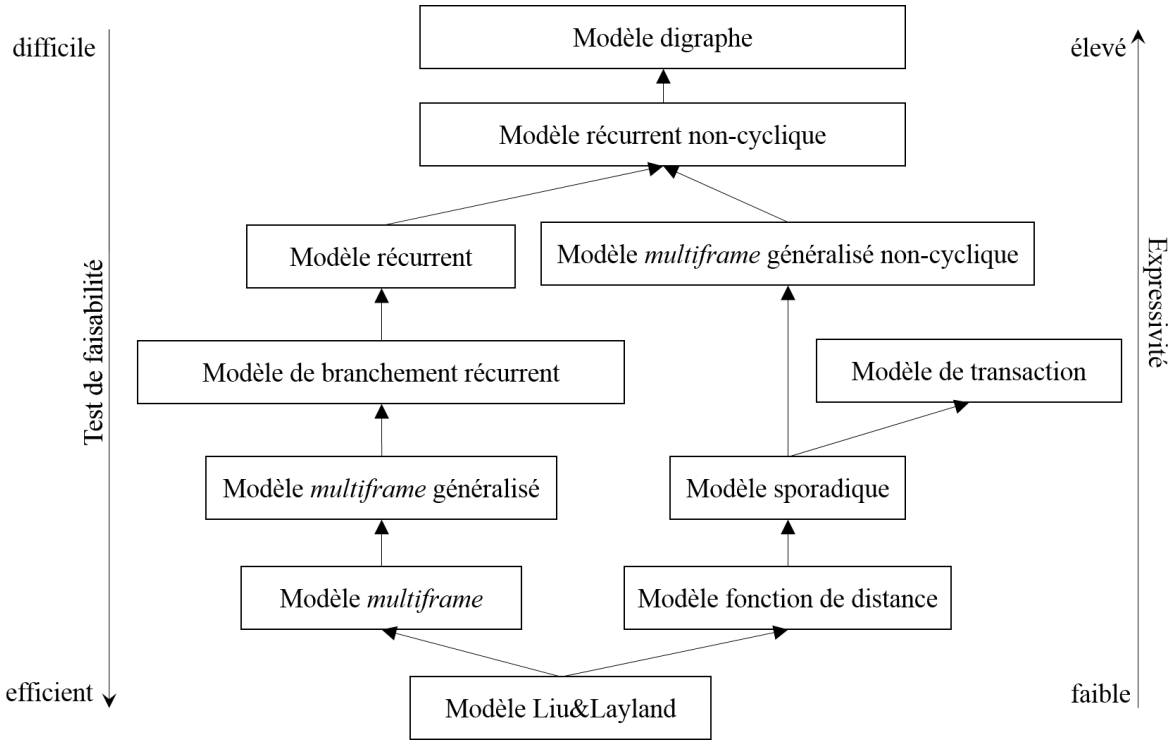

FIGURE 2.6 – Modèles de tâche [SY15]

2.3.3 Familles de tests

On peut distinguer cinq familles principales d'analyse : l'analyse de l'utilisation processeur, l'analyse de la demande de processeur, l'analyse de temps de réponse, la simulation et la vérification de modèle.

2.3.3.1 Analyse de l'utilisation du processeur (*Processor Utilisation Analysis*)

Cette analyse est basée sur la charge du processeur définie par $U_i = \frac{C_i}{T_i}$. Cette charge du processeur doit souvent être inférieure à un seuil spécifique pour que le système soit ordonnançable. L'analyse de l'utilisation processeur peut être utilisée comme une condition suffisante, par exemple dans le contexte des tâches indépendantes à échéance sur requête ordonnancées par RM sur un processeur, on pourra utiliser la condition suffisante de faisabilité de Liu&Layland [LL73a], $U = \sum_{i=1}^{n} U_i \leq n(2^{\frac{1}{n}} - 1)$ (n est le nombre de tâches), ou la condition suffisante de faisabilité basée sur la borne hyperbolique [BBB03] $\prod_{i=1}^{n} (U_i + 1) \leq 2$. La densité est une variante conservative (i.e. pouvant se substituer à l'utilisation processeur U_i afin d'obtenir un pire cas) dans le cas où les tâches sont à échéances contraintes, celle-ci est donnée par $\frac{C_i}{min(D_i,T_i)}$.

2.3.3.2 Analyse de la demande de processeur (*Processor Demand Analysis*)

Cette analyse repose sur le calcul de la demande cumulée des exécutions des tâches réveillées et terminées dans un intervalle de temps (*Demand Bound Function* [JS93a, BRH90]). L'analyse consiste à vérifier que la demande cumulée (la charge totale à traiter) à n'importe quel moment est toujours inférieure ou égale à la largeur de l'intervalle. Cette analyse permet d'obtenir un test de faisabilité qui est exact pour les algorithmes d'ordonnancement optimaux en monoprocesseur, comme EDF pour les tâches indépendantes.

2.3.3.3 Analyse des temps de réponse (*Response Time Analysis* [JP86a, Leh90])

Cette analyse permet de calculer la longueur d'une période d'activité qui permet de déduire le pire temps de réponse d'une tâche dans le contexte monoprocesseur, pour les algorithmes d'ordonnancement à priorités fixes aux tâches. Cette analyse, de complexité pseudo-polynomiale, considère un instant critique, elle est exacte pour les systèmes de tâches indépendantes, mais seulement suffisante (i.e. donne une borne supérieure du temps de réponse) lorsqu'il ne peut pas exister d'instant critique. Elle a été adaptée pour prendre en compte les durées de blocage lorsqu'il existe des parties non préemptibles (exclusions mutuelles, ou non préemptibilité de certaines tâches), ou encore les déclenchement des tâches par messages pouvant retarder leur démarrage par rapport à leur période, qui est pris en compte sous forme d'une gigue d'activation. Si le pire temps de réponse obtenu pour une tâche est inférieur à son délai critique, celle-ci respectera donc toutes ses échéances.

2.3.3.4 Simulation

Cette technique consiste à simuler l'exécution du système de tâches sur un intervalle de temps suffisamment grand pour détecter tout mauvais comportement du système. L'intervalle de temps doit être suffisamment grand pour couvrir tous les comportements possibles du système. L'intervalle de simulation maximal dans le contexte monoprocesseur est $[0, \max(r_i) + 2ppcm(T_1, ..., T_n))$ [LM80]. La séquence d'ordonnancement infinie générée par le système de tâches est représentée par l'équation 2.1 dont $r = max(r_i)$ et $P = ppcm(T_1, ..., T_n)$ (**ppcm** représente plus petit commun multiple). La séquence $\sigma_{[0..r+P)[}$ est la partie acyclique, tandis que la séquence σ^* est la partie cyclique de longueur hyperpériode (i.e., P), elle se répète jusqu'à l'infini.

$$\sigma = \sigma_{[0..r+P[}\sigma^*_{[r+P..r+2P[} \tag{2.1}$$

Grolleau et Choquet dans [GCG00a] ont proposé une autre approche pour définir l'intervalle de simulation de façon exacte à l'aide des temps creux acycliques. Les temps creux acycliques surviennent dans la partie acyclique dans la séquence d'ordonnancement et les temps creux cycliques surviennent dans la partie cyclique et ils apparaissent périodiquement à chaque hyperpériode. Dans la partie cyclique, un intervalle de longueur hyperpériode P a exactement $P(1-U)$ unités de temps creux. Or, un intervalle de longueur P dans n'importe quelle partie a au moins $P(1-U)$ temps creux grâce à l'existence des temps creux acycliques. La méthode de simulation consiste à décaler une fenêtre de longueur P et compter les temps creux dedans. Ce processus s'arrête quand le nombre de temps creux dans la fenêtre est exactement $P(1-U)$. L'intervalle de simulation sera de 0 à la borne supérieure de la fenêtre de test.

Exemple : soit un système de tâches $S = \{\tau_i < r_i, C_i, D_i, T_i >\} = \{\tau_1 < 0, 1, 3, 3 >, \tau_2 < 2, 1, 4, 4 >, \tau_3 < 3, 2, 6, 6 >\}$ ordonnancé par DM (*Deadline Monotonic*). L'utilisation processeur est $U = 11/12$ donc le nombre de temps creux cyclique est 1 pour chaque hyperpériode. Le processus de déterminer l'intervalle de simulation est illustré dans la figure 2.7, ce processus s'arrête quand la fenêtre de test a exactement 1 temps creux. Initialement, la simulation est construite sur $[0, 12[$. Comme nous y observons 2 temps creux, le premier temps creux, se terminant à l'instant 2, ne peut faire partie du cycle. Le cycle est alors recherché sur $[2, 14[$. Comme il y a

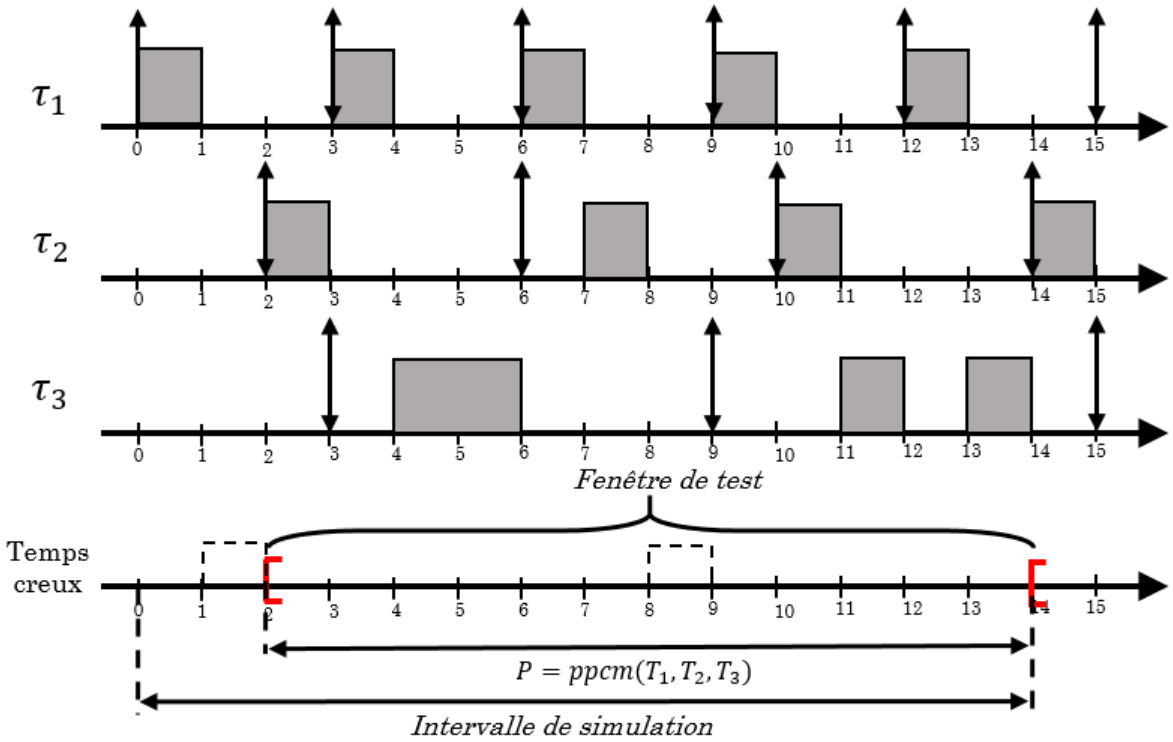

FIGURE 2.7 – Exécution du système de tâches sous *DM*

un seul temps creux sur cet intervalle, la simulation s'arrête.
Cet intervalle réduit significativement la longueur de l'intervalle de simulation par rapport à l'intervalle de Leung et Merrill [LM80]. La simulation est une condition exacte dans le contexte monoprocesseur et souvent nécessaire dans le contexte multiprocesseur ou distribué. La complexité de cette technique est exponentielle, en plus cette technique n'est pas viable s'il existe des portions de tâches non-préemptibles, ou bien s'il existe des contraintes de précédence non consistantes avec les priorités (i.e., une tâche moins prioritaire précède une tâche plus prioritaire).

2.3.3.5 Vérification de modèle (*Model checking*)

Cette technique consiste à modéliser le système de tâches sous forme de formalismes spécifiques tels que les réseaux de Petri [CEP03] ou les automates temporisées [Alu99] et vérifier la propriété souhaitée exprimée dans une logique temporelle en parcourant exhaustivement tous les états accessibles.

Cette technique est toujours exacte. Cependant, l'inconvénient de cette technique est l'explosion combinatoire (le nombre d'états augmente exponentiellement en fonction de la complexité du système étudié). Elle ne peut pas passer à l'échelle pour les systèmes industriels complexes.

Un réseau de Petri (*Petri net*) est un graphe bipartite. Il se compose d'un ensemble fini de **places**, représentées par des cercles, d'un ensemble fini de **transitions**, représentées par des rectangles généralement aplatis et enfin d'un ensemble fini d'**arcs orientés** qui relient les places et les transitions. Chaque place peut contenir des jetons. Chaque arc possède un poids qui exprime soit le nombre de jetons consommés si l'arc sort d'une place, soit le nombre de jetons produits si l'arc sort d'une transition. Une transition est **sensibilisée** si toutes les places d'entrée de cette transition possèdent chacune un nombre de jetons supérieur ou égal au poids des arcs reliant ces places à cette transition.

Formellement, un réseau de Petri est un triplet $N = (P, T, W)$ où :

— P est un ensemble fini de places

— T est un ensemble fini de transitions

— $W : P \times T \cup T \times P \to \mathbb{N}$ est la fonction de valuation
  — Pré $W_{|P \times T}$ est la fonction de précondition
  — Post $W_{|T \times P}$ est la fonction de postcondition

L'état d'un réseau de Petri est représenté par la fonction de marquage $M : P \to \mathbb{N}$ ou le vecteur de marquage $M \in \mathbb{N}^{|P|}$. Le marquage initial est noté M_0. Une transition $t \in T$ est sensibilisée à partir du marquage M si et seulement si $\forall p \in P, M(p) \geq W(p,t)$. Le tir de la transition t donne le nouveau marquage $M' : \forall p \in P, M'(p) = M(p) - W(p,t) + W(t,p)$.

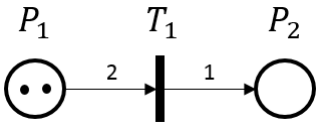

FIGURE 2.8 – Un réseau de Petri simple

La figure 2.8 présente un exemple simple de réseau de Petri. Le réseau contient deux places (i.e., l'ensemble de places $P = P_1, P_2$) et une transition (i.e, l'ensemble de transitions $T = T_1$) et la place P_1 est marquée par deux jetons, la place P_2 n'a pas de jeton (le marquage initial est donc $M_0 = [2, 0]$). $M(P_1) = 2 \geq 2 = W(P_1, T_1)$ ($W(P_1, T_1)$ est le poids de l'arc (P_1, T_1)), la transition est donc sensibilisée. Lorsque la transition est tirée, deux jetons de la place P_1 sont enlevés (i.e., consommés), et un jeton est généré (i.e., produit) dans la place P_2 puisque le poids de l'arc $(P_1, T_1) : W(T_1, P_1) = 1$.

Dans cette thèse, nous nous focalisons sur le réseau de Petri temporel (*Time Petri Net* - TPN). Le réseau de Petri temporel [MF76] est une extension du réseau de Petri classique où chaque transition est associée à un intervalle de temps. Cet intervalle spécifie les dates de tirs possibles.

Formellement, un réseau de Petri temporel est un n-uplets $N = (P, T, W, \alpha, \beta)$ où :

— (P, T, W) est un réseau de Petri classique avec $M_0 \in \mathbb{N}^{|P|}$ est le marquage initial
— Date de tir statique au plus tôt $\alpha : T \to \mathbb{Q}^+$
— Date de tir statique au plus tard $\beta : T \to \mathbb{Q}^+ \cup \{+\infty\}$
  — Intervalle de tir statique de la transition $t : [\alpha(t), \beta(t)]$ avec $\alpha(t) \leq \beta(t)$

Ainsi, soit θ la date où la transition t est sensibilisée, la transition ne peut être tirée qu'à partir de l'instant $\theta + \alpha(t)$ et avant l'instant $\theta + \beta(t)$. Bien sûr, si la transition n'est plus sensibilisée à $\theta + \beta(t)$, elle ne sera pas franchie. On comprend mieux la sémantique de franchissement temporel si on munit chaque transition sensibilisée d'une horloge : lorsque la transition devient nouvellement franchissable (i.e. elle devient sensibilisée ou elle vient d'être franchie et est encore sensibilisée après franchissement), son horloge est mise à zéro. Lorsque son horloge atteint la date de tir statique au plus tôt, la transition peut être franchie. Elle doit être franchie au plus tard (si elle est encore sensibilisée) lorsqu'elle atteint sa date de tir statique au plus tard.

L'état d'un réseau de Petri temporel est représenté par un couple formé d'un marquage et d'une application qui associe à chaque transition valide un intervalle temporel. Le réseau de Petri temporel s'exécute en temps continu, en théorie il y a donc une infinité d'états possibles. Il faut regrouper des états *équivalents* de façon à obtenir un graphe d'états fini si le marquage est fini. Il existe plusieurs façons d'opérer des regroupements d'états équivalents telles que :

— Graphe de classes [BM83, BD91]
— Graphe de régions géométriques [YR98]
— Graphe de classes fort [BV03]
— Graphe basé zones [GRR06]
— Graphe de classes atomique [BH06]

Chaque graphe peut préserver des types de propriétés différents. A titre d'exemple, nous présentons succinctement le graphe des classes. Ce graphe se compose de classes d'état. L'intervalle temporel associé est relatif à la date de sensibilisation de la transition. On reprend un exemple de réseau de Petri temporel issu de [LR06] présenté dans la figure 2.9 suivante :

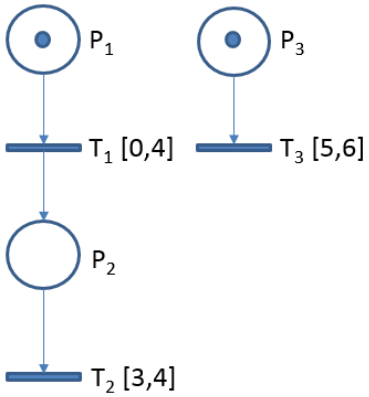

FIGURE 2.9 – Exemple de réseau Petri temporel

Le graphe de classes associé à ce réseau de Petri est présenté ci-dessous.

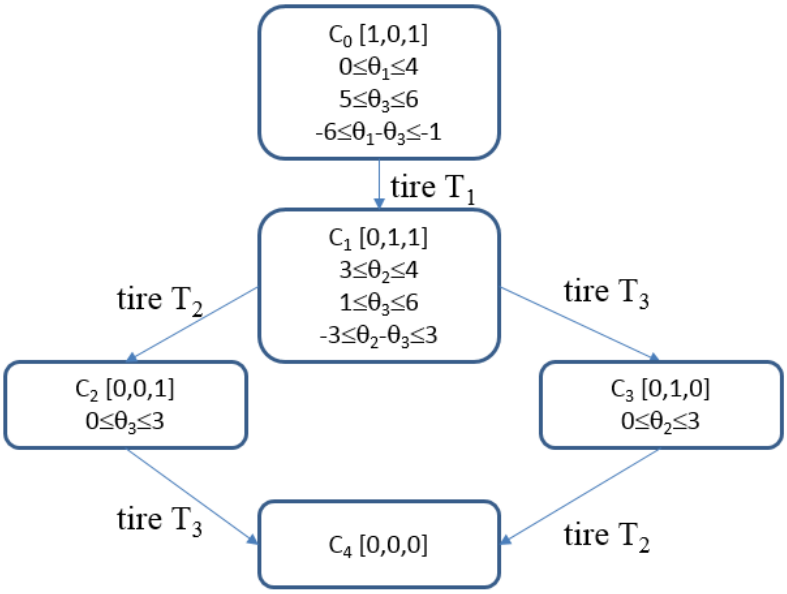

FIGURE 2.10 – Graphe de classes de l'exemple de la figure 2.9

Dans ce graphe de classes, C_i représente le marquage sous forme de vecteur, θ_i représente la date de tir des transitions. Au début, il n'y a que les places P_1 et P_3 qui ont un jeton, donc le vecteur de marquage est $[1, 0, 1]$, la transition T_1 ne peut être tirée avant l'instant 0 et après l'instant 4 donc $0 \leq \theta_1 \leq 4$. Similairement pour la transition T_3, on a $5 \leq \theta_3 \leq 6$. Depuis les deux inéquations de deux transitions, on déduit que $-6 \leq \theta_1 - \theta_3 \leq -1$. Après, T_1 est tirée donc $\theta_1 = 0$, donc $1 \leq \theta_3 \leq 6$, on a $3 \leq \theta_2 \leq 4$, on peut déduire que $-3 \leq \theta_2 - \theta_3 \leq 3$, le vecteur de marquage devient $[0, 1, 1]$. Dans cet état, les deux transitions T_2 et T_3 sont en même temps tirables, l'exécution de deux chemins sont semblables. Si on tire T_2, le vecteur de marquage devient $[0, 0, 1]$, on a $0 \leq \theta_3 \leq 3$ et il reste T_3 tirable, si on la tire le vecteur de marquage devient $[0, 0, 0]$. Le graphe des classes permet donc de regrouper plusieurs états jugés équivalents dans une classe d'états, avec des contraintes sur les horloges des transitions. Cependant, ce graphe ne préserve pas toutes les propriétés du réseau de Petri. Ainsi, dans ce graphe, il est possible d'exhiber un chemin dans lequel T_1 est franchie à l'instant 4, puis T_2 trois unités de temps plus

tard, alors que sur le réseau de Petri, si T_1 est franchie à l'instant 4, alors T_3 doit être franchie avant T_2, entre une et deux unités de temps plus tard.

Le réseau de Petri temporel avec arcs inhibiteurs à stopwatch est une extension du réseau de Petri temporel qui permet de représenter la suspension et la reprise d'actions. Concrètement, ce nouveau type dr réseau de Petri présente un nouvel arc : l'arc inhibiteur à stopwatch (voir Figure 2.11). L'arc inhibiteur relie une place à une transition. Une transition t sera **inhibée** par un marquage M si la place connectée à t par un arc inhibiteur possède le nombre de jetons supérieur ou égal au poids de l'arc inhibiteur. Cependant, lorsque la place est désinhibée, au lieu de considérer la transition nouvellement franchissable, et réinitialiser son horloge, celle-ci conserve la valeur d'horloge qu'elle possédait au moment où elle avait été inhibée.

Formellement, un réseau de Petri temporel avec arcs inhibiteurs à stopwatch est défini par $N = (P, T, W, \alpha, \beta, I)$ où :

— (P, T, W, α, β) est un réseau de Petri temporel

— $I : P \times T \to \mathbb{N}$ est la fonction d'inhibition à stopwatch

Une transition t est inhibée si $\exists p \in P, 0 < I(p,t) \leq M(p)$, le temps pour cette transition est alors gelé, ainsi l'horloge de la transition conserve sa valeur.

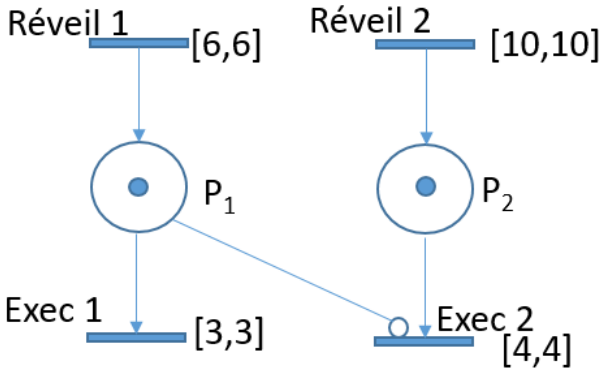

FIGURE 2.11 – Réseau de Petri temporel avec arc inhibiteur

La figure 2.11 présente un exemple d'un réseau de Petri temporel avec arc inhibiteur à stopwatch. Le réseau de Petri y modélise deux tâches périodiques exécutées sur un processeur : τ_1 à gauche, est de période 6 et de durée 3, alors que τ_2, à droite, est de période 10 et de durée 4. Les tâches sont ordonnancées suivant une politique à priorités fixes aux tâches, affectant la plus grande priorité à τ_1. A l'instant 0, la place P_1 a un jeton donc la transition $Exec2$ est inhibée, la valeur de son horloge valant 0 est conservée. Après 3 unités de temps, $Exec1$ est franchie et le jeton de P_1 est consommé. La transition $Exec2$ est sensibilisée et son horloge démarre. Après 3 unités de temps, à l'instant 6, un jeton est produit à la place P_1, la transition $Exec2$ est inhibée une nouvelle fois, la valeur de l'horloge de la transition valant 3 est conservée. Après 3 unités de temps, à l'instant 9, le jeton à P_1 est consommé par $Exec1$, la transition $Exec2$ est sensibilisée, et son horloge reprend sa valeur de 3. Elle est donc franchie une unité de temps plus tard. Le chronogramme d'exécution correspondant est présenté sur la figure 2.12.

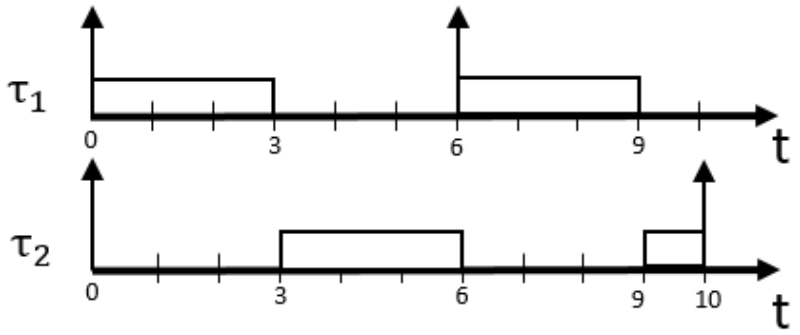

FIGURE 2.12 – Chronogramme d'exécution des tâches correspondant au comportement du réseau de Petri avec inhibiteurs à stopwatch

2.4 Ingénierie dirigée par les modèles

Comme nous nous intéressons à l'ingénierie système d'un point de vue temps réel, nous présentons dans cette section le paradigme nommé ingénierie dirigée par les modèles (IDM) qui est une pierre angulaire dans la conception logicielle des systèmes temps réel. En effet, l'IDM permet de fluidifier le passage d'une étape à l'autre lors des cycles de développement grâce l'interopérabilité entre les outils de modélisation, d'analyse, de transformation et de génération de code.

L'ingénierie Dirigée par les Modèles-IDM (*Model-driven Engineering*) est un paradigme prometteur pour le développement de logiciels. L'IDM est une pratique de l'ingénierie des systèmes qui décrit à travers des modèles le problème posé et sa solution. Le but de l'IDM est d'augmenter le niveau d'abstraction dans la représentation d'un système et l'automatisation de sa construction. Dans l'IDM, les modèles jouent un rôle important dans le développement de logiciels. L'automatisation de la production du système peut être réalisée par des transformations de modèles.

Définition 2.4.1 *Un modèle est une abstraction et une simplification de systèmes permettant de comprendre et de fournir des réponses relatives au système modélisé. Ensuite, un système peut être décrit par différents modèles liés les uns aux autres.*

Les modèles sont considérés comme des entités centrales dans l'IDM. Un modèle est une représentation abstraite d'un (ou partie de) système. Il montre une vue partielle du système en simplifiant ce qui doit être capturé ou automatisé. Par conséquent, une meilleure représentation du système nécessite souvent plusieurs modèles. L'utilisation de modèles a pour objectif d'améliorer la qualité du logiciel obtenu, car il est plus facile de comprendre, de simuler, d'analyser et de valider des modèles abstraits que les programmes informatiques.

Définition 2.4.2 *Un méta-modèle est une abstraction permettant de mettre l'accent sur les propriétés du modèle lui-même. Un méta-modèle décrit les différents types d'éléments du modèle et la façon dont ils sont agencés.*

Pour connaître la nature des différents modèles utilisés dans les systèmes informatiques, on identifie deux relations. La première relation est **représentée par**, indiquant une représentation d'un objet qui est modélisé à travers un modèle. Par exemple, un programme informatique peut être **représenté par** un diagramme de classes. La deuxième relation est **conforme à**, indique la dépendance d'un modèle à un langage de modélisation. Un modèle est **conforme à** son méta-modèle comme un programme est conforme à la grammaire du langage de programmation selon lequel il est écrit [Sch06].

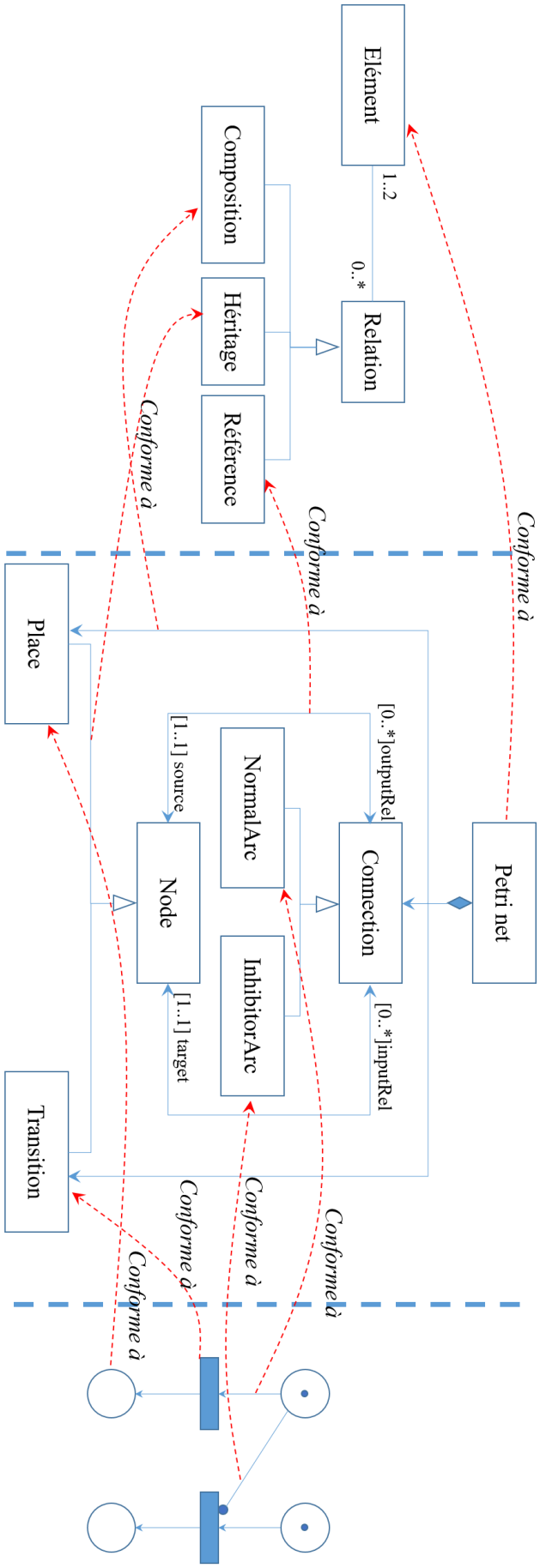

FIGURE 2.13 – Illustration des concepts de méta-modélisation

La figure 2.13 représente la modélisation d'un réseau de Petri (RdP). La partie (a) de la figure représente un RdP avec l'arc inhibiteur. Ce réseau comprend quatre places et deux transitions connectées par les arcs. Les arcs normaux sont représentés par les flèches, cependant l'arc inhibiteur a un cercle à la fin de l'arc. La partie (b) de la figure représente le méta-modèle de réseau de Petri de la partie (a). En effet, un réseau se compose des places (instances de la classe Place), des transitions (instances de la classe Transition) et des connexions (instances de la classe Connection). Les connexions relient les noeuds (instances de la classe *Node*). Un noeud peut être soit une place soit une transition. Il y a deux types de connexions : l'arc normal (instance de la classe *NormalArc*) et l'arc inhibiteur (instance de la classe *InhibitorArc*). Une connexion a une source et une cible qui sont des noeuds. Chaque noeud peut avoir un ensemble d'arcs entrants (ayant le rôle *inputRel*) et un ensemble d'arcs sortants (ayant le rôle*outputRel*). La partie (c) de la figure représente le méta-méta-modèle du réseau de Petri. Ce méta-méta-modèle se compose des éléments et des relations de trois types différents : composition, héritage et référence. Ce méta-méta-modèle est conforme à lui même car si on cherche à méta-modéliser ses éléments on obtiendra un méta-méta-méta-modèle qui lui ressemble.

2.4.1 Méta-modélisation

La méta-modélisation est le processus de définition du méta-modèle d'un **langage de modélisation**. Un **langage de modélisation** est un langage artificiel qui peut être utilisé pour exprimer des informations ou connaissances de systèmes dans une structure. Un langage de modélisation peut être graphique ou textuel.

— Les langages de modélisation graphiques se basent essentiellement sur l'utilisation des diagrammes avec (i) des symboles nommés qui représentent des concepts, (ii) des liens qui relient les symboles et représentent des relations, et (iii) diverses autres notations graphiques pour représenter des contraintes ou des propriétés.

— Les langages de modélisation textuelles peuvent utiliser des grammaires normalisées pour créer des expressions.

Afin d'organiser et de structurer les modèles, l'*OMG* (*Object Management Group*) [OMGc] a défini une architecture appelée ; "Architecture à 4 niveaux" représentée dans la figure 2.14.

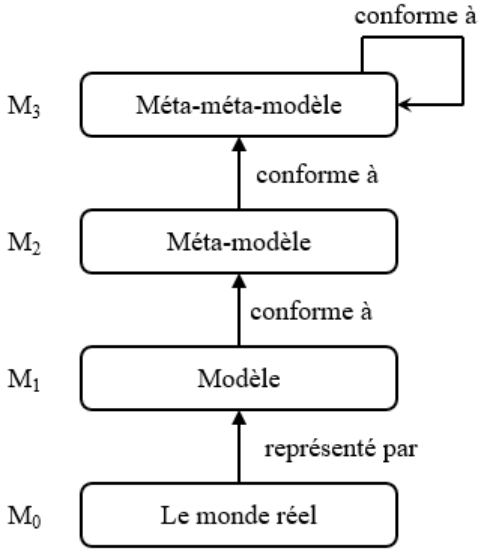

FIGURE 2.14 – Architecture à 4 niveaux définie par l'OMG [WO01]

— Le niveau M3 correspond au MOF (*Meta-Object Facilities*) [OMGe]. Le MOF est un langage de définition de méta-modèles. Les langages de niveau M3 sont conformes à eux-

mêmes, c'est-à-dire qu'ils peuvent se définir. Par exemple, le langage Ecore [Ecl] est une implémentation du MOF, intégrée au framework de devloppement Eclipse sous forme d'un plugin nommé EMF (*Eclipse Modeling Framework*) [emf].

— Le niveau M2 contient les méta-modèles définis par M3, comme les méta-modèles de UML (*Unified Modeling Language*) [G+02], de CWM (*Common Warehouse Metamodel*) [PCTM02], de BPMN (Business Process Management Notation), etc.

— Le niveau M1 contient les modèles qui sont conformes aux méta-modèles du niveau M2, par exemple, les modèles exprimés en diagrammes de classes d'UML.

— Le niveau M0 représente les systèmes réels ou les codes exécutables correspondant aux modèles du niveau M1.

2.4.2 Transformation de modèles

La transformation de modèles permet à un ou plusieurs modèles sources d'être automatiquement transformés en un ou plusieurs modèles cibles selon des règles de transformation. On peut définir des correspondances entre des modèles du même langage ou des modèles de langages différents. Les outils de transformation sont essentiels pour maximiser les avantages de l'utilisation de modèles et minimiser les efforts de construction du système modélisé. Ainsi, ils peuvent considérablement améliorer la productivité du développement logiciel.

Définition 2.4.3 *Une transformation de modèles est une fonction $\phi : S \rightarrow C$, telle que : ϕ prend en entrée un ensemble de modèles source S et génère en sortie un ensemble de modèles cible C. S et C sont les ensembles de modèles conformes à deux ensembles de méta-modèles. Si les deux ensembles de méta-modèles sont identiques, la transformation du modèle est appelée* ***endogène****, sinon elle est appelée* ***exogène****.*

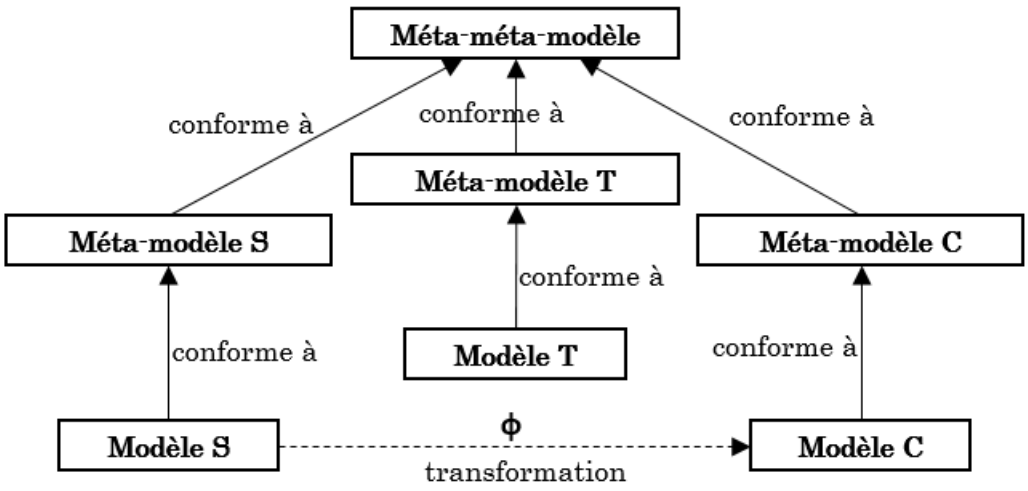

FIGURE 2.15 – Transformation de modèles en modèles en IDM

La figure 2.15 montre une vue d'ensemble du mécanisme de transformation de modèles tel qu'il est utilisé via l'ingénierie dirigée par les modèles. En général, une transformation de modèles est un programme composé d'un ensemble de règles de correspondance. Une règle de transformation est une description de la manière dont un ou plusieurs éléments du langage source peuvent être transformés en un ou plusieurs éléments du langage cible.

On s'intéresse à deux types d'approches de transformation de modèles.

— La transformation de modèles à base de graphe en modèles à base de graphe (*Model to Model* - M2M). Ce type de transformation est supporté par des langages de description de

règles de transformation comme ATL (*Atlas Transformation Language*) [JAB+06] et QVT (*Query-View-Transformation*) [OMGd].

— La transformation de modèles en texte (*Model to Text* - M2T) qui prend en entrée des modèles conformes aux méta-modèles sources et produit des fichiers de texte en sortie. La transformation de modèles en texte est généralement utilisée pour effectuer la génération de code ou générer un format d'échange textuel basé sur une grammaire spécifique. Ce type de transformation est supporté par des outils comme Acceleo [MJL+06] qui est un langage de description de templates.

2.4.3 Langage de modélisation dédié (*Domain Specific Modeling Language* - DSML)

L'ingénierie dirigée par les modèles (IDM) a émergé pour permettre le développement d'applications basées sur la définition des modèles issus du domaine du problème. Pour ce faire, l'IDM utilise les langages de modélisation dédiés (DSML) qui sont des langages de modélisation définis pour un domaine spécifique et permettent aux utilisateurs de travailler directement avec les concepts de ce domaine. Les DSML formalisent la structure, le comportement et les exigences de l'application dans un domaine particulier.

La définition d'un DSML implique trois aspects : la **syntaxe abstraite**, la **syntaxe concrète** et la **sémantique**.

— La **syntaxe abstraite** décrit les concepts du domaine, les relations entre eux et les règles de structuration qui contraignent les éléments. La syntaxe abstraite est généralement définie par un méta-modèle.

— La **syntaxe concrète** décrit la notation utilisée pour représenter les concepts et les instancier. Cette syntaxe peut être textuelle ou graphique.

— La **sémantique** d'un DSML est généralement décrite en langue naturelle (sous forme de document de spécification quand il s'agit d'un standard) ou parfois par des éléments d'autres langages dont la sémantique est bien définie tels que les réseaux de Petri, machines à états finis, etc.

Quand le méta-modèle d'un DSML ne permet pas de répondre précisément à la sémantique du domaine dédié, il peut être enrichi par des contraintes (invariants, pré-conditions, post-conditions) grâce aux langages de définition de contraintes tels qu'OCL. OCL (*Object Constraint Language*) [OMGb] est un langage formel basé sur la logique du premier ordre. C'est un langage complémentaire à la syntaxe abstraite afin de renforcer la sémantique d'un DSML. La vérification des contraintes OCL se fait au niveau modèle au moment de l'instanciation.

2.5 Langages dédiés aux systèmes temps réel

Plusieurs langages de modélisation ont été proposés pour aider les architectes lors de la conception des systèmes temps réel. Certains langages proposent des artefacts pour couvrir l'évolution de la conception lors des différentes phases tandis que d'autres se focalisent sur une phase précise. Nous présentons par la suite quelques langages de modélisation.

2.5.1 AADL

L'*Architecture Analysis and Design Language* (AADL) [AAD19] est un langage de description d'architecture standardisé par SAE (*Society of Automotive Engineers*) en 2004. AADL est un langage dérivé de MetaH [VK00], un langage de description d'architecture créé par Honeywell. AADL fournit des éléments permettant de décrire à la fois la partie logicielle et la partie matérielle d'un système. L'utilisation de AADL permet de décrire finement l'architecture interne

d'un système temps réel ainsi que l'architecture des liens entre le système et les autres périphériques (capteurs et actionneurs). Autrement dit, les artefacts de AADL permettent de décrire un système en phase de déploiement.

2.5.2 MARTE

En 2009, l'OMG a adopté le langage *Modeling and Analysis of Real-Time and Embedded system* (MARTE) [OMGa]. MARTE est un profil UML prenant en charge les étapes de spécification, de conception et de validation. Il remplace et étend l'ancien profil UML pour *Schedulability, Performance and Time* (SPT [G+02]). Les artefacts présents dans MARTE permettent de modéliser différentes vues d'un système embarqué temps réel en plus de la vue vérification temporelle. Les concepts de MARTE permettent la spécification des aspects logiciels (fonctions, tâches, ressources, messages, RTOS), matériels (processors, réseaux, mémoires), et des propriétés non-fonctionnelles (temps, sécurité, énérgie).

2.5.3 MoSaRT

MoSaRT (*Modeling oriented Scheduling analysis of Real-Time systems*) [Ouh13] est un framework à base de modèles permettant la conception et l'analyse d'ordonnaçabilité des systèmes temps réel. L'objectif de MoSaRT est de combler le fossé sémantique existant entre les analyses d'ordonnancement proposées dans le domaine académique et leurs applications dans l'industrie.

MoSaRT se compose de deux parties. La partie *front-end* de *MoSaRT* est un DSML de modélisation graphique orienté analyse temporelle. Autrement dit, contrairement aux autres langages de modélisation qui sont plus orientés vers des modèles métiers, MoSaRT est dédié à l'expression des modèles de tâches comme ceux évoqués dans la section 2.3.2 de ce chapitre.

Afin de capitaliser les efforts d'analyse temporelle, la partie *back-end* de MoSaRT est basé sur un modèle de référentiel d'analyses dédié aux analystes experts et aux chercheurs afin de partager leur savoir-faire et leur expertise d'une manière collaborative.

2.5.4 Time4Sys

Time4Sys [1] est un langage de modélisation graphique pivot orienté analyse. Time4Sys est développé par le consortium WARUNA comprenant Artal, Clearsy, Inria, LIAS/ENSMA, Real-Time-at-Work et Thales Research & Technology. Time4Sys hérite une partie du standard MARTE et a été enrichi par des concepts de MoSaRT (comme les relations de précédence) permettant ainsi de représenter une vue synthétique du système qui capture tous les éléments, les données et les propriétés nécessaires pour la vérification temporelle.

2.5.5 Discussion

La figure 2.16 est une extension de la figure 1.3 en ajoutant l'utilisation des langages de modélisation au cours de la conception des systèmes embarqués temps réel. En effet, cette figure représente notre comparaison des langages de modélisation après les avoir utilisé à différentes étapes de conception. On peut remarquer que contrairement à AADL, MARTE supporte toute les étapes de conception. Cependant, pour décrire le modèle d'architecture, AADL est plus précis et concis. Cette figure a été aussi enrichie par l'ajout des modèles de tâche formels coté méthodes d'analyse ainsi que le compromis à faire entre pessimisme et expressivité. Des fois, il s'avère judicieux de prendre en considération ceci au moment de la modélisation. Dans ce contexte, l'utilisation des langages dédiés analyse comme MoSaRT ou Time4Sys pourrait alors être plus adéquate. En outre, préserver au mieux la sémantique de la modélisation et savoir la traduire d'une manière fiable et conservative au moment de la vérification temporelle est primordial et

1. https ://github.com/polarsys/time4sys

nécessite dans connaissances approfondies dans les deux domaines (modélisation et analyse). L'architecte systèmes doit avoir des réponses aux questions suivantes. Quel est le modèle de tâches le plus adapté à ma modélisation ? quels sont le(s) test(s) le(s) plus approprié(s) ? Quelle est la complexité de ce(s) test(s) ?

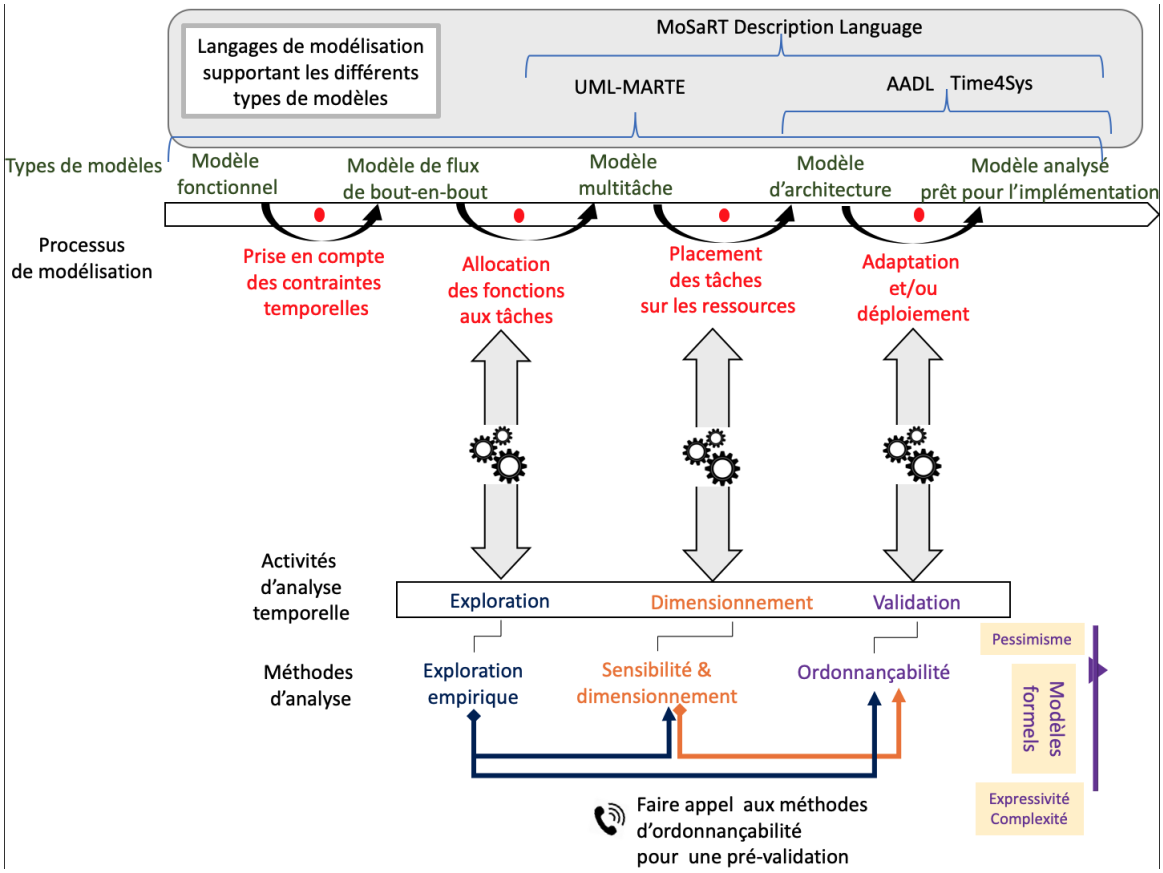

FIGURE 2.16 – Utilisation des langages modélisation dans le cycle de conception

2.6 Conclusion

Dans ce chapitre, nous avons d'abord présenté la structure des systèmes embarqués temps réel. Ensuite, nous avons introduit les caractéristiques des tests d'ordonnançabilité et puis nous avons abordé le paradigme de l'ingénierie dirigée par les modèles et son intégration dans la conception et l'analyse des systèmes temps réel. Enfin, nous avons présenté succinctement quelques langages de conception des systèmes temps réel et discuté leur complémentarité. Dans le chapitre suivant, nous allons nous focaliser sur les tests et présenter des analyses d'ordonnancement relatives aux différentes architectures.

Chapitre 3

Focus sur l'analyse d'ordonnançabilité

Sommaire

Ce chapitre est dédié à la présentation des différentes politiques d'ordonnancement et les tests d'ordonnançabilité associés aux différentes architectures. L'objectif est de mettre en avant l'ensemble d'hypothèses, cas particuliers, usages possibles pour les tests d'ordonnançabilité.

3.1 Introduction

Ce chapitre introduit plus de détails sur les politiques d'ordonnancement ainsi que les tests de validation leur correspondant. L'objectif est de faire ressortir les différentes hypothèses et caractéristiques permettant d'appliquer ou pas certains tests. Par exemple un test, exact dans un contexte, peut être utilisé dans un contexte plus large en perdant son exactitude, mais en restant conservatif (i.e., pire cas). Il convient donc, même lorsqu'on parle de travaux de recherche fondateurs de la validation temporelle, d'être extrêmement attentif au contexte précis dans lequel l'analyse est effectuée. Le reste du chapitre est organisée comme suit. La section 3.2 présente les tests pour les architectures type monoprocesseur pour différents types de tâches et aussi en prenant en considération des facteurs pratiques comme la précédence. La section 3.3 présente les tests pour les architectures type multiprocesseur. La section 3.4 présente les réseaux distribués avec les tests associés. Nous présentons également dans ce chapitre quelques outils d'analyse académiques qui ont implémenté certains tests dans la section 3.5. La section 3.6 présente une discussion relevant certaines problématiques. Enfin, la dernière section conclut ce chapitre.

3.2 Ordonnancement monoprocesseur

Dans cette section, nous allons introduire les politiques d'ordonnancement dédiées aux deux classes de systèmes monoprocesseurs : la classe des tâches indépendantes et la classe des tâches dépendantes. Dans la classe des tâches dépendantes, les tâches peuvent partager des ressources critiques (en exclusion mutuelle) ou être soumises à des contraintes de précédence.

3.2.1 Ordonnancement des tâches indépendantes

3.2.1.1 Ordonnancement à priorité fixe aux tâches

Les politiques d'ordonnancement à priorités fixes aux tâches sont les plus répandues dans l'industrie.

***Rate Monotonic* (RM) [LL73b]** Dans un contexte d'architecture monoprocesseur et où les tâches sont périodiques, préemptibles, synchrones ou sporadiques, indépendantes et à échéance sur requête, la politique *Rate Monotonic* est optimale dans la classe des algorithmes à priorités fixes aux tâches. La politique consiste à affecter la priorité en fonction de la période : plus la période est petite, plus la tâche est prioritaire.

L'équation 3.1 représente une condition nécessaire d'ordonnançabilité à base de l'utilisation processeur dans ce contexte :

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{3.1}$$

Si U dépasse 1, le système est trivialement non-ordonnançable.

La simulation peut être utilisée pour valider un tel système, car celle-ci est T-viable, et C-viable. Le fait que le système soit C-viable signifie que si les dates de réveil sont connues, considérer une pire durée d'exécution des tâches est bien un pire cas de ce que l'on construit en simulant le système. Cela signifie qu'en réduisant une durée d'exécution, aucun temps de réponse ne peut être rallongé dans la simulation. La T-viabilité est importante si certaines tâches du système sont sporadiques. la T-viabilité signifie que si l'on allonge une période, on ne peut pas empirer le pire temps de réponse d'une tâche dans une simulation. Bien entendu, cela n'est vrai que si l'on simule à l'instant critique. La simulation permet à un coût exponentiel de valider le système. L'intervalle de simulation est $[0, PPCM(T_i)]$ pour un système de tâches synchrones. Pour un système de tâches différées, le cycle est obtenu dans la première fenêtre de taille $PPCM(T_i)$ contenant exactement le nombre de temps creux attendu, cette fenêtre démarrant au plus tard à la date $max(r_i) + PPCM(T_i)$.

Liu&Layland [LL73b] ont proposé une condition suffisante à coût linéaire (équation 3.2) basée sur la charge processeur :

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \tag{3.2}$$

Bini et al [BBB03] ont proposé une borne hyperbolique (équation 3.3) dérivée du même scénario du pire cas identifié par Liu&Layland [LL73b] pour des systèmes de tâches périodiques. La borne est démontrée être strictement meilleure (i.e., moins pessimiste) que celle de Liu&Layland. C'est une condition suffisante à coût linéaire.

$$\prod_{i=1}^{n} (\frac{C_i}{T_i} + 1) \leq 2 \tag{3.3}$$

Joseph et Pandya ont proposé la *Response Time Analysis* (RTA) [JP86b], un test exact à coût pseudo-polynomial calculant le pire temps de réponse des tâches. Ce test, et ses variantes, est le plus répandu dans les outils implémentés pour l'analyse temporelle. Si le pire temps de réponse de chaque tâche est inférieur ou égal à son échéance relative, le système est alors ordonnançable. Dans le contexte de système de tâches différées, le test devient suffisant mais pas nécessaire. Le pire temps de réponse du premier travail de la tâche τ_i est le plus petit point fixe obtenu par la suite décrite par l'équation 3.4 ($hp(i)$ est l'ensemble de tâches plus prioritaires que la tâche i) :

$$\begin{aligned} R_i^{(0)} &= C_i \\ R_i^{(n+1)} &= C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil C_j \end{aligned} \tag{3.4}$$

Le test de Joseph et Pandya se limite aux systèmes de tâches contraintes ou implicites. Dans le cas d'échéance arbitraire, une instance peut interférer avec les instances suivantes. Le test a été ensuite étendu par Lehoczky [Leh90] pour supporter ce problème. La date de fin la plus tardive de chaque instance est calculée via la formule 3.5 (i.e., le point fixe obtenu $R_i^{(*)}(k)$ représente la date de fin de $k^{ième}$ instance de la tâche τ_i). Le pire temps de réponse de chaque instance ($TR_i(k)$) est donné par la formule : $TR_i(k) = R_i^{(*)}(k) - (k-1)T_i$. Le pire temps de réponse de tâche est le pire temps de réponse parmi les instances d'une même tâche. On applique cette formule de façon suivante : on commence par $k = 1$, si $R_i^{(*)}(1) > T_i$, la deuxième instance fait partie de la première période d'activité donc il faut tester avec $k = 2$, si $R_i^{(*)}(2) > 2T_i$, il faut tester avec $k = 3$ et ainsi de suite. Tant que $R_i^{(*)}(k) > kT_i$, il faudra tester avec $k + 1$. Le processus converge si et seulement si $U < 1$.

$$\begin{aligned} R_i^{(0)}(k) &= kC_i \\ R_i^{(n+1)}(k) &= kC_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(n)}(k)}{T_j} \right\rceil C_j \end{aligned} \tag{3.5}$$

Sjödin et Hansson [SH98] ont proposé une borne supérieure de RTA (voir l'équation 3.6) calculable en temps polynomial. Si toutes les bornes supérieures des tâches sont inférieures ou égales aux délais, le système de tâches est ordonnançable. Le test est donc une condition suffisante.

$$R_i \leq \frac{\sum_{j \in hp(i) \cup i} C_j}{1 - \sum_{j \in hp(i)} \frac{C_i}{T_i}} \tag{3.6}$$

Deadline Monotonic (DM) [LW82] Dans le contexte d'architecture monoprocesseur et où les tâches sont préemptives, périodiques et synchrones ou sporadiques (i.e., il existe ou peut exister un instant critique), indépendantes et à échéances contraintes, *Deadline Monotonic* est la politique optimale dans la classe des algorithmes à priorité fixe. La politique consiste à affecter la priorité inversement proportionnelle à l'échéance relative : plus l'échéance est petite plus la tâche est prioritaire. *Rate Monotonic* est le cas particulier de *Deadline Monotonic* pour les systèmes de tâches à échéance sur requête et *Deadline Monotonic* est dominant par rapport à *Rate Monotonic*.

La simulation et le test RTA restent valables pour *Deadline Monotonic*. Cependant, la condition suffisante à base de la charge processeur devient comme suit (voir l'équation 3.7), ce qui induit un pessimisme encore plus grand.

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{\frac{1}{n}} - 1) \tag{3.7}$$

Optimal Priority Assignment (OPA) [Aud91] L'affection de priorités d'Audsley est à la fois une politique d'ordonnancement et une condition exacte d'ordonnançabilité. *OPA* consiste à affecter les priorités aux tâches dans l'ordre suivant : "de priorité faible à priorité forte". Elle repose sur trois hypothèses que le contexte doit satisfaire pour que la priorité d'une tâche puisse être affectée indépendamment des priorités plus élevées :

— l'ordonnançabilité d'une tâche peut dépendre des tâches plus prioritaires, individuellement, mais pas de leur ordre de priorités entre elles ;

— l'ordonnançabilité d'une tâche peut dépendre des tâches moins prioritaires, individuellement, mais pas de leur ordre de priorités entre elles ;

— si une tâche respecte ses échéances avec un niveau de priorité donné, elle ne peut pas ne pas les respecter avec un niveau de priorité plus élevé.

Ainsi, dans le contexte de tâches indépendantes, comme le temps de réponse d'une tâche ne dépend pas de tâches moins prioritaires ni de l'ordre des tâches plus prioritaires, on peut appliquer *OPA*. Il y a au plus $\frac{n(n+1)}{2}$ affectations à tester (n est le nombre de tâches). A chaque affectation de priorité, il faut tester l'ordonnançabilité de la tâche sous la priorité affectée. Dans les contextes de tâches indépendantes où ni DM ni RM ne sont optimaux (typiquement absence d'instant critique), un test exact est forcément basé sur la simulation ou équivalent. En effet, en l'absence d'instant critique, il est nécessaire de construire toutes les périodes d'activité, l'instant critique ne permettant pas de dégager immédiatement l'emplacement de la plus longue période d'activité, qui contient le pire temps de réponse. Il est assez simple dans ce contexte de généraliser les résultats de Leung et Whitehead [LW82] qui ont montré que l'ordonnançabilité était un problème co-NP-difficile au sens fort dès lors qu'il y avait absence d'instant critique. Une fois que toutes les tâches ont leur ordre de priorité, le système de tâches est ordonnançable par construction. *OPA* est optimal dans les ordonnancements à priorités fixes aux tâches dans toutes les classes de systèmes qui respectent les trois hypothèses nécessaires. *OPA* est donc dominant par rapport aux politiques d'ordonnancement *Deadline Monotonic* et *Rate Monotonic*, mis à part dans les contextes où ces algorithmes sont optimaux, dans lesquels bien entendu, ils sont équivalents.

3.2.1.2 Ordonnancement à priorités fixes aux travaux

Earliest Deadline First (EDF) [Hor74, Der74] EDF fait une timide apparition dans l'industrie depuis son apparition dans les normes POSIX et Ada. Il est cependant très peu utilisé. Cette politique consiste à affecter les priorités aux instances des tâches en fonction des délais absolus : plus le délai absolu d'une instance est proche, plus l'instance est prioritaire. La priorité

est affectée au réveil d'une instance et ne changera pas par rapport aux autres instances actives, EDF est donc à priorités fixes aux travaux. bien entendu, de nouvelles instances peuvent se réveiller et s'intercaler entre les priorités des instances présentes, en fonction de leur date d'échéance absolue. Dans le contexte où les tâches sont préemptibles et indépendantes, un système est faisable si et seulement s'il est ordonnançable par *EDF*. Pour les systèmes de tâches indépendantes, à échéances sur requête, le test d'utilisation du processeur [LL73b] (équation 3.1) est non seulement nécessaire mais aussi suffisant. Pour les systèmes de tâches à échéances arbitraires, une variante du test à base de l'utilisation du processeur (équation 3.8) est une condition suffisante :

$$\sum_{i=1}^{n} \frac{C_i}{min(D_i, T_i)} \leq 1 \tag{3.8}$$

La simulation est une condition exacte pour *EDF* dès lors que les tâches sont indépendantes (C-viabilité, T-viabilité). Une autre condition exacte pour les systèmes de tâches à échéances contraintes repose sur la notion de *Demand Bound Function* (DBF) représentant la demande cumulée des tâches réveillées et terminées dans un intervalle de temps. La DBF est un test d'ordonnançabilité exact pour les algorithmes d'ordonnancement optimaux, par conséquent, ce test d'ordonnançabilité s'applique dans les contextes où *EDF* est optimal. La DBF d'un intervalle ne doit pas dépasser la largeur de l'intervalle pour que le système soit faisable, celle-ci représente la quantité de travail que le système devra avoir consacré à une tâche au moment de son échéance. Cela mène à l'équation 3.9 [PL05] :

$$\begin{aligned} D &= \{d_{i,k} \mid d_{i,k} = kT_i + D_i, d_{i,k} \leq \frac{U}{1-U} max_i(T_i - D_i)\} \\ dbf_i(t) &= \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \\ \forall L \in D, dbf(L) &= \sum_{i=1}^{n} dbf_i(L) \leq L \end{aligned} \tag{3.9}$$

La complexité de ce test est pseudo-polynomiale.

3.2.1.3 Ordonnancement à priorités dynamiques

Ce type d'ordonnancement est purement académique, et n'est pas, ou quasiment pas, utilisé dans l'industrie.

Least Laxity First (LLF) [Mok83] Cette politique consiste à affecter les priorités d'une manière inversement proportionnelle à la laxité : plus la laxité est faible plus l'instance est prioritaire. Comme EDF, dans le contexte des tâches préemptibles, le système de tâches est faisable si et seulement s'il est ordonnançable par LLF. Les tests basés sur la charge processeur restent valables pour LLF. La puissance d'ordonnançabilité de LLF est équivalent à EDF dans le contexte monoprocesseur. On peut donc aussi lui appliquer le test d'ordonnançabilité basé sur la DBF.

3.2.2 Ordonnancement des tâches dépendantes

3.2.2.1 Contraintes de précédence

Des contraintes de précédence peuvent être mises en place pour gérer des communications ou des synchronisations liées à l'accès aux capteurs et actionneurs afin de garantir certaines contraintes comme la fraîcheur des données ou les délais d'exécution de bout en bout. Une tâche réceptrice en cours d'exécution, à un point de communication, se met en attente de la

satisfaction d'une condition. Autrement dit, l'exécution de la partie située après le point de communication de la tâche réceptrice doit être précédée par l'exécution de la tâche émettrice, ce qui génère des contraintes de précédence entre certaines parties des tâches. En général, les tâches communicantes sont découpées autour des points de synchronisation afin de transformer le modèle en tâches recevant leurs messages au début d'exécution et les envoyant à la fin. Par conséquent, les tests d'ordonnançabilité des tâches indépendantes peuvent être utilisés. En effet, les caractéristiques temporelles (comme la date de réveil r_i et la date d'échéance D_i) des tâches obtenues, sont modifiées en garantissant une affectation des priorités par les algorithmes classiques assurant que la tâche précédée par une autre aura une priorité inférieure à la tâche qui la précède.

3.2.2.2 Ordonnancement des tâches avec ressources partagées

Lorsque plusieurs tâches partagent une ressource critique, il faut tenir compte du problème d'exclusion mutuelle. La ressource critique ne peut pas être accédée en même temps par plusieurs tâches ou autrement dit une tâche en section critique ne peut pas être préemptée par d'autres tâches qui partagent la même ressource. L'ordonnancement des tâches partageant les ressources mène vers le problème d'**inversion de priorité** et d'**interblocage**. L'inversion de priorités [SRL90] survient lorsqu'une tâche est bloquée par une tâche moins prioritaire qui ne partage pas la même ressource. Un ensemble de tâches est en interblocage si chaque tâche de l'ensemble est en attente d'un événement qui peut être produit par une autre tâche de l'ensemble en attente [Hav68]. Plusieurs protocoles de gestion de ressources partagées sont proposés pour résoudre les problèmes : protocole à priorités héritées, protocole à priorité plafond et protocole d'allocation de la pile.

Un système de tâches sous le protocole à priorités héritées ou protocole à priorité plafond peut être analysé par une modification conservative du calcul de temps de réponse présenté par l'équation 3.10.

$$\begin{aligned} R_i^{(0)} &= B_i + C_i \\ R_i^{(n+1)} &= B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil C_j \end{aligned} \tag{3.10}$$

B_i est le temps de blocage maximal qu'une tâche peut subir de la part de tâches de priorité inférieure due à l'héritage de priorité trouvé dans les protocoles de gestion de ressource. Le calcul du facteur de blocage B_i dépend du protocole. Dans tous les cas, le calcul de temps de réponse devient dans ce cas conservatif, puisqu'il considère un scénario qui considère qu'à l'instant critique, la section critique la plus longue, ou l'enchaînement de sections critiques les plus longues, vient de démarrer, ce qui peut ne jamais arriver dans les faits.

Le protocole à priorités héritées (*Priority Inheritance Protocol* - PIP) proposé par Sha et al. [SRL90] pour les politiques à priorité fixe aux tâches. Lorsqu'une tâche τ_j bloque une autre tâche plus prioritaire τ_i, τ_j héritera la priorité de τ_i pendant toute la durée de son exécution. Après son exécution, τ_j retombe à la priorité initiale. Ce protocole permet de réduire le temps de blocage mais ne peut pas supprimer l'interblocage. Le temps de blocage B_i selon le protocole est calculé par l'équation 3.11. $lp(i)$ représente l'ensemble de tâches moins prioritaires que la tâche i, $duréeSC_j(R)$ représente la durée de la section critique de la tâche τ_j accédant à la ressource R. A cause de sa complexité d'implémentation et de ses performances moins bonnes que le PCP (présenté ultérieurement), ce protocole est peu utilisé. Nous le retrouvons cependant dans le RTOS VxWorks.

$$B_i = \sum_{\forall ressource\ R\ utilisée\ par\ une\ tâche \in hp(i) \cup \{i\}} max_{\forall j \in lp(i)}(duréeSC_j(R)) \tag{3.11}$$

Le protocole à priorité plafond (*Priority Ceiling Protocol* - PCP) [SRL90] est une extension du PIP permettant d'éviter l'interblocage et les attentes de plusieurs ressources. Comme PIP, ce protocole est adapté aux politiques d'ordonnancement à priorités fixes aux tâches. Chaque ressource se voit affecter une **priorité plafond**. Cette priorité plafond est définie par la plus grande priorité des tâches qui accèdent à cette ressource. A chaque instant t, la **priorité plafond du système** est définie par la plus grande priorité plafond parmi les ressources en cours d'utilisation à l'instant t. Une tâche ne pourra exécuter sa section critique que si elle est strictement plus prioritaire que la priorité plafond système ou qu'elle en est elle-même détentrice. Lorsqu'une tâche τ_i est bloquée pour accéder à une ressource R, la tâche τ_j détenant R hérite de la priorité de τ_i pendant la section critique sur R. Le temps de blocage selon ce protocole est calculé selon l'équation 3.12.

$$B_i = \max_{\forall ressource\ R\ utilisée\ par\ une\ tâche\ de\ hp(i)\cup\{i\},\ \forall j \in lp(i)} (duréeSC_j(R)) \tag{3.12}$$

Ce protocole n'est en fait jamais implémenté tel quel dans les RTOS. On l'y retrouve sous sa version immédiate (la priorité est héritée dès l'entrée en section critique), qui permet de ne pas utiliser la notion de plafond système. Elle correspond au protocole à super priorité proposé dans [K+82], et possède les mêmes propriétés pire cas (facteur de blocage) que le protocole dans sa version académique.

Le protocole d'allocation de la pile (*Stack Resource Protocol* - SRP) a été proposé par Baker dans [Bak91]. SRP est une adaptation du protocole PCP pour la politique EDF. Ce protocole présente deux nouvelles notions : **le niveau de préemption** et **le plafond de préemption** (*preemption ceiling*). Chaque tâche τ_i est affectée hors-ligne un niveau de préemption π_i indépendamment de l'algorithme d'ordonnancement utilisé. Chaque ressource R a un plafond de préemption C_R qui est la valeur maximale des niveaux de préemption des tâches pouvant bloquer sur R. Le **plafond du système** est la valeur maximale des plafonds de préemption des ressources.

Une tâche τ_j ne peut préempter une tâche τ_i que si :

— La priorité de τ_j est supérieure à la priorité de τ_i.

— Le niveau de préemption de τ_j est supérieur au niveau de préemption de τ_i ($\pi_j > \pi_i$).

— Le niveau de préemption de τ_j est supérieur au plafond du système.

L'utilisation de SRP assure que les tâches ne pourront pas démarrer leurs exécutions si les ressources nécessaires ne seront pas disponibles.

La formule du temps de blocage reste la même que celle du protocole PCP (l'équation 3.12) en remplaçant la priorité par le niveau de préemption et la priorité plafond par le plafond de préemption. Une condition suffisante pour le système de tâches sous EDF est présentée par la formule 3.13 :

$$\forall i, 1 \leq i \leq n : \sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{B_i}{T_i} \leq 1 \tag{3.13}$$

3.3 Ordonnancement multi-processeur

L'ordonnancement multiprocesseur fait apparaître la notion de **migration** - la possibilité de changement de processeur des tâches en cours d'exécution. En fonction du niveau de migration inter-processeur autorisé, nous considérons les trois catégories suivantes [CFH+04] :

— Pas de migration (partitionné) : chaque tâche est affectée à un processeur unique et ne change pas pendant l'exécution. Cela ne permet pas dans le cas général une utilisation totale de la plateforme.

— Migration restreinte (migration de tâches) : chaque instance de tâches doit s'exécuter entièrement sur un processeur. Pourtant, les différents instances d'une tâche peuvent s'exécuter sur différents processeurs.

— Migration libre (global) : les instances peuvent changer de processeur pendant l'exécution. Cependant, une instance ne peut pas s'exécuter parallèlement sur différents processeurs. Ce type d'hypothèse permet une utilisation totale de la plateforme.

— Semi-partitionné : la plupart des tâches sont partitionnées sur un cœur, certaines sont autorisées à migrer, ce qui permet une utilisation totale de la plateforme à partir du moment où au moins m tâches sont autorisées à migrer.

Les politiques d'ordonnancement dans lesquelles aucune migration n'est autorisée sont qualifiées comme **des techniques partitionnées**. Celles où la migration est autorisée sont qualifiées comme **des techniques globales**.

3.3.1 Ordonnancement partitionné

Dans le contexte des processeurs identiques, le principe de l'ordonnancement par partitionnement est relativement simple. Les techniques consistent à partitionner hors-ligne n tâches sur m processeurs pour que tous les sous-ensembles de tâches partitionnées soient ordonnançables en monoprocesseur. Les tâches affectées à un même processeur sont ordonnancées indépendamment selon une politique d'ordonnancement monoprocesseur. Le problème d'ordonnancement multiprocesseur est réduit au problème d'ordonnancement monoprocesseur après le partitionnement de tâches. Le partitionnement se base sur les tests d'ordonnançabilité, les caractéristiques des tâches et notamment leur facteur d'utilisation. Ce problème de partitionnement optimal est semblable au problème de *bin-packing* [Joh74] qui consiste à chercher le rangement le plus économique possible pour un ensemble d'articles de tailles différentes dans des boîtes de tailles limitées. La solution exacte pour le problème existe mais ne convient qu'à un nombre faible de tâches [Kor03, Sha04]. Une autre méthode est d'utiliser des heuristiques. Les heuristiques de base sont *First-Fit* [FBB06], *Best-Fit* [OS95], *Next-Fit* [AJ01] et *Worst-Fit* [mJGJ96].

L'algorithme *First-Fit* consiste à affecter une tâche au premier processeur trouvé tel que la tâche est ordonnançable sur ce processeur. La recherche de processeur se répète depuis le début de la liste des processeurs pour chaque tâche. Le principe de l'algorithme *Next-Fit* est similaire au *First-Fit*. La différence est que l'on ne répète pas la recherche de processeur depuis le début pour une nouvelle tâche mais depuis le processeur auquel on a affecté la tâche précédente. L'algorithme *Best-Fit* consiste à choisir le processeur qui a la plus petite capacité disponible et capable d'accepter la tâche tout en restant ordonnançable. Contrairement à *Best-Fit*, l'algorithme *Worst-Fit* choisit le processeur disposant de la plus grande capacité disponible et capable d'accepter la tâche tout en restant ordonnançable, ce qui a pour effet d'équilibrer la charge.

3.3.2 Ordonnancement global

Dans l'ordonnancement global, toutes les tâches prêtes sont stockées dans une file ordonnée par priorité. L'ordonnanceur **unique** global choisit les tâches les plus prioritaires et affecte aux processeurs disponibles sinon il préempte les tâches moins prioritaires en cours d'exécution.

La transposition des politiques d'ordonnancement classiques au cas multiprocesseur telles que G-RM, G-EDF (i.e., *Global-Rate Monotonic*, *Global-Earliest Deadline First*) ne permet pas de garder leur optimalité. De nombreuses conditions suffisantes d'ordonnançabilité [ABJ01, BG03, BCL05, SB02, GFB03] sont disponibles et représentées dans le tableau 3.1. Elles reposent à la fois sur $u_{max} = max_{i=1}^{n} \frac{C_i}{T_i}$, $u_{sum} = \sum_{i=1}^{n} \frac{C_i}{T_i}$, la présence d'une tâche lourde (dont l'utilisation processeur est élevée) peut empêcher de conclure sur l'ordonnançabilité d'un système de tâches.

Les politiques *Utilisation threShold* - US telles que RM-US[ζ] [ABJ01] et EDF-US[ζ] [SB02] dérivent des algorithmes G-RM et G-EDF en ajoutant un seuil ζ sur l'utilisation de processeur de

TABLEAU 3.1 – Tests d'ordonnançabilité pour *G-RM* et *G-EDF*.

G-RM	**G-EDF**
$u_{max} \leq \frac{m}{3m-2}$ et $u_{sum} \leq \frac{m^2}{3m-2}$	$u_{max} \leq \frac{m}{2m-1}$ et $u_{sum} \leq \frac{m^2}{2m-1}$
$u_{max} \leq \frac{1}{3}$ et $u_{sum} \leq \frac{m}{3}$	$u_{max} \leq \frac{1}{2}$ et $u_{sum} \leq \frac{m+1}{2}$
$u_{sum} \leq \frac{m}{2}(1 - u_{max}) + u_{max}$	$u_{sum} \leq m - (m-1)u_{max}$

chaque tâche. Au delà de ce seuil (i.e., $u_i > \zeta$), les tâches sont qualifiées de **lourdes** et se voient affecter une priorité maximale. Autrement, les tâches sont qualifiées de **légères** et ordonnancées selon G-RM ou G-EDF. Les tests d'ordonnançabilité ne sont pas définis de manière générale mais pour quelques valeurs particulières de ζ (représentées dans le tableau 3.2).

TABLEAU 3.2 – Tests d'ordonnançabilité pour RM-US[ζ] et EDF-US[ζ].

RM-US[ζ]		**EDF-US**[ζ]	
$\zeta = \frac{m}{3m-2}$	$u_{sum} \leq \frac{m^2}{3m-2}$	$\zeta = \frac{m}{2m-1}$	$u_{sum} \leq \frac{m^2}{2m-1}$
$\zeta = \frac{1}{3}$	$u_{sum} \leq \frac{m+1}{3}$	$\zeta = \frac{1}{2}$	$u_{sum} \leq \frac{m+1}{2}$

La politique EDF$^{(k)}$ étend G-EDF. Sous EDF$^{(k)}$, les tâches sont ordonnées selon l'utilisation décroissante de processeur (i.e, $u_i \geq u_{i+1} \forall i \in [1, n-1]$). Les priorités sont affectées en fonction de k. Si $i < k$ la priorité maximale est affectée aux tâches, autrement les tâches auront une priorité selon la politique originale. La condition suffisante d'ordonnançabilité du système de tâches sous EDF$^{(k)}$ [GFB03] est représentée par la formule 3.14.

$$m \geq (k-1) + \left\lceil \frac{\sum_{i=k+1}^{n} u_i}{1 - u_k} \right\rceil \tag{3.14}$$

Les extensions *Zero Laxity* - ZL telles que EDZL (*Earliest Deadline first until Zero Laxity*) [Lee94], RMZL (*Rate Monotonic until Zero Laxity*), FPZL (*Fixed Priority until Zero Laxity*) [DB11a] affectent la priorité maximale aux tâches quand leur laxité atteint zéro, autrement les tâches sont affectées de priorités selon les politiques originales. Les extensions ZL sont dominantes par rapport à leur politique originale [PHK$^+$05].

Les politiques de la famille *Proportionate Fair* - PFair (ou **équitable**) [BCPV96] ont une approche différente. Les politiques *PFair* imposent l'exécution des tâches à un taux régulier. L'objectif de *PFair* est d'allouer les processeurs aux tâches de manière proportionnelle par rapport à leur facteur d'utilisation. Soit $lag(\tau_i, t)$ la fonction de l'écart entre l'ordonnancement idéal et l'ordonnancement réel à l'instant t : $lag(\tau_i, t) = t.u_i - \sum_{l=1}^{t} S(\tau_i, l)$, avec $S(\tau_i, l)$ est la fonction caractéristique de l'ordonnanceur (1 si τ_i est exécutée sur $[t-1, t)$ et 0 sinon). Une politique d'ordonnancement est dite **PFair** (ou **équitable**) lorsque $\forall t \in \mathbb{N}^*, \forall \tau_i \in \tau, |lag(\tau_i, t)| < 1$. Les politiques d'ordonnancement *PFair* sont optimales pour les systèmes de tâches périodiques, synchrones et à échéance sur requête. Il existe différentes politiques de type *PFair* telles que PF, PD [BCPV96] et PD^2 [AS99]. Pour les tâches à échéance sur requête et périodiques, les politiques PFair permettent la totale utilisation de la plateforme puisque la condition nécessaire et suffisante d'ordonnançabilité est 3.15 [BCPV96] :

$$u_{sum} \leq m \tag{3.15}$$

3.3.3 Ordonnancement semi-partitionné

L'ordonnancement semi-partitionné est considéré comme une amélioration de l'ordonnancement par partitionnement en permettant une utilisation de processeur plus élevée. Il offre aussi moins de migrations et de préemptions par rapport à l'ordonnancement global.

La première politique semi-partitionnée proposée est EDF-fm (fm indique chaque tâche est soit fixe ou migrante) [ABD05]. EDF-fm affecte la priorité maximale aux tâches migrantes de manière statique. Les tâches fixes sont partitionnées aux processeurs à l'aide d'une variante de l'algorithme *Next-Fit*, ensuite ordonnancées selon EDF quand aucune tâche migrante n'est prête à s'exécuter. Cependant, EDF-fm est conçu pour les systèmes temps réel mous, l'ordonnançabilité des systèmes sous cette politique n'est donc pas garantie.

L'algorithme ***EDF with task splitting and K processors in a Group*** - EKG [AT06] propose de partager l'ensemble de m processeurs en groupes de taille k et ne permet la migration que parmi les processeurs du même groupe. De plus, une tâche migre au plus vers deux processeurs. EKG classifie les tâches en deux types basées sur l'utilisation de processeur : tâche lourde et tâche légère. Si l'utilisation de processeur d'une tâche est supérieure ou égale à la valeur de **SEP** alors la tâche est qualifiée de **lourde** et sinon elle est **légère** (SEP=$\frac{k}{k+1}$ si $k < m$ ou SEP=1 si $k = m$). Le principe de partitionnement est inspiré de l'algorithme *Next-Fit*. Premièrement, les tâches lourdes sont allouées à un processeur chacune, suivi par l'allocation des tâches légères. Quand le processeur n'a plus assez de capacité pour accepter une tâche, la tâche sera découpée et repartie parmi les processeurs dans un même groupe. Les instances des tâches migrantes se voient réserver du temps proportionnel à leur utilisation de processeur et leur période. Soit τ_i la tâche migrante allouée à deux processeurs, la portion τ_i' (dont l'utilisation de processeur $U(\tau_i')$) s'exécute sur un processeur et la portion τ_i'' (dont l'utilisation de processeur $U(\tau_i'')$) continue sur l'autre. Soient t_0, t_1 les dates de réveil successives de la tâche τ_i. Les portions sont réparties aux deux bouts de l'intervalle $[t_0, t_1]$. La portion τ_i' commence à l'instant t_0 et est réservée pendant un intervalle $U(\tau_i') \times (t_1 - t_0)$ et se termine à l'instant t_a, la portion τ_i'' commence à l'instant t_b, se termine à l'instant t_1 et est réservée sur un intervalle $U(\tau_i'') \times (t_1 - t_0)$ (illustrées par la figure 3.1). Ceci se base sur le même principe que la technique d'enveloppement de McNaughton [McN59]. $U(\tau_i) = U(\tau_i') + U(\tau_i'') \leq 1$ donc $t_a \leq t_b$, or les deux portions de la tâche τ_i s'exécutent sur des processeurs sans chevauchement. Les autres tâches sont ordonnancées selon EDF. L'équation 3.16

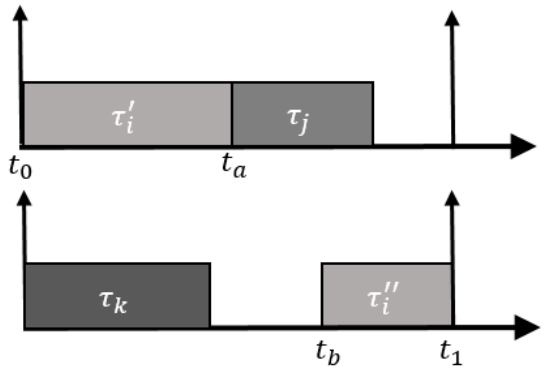

FIGURE 3.1 – Ordonnancement par EKG

représente une condition suffisante pour les systèmes de tâches sous *EKG*.

$$\frac{1}{m} \sum_{i=1}^{n} \frac{C_i}{T_i} \leq SEP \tag{3.16}$$

L'algorithme EKG est optimal pour les systèmes de tâches périodiques à échéance sur requête et pour $k = m$. Lorsque $k < m$ l'algorithme n'est plus optimal mais génère moins de préemptions.

EKG est généralisé aux tâches sporadiques dans [AB08] et aux tâches à échéances arbitraires dans [ABB08].

Contrairement à l'algorithme EKG, l'algorithme *Earliest Deadline and Highest-priority Split* (EDHS) [KY08] procède en deux étapes bien séparées. La première étape consiste à répartir les tâches sur les processeurs selon un algorithme de partitionnement. La prochaine étape est d'affecter les tâches non-partitionables aux processeurs selon un deuxième algorithme. Grâce à cette séparation, tous les systèmes de tâches ordonnançables par l'algorithme de partitionnement sont aussi ordonnançables par EDHS. A la deuxième étape, une tâche migrante peut être répartie sur plusieurs processeurs mais chaque processeur ne peut accepter qu'une seule tâche migrante au plus. La politique EDHS est relativement simple :

— Les tâches migrantes ont les priorités maximales globales.

— Toutes les instances d'une tâche migrante commencent sur le premier processeur auquel la tâche est affectée et elles sont migrées et continuent séquentiellement sur le prochain processeur lorsque la portion affectée est toute consommée au processeur courant.

— Les tâches partitionnées sont alors ordonnancées par EDF.

Cela permet d'éviter des chevauchements pendant l'exécution des tâches migrantes.

L'algorithme *EDF with Window-constraint Migration* (EDF-WM) a une approche différente. Comme EDHS, EDF-WM se compose de deux étapes. En première étape, les tâches sont reparties sur les processeurs selon l'algorithme *First-Fit*. Les tâches qui ne sont pas partitionnées sont migrantes. En deuxième étape, les tâches migrantes sont divisées en sous-tâches, il est défini pour chaque sous-tâche un temps d'exécution, une date de pseudo-réveil et une pseudo-échéance. Les sous-tâches sont alors ordonnancées selon EDF comme des tâches classiques.

3.4 Ordonnancement distribué (réparti)

L'architecture des systèmes distribués se compose de noeuds. Les processeurs des noeuds différents communiquent en échangeant des messages sur les réseaux. L'ordonnancement des systèmes distribués doit donc prendre en compte le mécanisme de communication des réseaux sur lesquels les messages transitent. Des réseaux de différentes topologies avec différents protocoles de communication ont été étudiés dans la littérature.

Il existe deux modèles de base représentant les problèmes canoniques : le comportement dirigé par les événements et le comportement dirigé par le temps. Pour le comportement dirigé par les événements, les messages sont transmis à l'occurrence d'un événement reconnu. Le protocole doit garantir l'accès au réseau pour la transmission des données assez rapide pour que les données transmises ne soient pas caduques lors de leur consommation. Cette approche est très efficace en termes d'utilisation de bande passante mais il reste difficile de vérifier que le temps de réponse des messages respecte leur échéance. Pour le comportement dirigé par le temps, les données sont transmises à des instants prédéterminés au sein des **trames** sur le réseau.

Un réseau embarqué temps réel doit être **déterministe**. Un réseau est déterministe s'il peut certifier la non perte de données et garantir une borne supérieure sur le délai de bout en bout de chaque trame traversant le réseau. Il faut donc soit interdire les collisions sur les liens, soit détecter les collisions et ré-émettre des données de façon déterministe.

Dans la suite, nous focalisons sur les réseaux utilisés souvent dans le domaine d'aéronautique : ARINC664 partie 7 (AFDX) et CAN, car ils sont représentatifs, respectivement, des réseaux commutés, à brins dédiés et des bus de communication partagés.

3.4.1 Bus CAN

Le bus CAN (*Controller Area Network*) a été développé par Bosch et Intel pour le domaine de l'automobile. Il est présenté la première fois en 1985 et normalisé sous les normes ISO 11898 et ISO 11519. Il a été dérivé pour d'autres domaines, y compris l'aéronautique. Le bus connecte

par un seul câble (bus) un ensemble d'équipements qui vont communiquer tour à tour. Le bus est appelé **médium**. Le débit maximal du réseau CAN peut atteindre 1 Mbits/s [Nav96], ce qui est assez faible, et surtout dû au fait que chaque bit transmis est visible sur la totalité du réseau dans le même "temps bit". La figure 3.2 présente la topologie d'un réseau CAN.

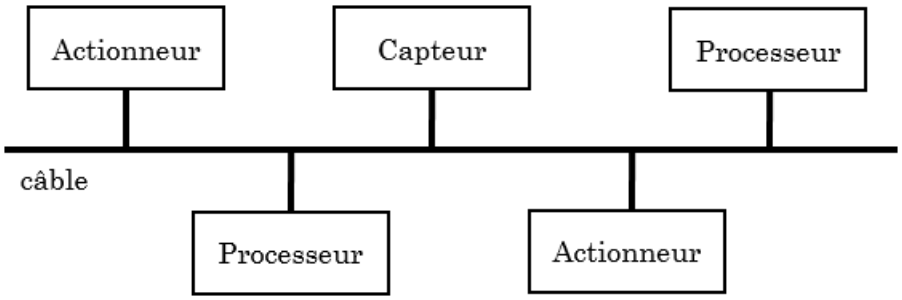

FIGURE 3.2 – Topologie du réseau CAN

Deux protocoles de communication disponibles pour le réseau CAN sont CSMA/CA (*Carrier Sense Multiple Acces with Collision Avoidance*) ou TDMA (*Time Division Multiple Access*). Pour le protocole CSMA/CA, l'accès au médium se fait en fonction de la priorité des trames. La priorité d'une trame CAN se définit par l'identifiant. Plus l'identifiant est petit, plus la trame est prioritaire. Pour le protocole TDMA, chaque médium dispose d'un intervalle de temps pendant lequel il sera le seul à transmettre les données sur le réseau sachant que la transmission est cyclique pour tous les médiums. Dans la suite, on ne considère que le protocole CSMA/CA.

Le pire temps de réponse des tâches et des messages peut être déterminé en utilisant l'**analyse holistique** introduite la première fois en 1994 [TC94, Tin94]. Selon l'analyse holistique, le réseau est considéré comme un processeur ordonnancé de manière non-préemptive [LRS96]. La dépendance de la tâche réceptrice d'un message à la tâche émettrice de ce message est caractérisée par une gigue d'activation. Les trames sont représentées par des tâches ordonnancées de manière non préemptible sur le réseau, en prenant en compte qu'un temps bit sépare la vision que deux nœuds différents ont de l'état du réseau.

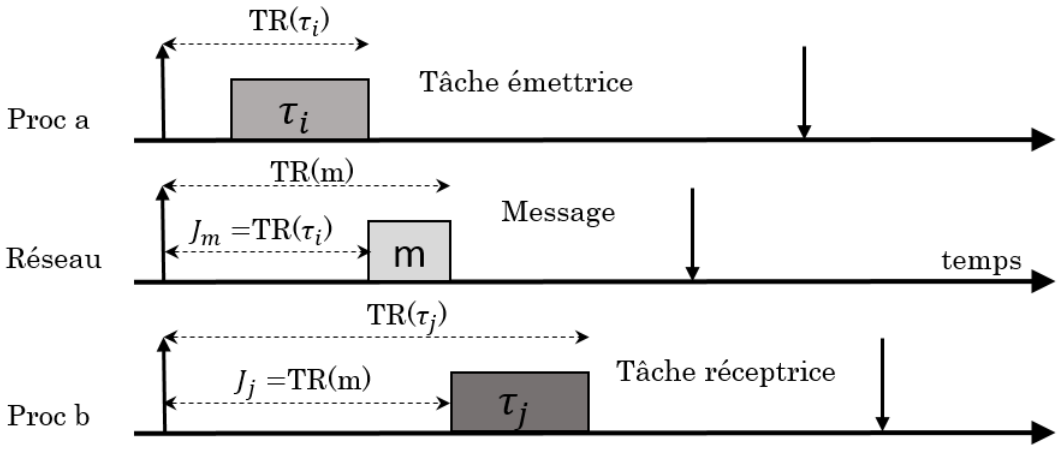

FIGURE 3.3 – Exemple d'analyse holistique pour 2 processeurs

La figure 3.3 présente un simple exemple de deux processeurs communiquant par le réseau CAN. La tâche τ_i sur le processeur a émet le message m à la tâche τ_j sur le processeur b. L'analyse sera effectuée dans l'ordre. Premièrement, le temps de réponse de la tâche émettrice est calculé localement sur le processeur a en utilisant le calcul de temps de réponse [JP86a]. Ensuite, l'analyse de temps de réponse du message non-préemptif m [LRS96] doit prendre en compte le pire temps de réponse de la tâche émettrice sous forme d'une gigue (J_m). Enfin,

l'analyse de temps de réponse de la tâche réceptrice prend en compte le pire temps de réponse du message sous forme d'une gigue (J_j). L'analyse de temps de réponse prenant en compte les gigues dans le cas simple (échéance contrainte et absence de ressource partagée) est présentée dans la formule 3.17.

$$\begin{aligned} R_i^{(0)} &= C_i \\ R_i^{(n+1)} &= C_i + \sum_{k \in hp(i)} \left\lfloor \frac{J_k + R_i^{(n)}}{T_k} \right\rfloor C_k \end{aligned} \tag{3.17}$$

Il faut souligner ici que la seconde tâche est activée par l'arrivée du message (déclenchement basé événements), ce qui n'est pas le cas de toutes les applications communicantes. Ainsi par exemple, dans certains cas, la tâche réceptrice pourrait faire de la scrutation de l'état des nouveaux messages arrivés à chacune de ses exécutions périodiques. Dans l'analyse holistique, le pire temps de réponse calculé pour une tâche soumise à gigue d'activation correspond non pas au temps de réponse entre son activation (par l'arrivée du message) et sa terminaison, mais au pire temps de réponse de la chaîne de précédence entière, de l'activation de la tâche l'initiant, à la terminaison de la tâche étudiée.

3.4.2 Réseau ARINC 664 partie 7

ARINC 664 partie 7 a été développé et normalisée pour le domaine avionique, issu de la technologie Ethernet commuté. ARINC 664 partie 7 est plus connue sous le nom AFDX (***A**vionics **F**ull **D**uple**X** Switched Ethernet*). AFDX est un réseau de topologie maillée. AFDX repose sur des équipements appelés **commutateurs** (*switches*). Les terminaux du réseau appelés les ***End/System*** (E/S) sont les équipements connectés au réseau. La figure 3.4 illustre la topologie d'AFDX avec cinq E/S et trois commutateurs.

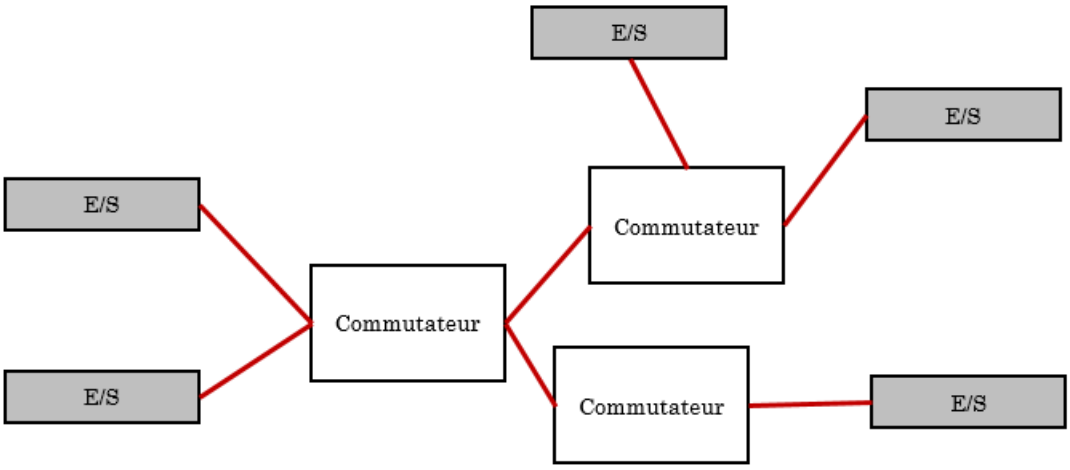

FIGURE 3.4 – Topologie du réseau AFDX

Chaque trame est émise depuis un E/S émetteur et est acheminée vers des E/S destinataires à travers l'intermédiaire de commutateurs. Les E/S sont reliés par les liens physiques (les fils rouges dans la figure 3.4). Les liens sont en mode *Full Duplex*. Les trames peuvent donc être transmises dans les deux sens sans collision possible. Le débit de chaque lien est de 10 ou 100 Mbits/s [Kem14]. L'arbitrage de ressources a donc lieu dans chaque buffer de sortie de chaque switch : par exemple, si deux ports d'entrée acheminent à 100Mbits/s des messages qui doivent être émis sur le même port de sortie, lui aussi à 100Mbits/s, on a 200 Mbits/s de données qui arrivent, mais seulement la moitié qui peut sortir du switch par le port choisi. Les trames sont

transmises sur le réseau à travers des **liens virtuels** (*Vitual Link* - VL). Un VL est un canal virtuel, statique, unidirectionnel et reposant sur des liens physiques, possédant une source et un ensemble de destinations. La figure 3.5 illustre trois VLs reposant sur les liens physiques. Les VLs en orange et en bleu sont *unicast* (un émetteur vers un destinataire). Le VL en rouge est *multicast* (un émetteur vers plusieurs destinataires).

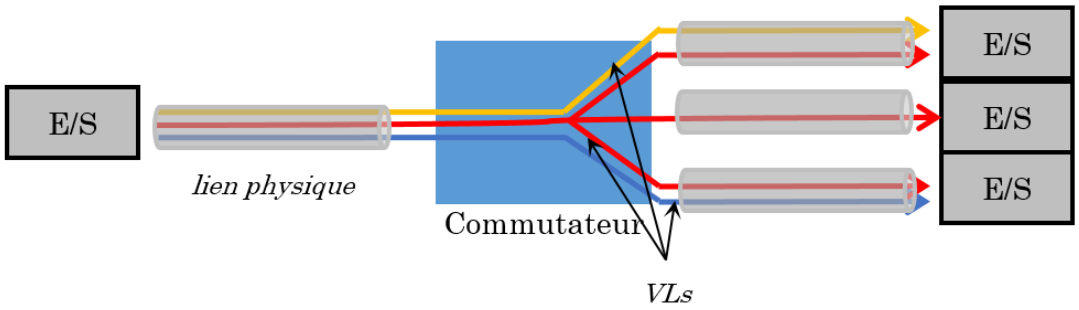

FIGURE 3.5 – Caractérisation des VL sur les liens physiques [Kem14]

Un VL est caractérisé par les propriétés suivantes :

— un identifiant unique.
— un E/S source.
— un chemin statique de E/S source vers des E/S destinataires.
— la taille maximale de la trame qu'il transmet.
— un *Bandwidth Allocation Gap* (BAG) : le temps minimal séparant deux trames consécutives sur le VL.

Une approche holistique [GPH12] a été proposée pour l'analyse du délai de bout en bout des VLs traversant le réseau AFDX. Elle reprend le principe de la méthode holistique dans [MMG04] pour les commutateurs du réseau. Le pire temps de traversée de bout en bout d'un flux correspond à la somme des pires temps de traversée locaux des noeuds (en particulier ports de sortie de switchs) traversés par le flux.

Le *Network Calculus* [C+91, Cru91, Gri04] présente une autre approche pour l'analyse du réseau AFDX en se basant sur l'algèbre *Min-Plus* [CGQ05]. Cette méthode est retenue par Airbus pour certifier le réseau AFDX dans l'A380. Selon cette méthode, un flux sera modélisé par une **courbe d'arrivée** représentant la quantité maximale de *bits* générés par le flux en fonction de temps. Les courbes d'arrivée sont de type **seau percé** (i.e., $\gamma_{r,b}(t) = b + rt$). Un commutateur sera modélisé par une *courbe de service* représentant la quantité de *bits* traités par le commutateur en fonction de temps. Les courbes de service sont de type **latence-débit** (i.e., $\beta_{R,T}(t) = max(0, R(t-T))$). La figure 3.6 présente 2 types de courbe introduits précédemment. Le *Network Calculus* traite de manière séquentielle chaque noeud traversé. Pour l'analyse d'un noeud, les courbes d'arrivée des flux d'entrée i (i.e., $\gamma_{r_i,b_i}(t) = r_i t + b_i$) traversant le noeud sont sommées pour avoir une courbe d'arrivée unique $\gamma_{r,b}(t)$ dont $r = \sum_i r_i$ et $b = \sum_i b_i$. Le *Network Calculus* permet de déduire à partir des courbes d'arrivée et des courbes de service d'un noeud le temps maximal de traversée du noeud et également sa courbe de sortie qui joue le rôle de la courbe d'arrivée du prochain noeud. Le temps maximal de traversée correspond à l'écart horizontal maximal entre deux courbes (i.e., $h(\gamma, \beta)$ dans la figure 3.7). La quantité maximale des *bits* présents dans le noeud à tout moment est représentée par l'écart vertical maximal entre deux courbes (i.e., $v(\gamma, \beta)$ dans la figure 3.7).
Supposant que la courbe de service soit $\beta_{R,0}(t) = Rt$, selon [GRRR13] la courbe de sortie de chaque flux d'entrée i sera donnée par l'équation 3.18.

$$\gamma_{r_i,b_i+r(b-b_i)/R}(t) = r_i t + b_i + \frac{r_i(b-b_i)}{R} \tag{3.18}$$

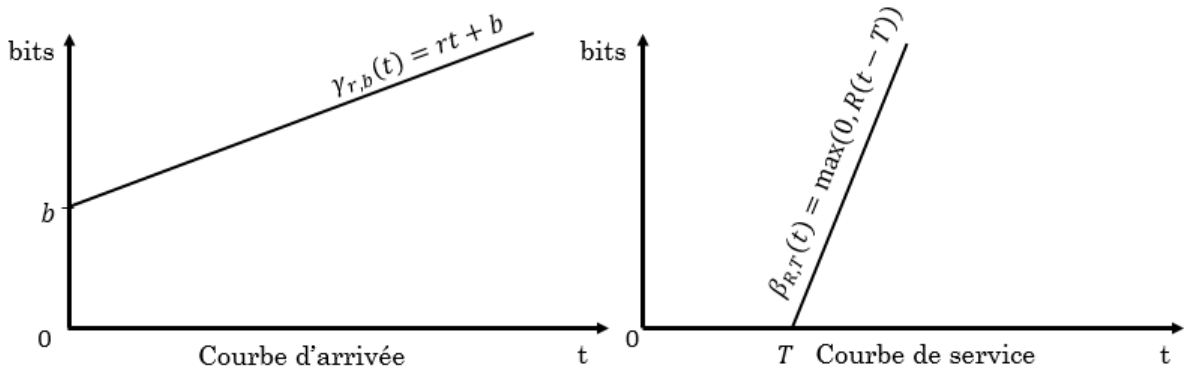

FIGURE 3.6 – Courbe d'arrivée d'un flux et courbe de service d'un commutateur

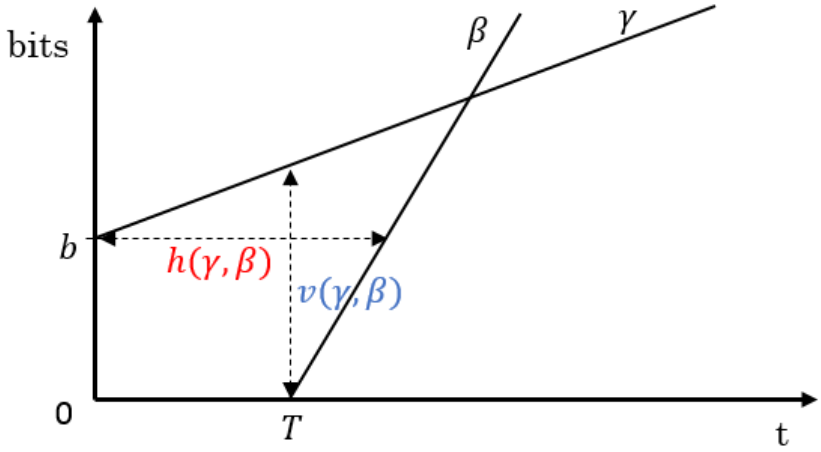

FIGURE 3.7 – Courbe d'arrivée d'un flux et courbe de service d'un commutateur

3.5 Outils d'analyse

Plusieurs tests d'analyse ont été implémentés afin de faciliter leur utilisation et aussi faciliter leur intégration dans une chaîne de conception outillée. Cette implémentation a donné lieu à des prototypes et des outils académiques et commerciaux. Bien entendu, comme il y a toujours des nouveaux travaux de recherche liés au tests d'analyse, il n'existe aucun outil qui offre tous les tests existants dans la littérature. Nous présentons par la suite quelques outils d'analyse académiques.

3.5.1 Cheddar

Cheddar [SLNM04] est un outil d'analyse d'ordonnançabilité et de simulation, développé en Ada. Cheddar supporte des tests classiques d'ordonnançabilité et des politiques d'ordonnancement usuelles (telles que RM, EDF, DM, LLF, FIFO) sur monoprocesseur. Il permet ainsi d'analyser les systèmes avec des ressources partagées et des contraintes de précédence. Cheddar fournit une interface graphique permettant de manipuler le modèle d'entrée. Il peut aussi prendre en entrée des modèles en format AADL.

3.5.2 MAST

MAST (*Modeling and Analysis Suite for real-Time applications*) [HGGM01] est une suite d'outils pour la modélisation des systèmes temps réel et l'analyse d'ordonnançabilité. MAST se base sur le modèle de **transactions** telles que définies dans [Tin94, PH98]. MAST supporte les analyses de temps de réponse, le calcul de temps de blocage et l'analyse de sensibilité sur

l'architecture monoprocesseur et distribuée. Le modèle d'entrée de MAST peut être exprimé de deux manières : à travers de son interface graphique ou à l'aide d'une grammaire spécifique. MAST restitue sous un format spécifique de type XML les résultats d'analyse.

3.5.3 Simso

Simso (*SImulation of Multiprocessor Scheduling with Overheads*) [CHD14] est un outil de simulation d'ordonnancement pour des architectures multiprocesseurs. Simso a été développé en Python [pyt]. Il a été créé avec l'objectif de faciliter la comparaison, l'évaluation et la compréhension des politiques d'ordonnancement en se basant sur la simulation. Simso contient plus de vingt-cinq politiques d'ordonnancement et permet d'être enrichi simplement en définissant de nouvelles politiques en Python. Simso fournit une interface graphique pour faciliter la manipulation du modèle d'entrée et accepte également en entrée des fichiers XML conformes à un schéma spécifique.

3.5.4 Roméo

Roméo est un outil développé par l'équipe temps réel du laboratoire IRCCyN. Sa première version a vu le jour en octobre, 2004 [IRC]. Le logiciel Roméo se compose de deux parties : une interface graphique et un noyau de calcul MERCUTIO. L'interface graphique permet à l'utilisateur d'éditer les modèles sous forme de réseau de Petri temporel et exprimer les contraintes sur les paramètres. Grâce au noyau de calcul MERCUTIO, Romeo peut effectuer une vérification de modèle sur les modèles de réseaux de Petri temporel. Les propriétés à vérifier doivent être exprimées sous forme de formules exprimées en logique temporelle quantitative (TCTL [ACD93]).

3.6 Discussion

Comme le lecteur peut le constater, nous avons à chaque fois, en particulier pour les problèmes les plus classiques trouvés dans l'ordonnancement temps réel, essayé de donner de façon exhaustive les hypothèses sous-jacentes aux modèles, plateformes, et méthodes d'analyse, afin de mettre en avant le fait qu'il faut absolument connaître parfaitement les caractéristiques du système avant d'appliquer une analyse.

De nombreux tests d'ordonnançabilité ont été proposés depuis les années 70 pour traiter différents types des systèmes temps réel en raison de l'évolution des composants matériels ainsi que l'architecture logicielle. En fait, chaque test d'analyse est lié à une situation d'application spécifique, i.e., un ensemble d'hypothèses liées à l'architecture logicielle et matérielle des systèmes étudiés. Dans le reste de ce manuscrit, nous appelons **contexte d'analyse** l'ensemble des hypothèses requises pour qu'un test d'analyse soit applicable. Autrement dit, le résultat d'un test d'analyse ne peut jamais être fiable si le système ne correspond pas parfaitement au contexte d'analyse du test appliqué. En analysant la littérature (un ensemble d'articles publiés dans les conférences connues du domaine temps réel : RTSS, ECRTS, RTNS, RTCSA, etc.), nous avons constaté la présence d'une "jungle" de tests d'analyse dont les modèles d'analyse sont parfois bien définis dans les articles scientifiques, et parfois noyés dans les différentes sections et discussions, voire même laissés parfois implicites. Par exemple, la figure 3.8 montre deux extraits issus de deux articles où dans un cas (partie A de la figure) le contexte d'application du test est éparpillé dans le texte et dans l'autre cas (partie B) le contexte est bien défini sous forme d'un ensemble d'hypothèses qui doivent être vérifiées par le système souhaitant être analysé par ce test.

La problématique présentée ici se joint à celle présentée dans la discussion du chapitre précédent. L'idée est de montrer que l'utilisation des outils d'analyses actuels n'est pas suffisante pour aider les concepteurs lors de l'analyse. En effet, les outils d'analyse permettent d'automatiser le

A

Consider such a real-time system, in which a number of devices are connected to a single processor computer system at different priority levels. Initially, assume that

⋮

Let T_i be the minimum time between successive inputs from the device connected at priority level i, and let C_i be the maximum associated processing requirement.

⋮

composed of n processes, one servicing each device. For simplicity, let these processes be independent and have no communication with each other. Then, provided the

B

3 The Environment

To obtain any analytical results about program behavior in a hard-real-time environment, certain assumptions must be made about that environment. Not all of these assumptions are absolutely necessary, and the effects of relaxing them will be discussed in a later section.

(A1) The requests for all tasks for which hard deadlines exist are periodic, with constant interval between requests.

(A2) Deadlines consist of run-ability constraints only – i.e. each task must be completed before the next requests for it occurs.

(A3) The tasks are independent in that requests for a certain task do not depend on the initialization or the completion of requests for other tasks.

(A4) Run-time for each task is constant for that task and does not vary with time. Run-time refers to the time which is taken by a processor to execute the task without interruption.

(A5) Any non-periodic tasks in the system are special; they are initialization or failure-recovery routines; they displace periodic tasks while they themselves are being run, and do not themselves have hard, critical deadlines.

FIGURE 3.8 – Des hypothèses extraites de deux articles

calcul (qui peut se faire à la main) mais ne permettent pas d'orienter vers le test le plus approprié ou le modèle de tâche considéré au moment de l'analyse. A cet égard, nous considérons un exemple d'un système qui se compose de quatre tâches indépendantes. Chaque tâche est définie par un ensemble de propriétés introduites dans le tableau 3.3 suivant :

Tâche	Le pire temps d'exécution	Echéance	Période	Date de réveil	Priorité
Tâche 1	3ms	15ms	20ms	2ms	4
Tâche 2	4ms	8ms	23ms	0ms	3
Tâche 3	5ms	13ms	23ms	5ms	2
Tâche 4	9ms	23ms	23ms	7ms	1

TABLEAU 3.3 – Propriétés des tâches

La tâche 1 dont la priorité est 4 est la plus prioritaire et respectivement la tâche 4 dont la priorité est 1 est la moins prioritaire. Le système de tâches s'exécute sur une architecture monoprocesseur. Le système est analysé par deux outils différents : MAST et Rt-Druid. Les résultats d'analyse (i.e., le pire temps de réponse) sont présentés dans le tableau 3.4. Selon le résultat obtenu, le système est non-ordonnançable d'après Rt-Druid (la dernière tâche rate son échéance) et ordonnançable d'après MAST (toutes les tâches respectent leurs échéances).

La différence n'est pas due à la mauvaise implémentation des tests dans les outils, mais elle

Tâche	Le pire temps de réponse (MAST)	Le pire temps de réponse (Rt-Druid)
Tâche 1	3ms	3ms
Tâche 2	7ms	7ms
Tâche 3	8ms	12ms
Tâche 4	21ms	33ms

Tableau 3.4 – Les résultats d'analyse par MAST et Rt-Druid

est due aux méthodes d'analyse appliquées automatiquement par l'outil. En effet, MAST traite le système comme un modèle de transaction [PH98, RGRR12] et Rt-Druid le traite comme un modèle de tâche périodique conventionnel de Liu&Layland [LL73a]. Le modèle de transaction est le modèle qui prend en compte les dates de réveil des tâches, cependant le modèle conventionnel de Liu&Layland ne l'est pas. L'objectif de cet exemple est de montrer au lecteurs que le choix du modèle de tâche ainsi que l'analyse associée est une étape importante.

3.7 Conclusion

Dans ce chapitre, nous avons premièrement introduit les modèles de tâches temps réel. Ensuite, nous avons présenté les politiques d'ordonnancement et les tests d'ordonnançabilité associés à l'architecture monoprocesseur. Les politiques EDF et OPA sont dominants par rapport à d'autres politiques pour l'architecture monoprocesseur. Pour l'architecture multiprocesseur, l'ordonnancement partitionné permet de tranfromer le problème d'ordonnancement multiprocesseur en un ensemble de problèmes d'ordonnancement monoprocesseur. Cette approche n'autorise pas la migration et donc l'utilisation processeur n'est pas complète. Au contraire, l'ordonnancement global permet d'obtenir l'optimalité (donc l'utilisation processeur est plus élevée) mais présente un grand nombre de préemptions et de migrations. L'ordonnancement semi-partitionné présente un bon compromis entre l'ordonnancement global et l'ordonnancement partitionné en permettant d'obtenir l'utilisation processeur relativement élevée avec un nombre acceptable de préemptions et migrations. Pour l'architecture distribuée, nous avons présenté les deux réseaux les plus utilisés en avionique, à savoir CAN et AFDX. Avant de conclure ce chapitre, nous avons présentés quelques outils d'analyse implémentant les différents tests. Nous avons également discuté les problèmes qu'un architecte peut rencontrer afin d'utiliser le bon test d'analyse. Cette difficulté réside dans la multitude des modèles de tâches, des tests d'analyse ainsi que l'implémentation de ces derniers qui sont éparpillés dans différents outils.

Deuxième partie

Contributions

Chapitre 4

Contraintes de précédence à base de sémaphore pour une communication multi-périodique déterministe

Sommaire

Ce chapitre correspond à la première contribution, de cette thèse, portant sur les communications déterministes. La plupart des langages de modélisation ne supportent pas les communications déterministes multi-périodiques. Il en va de même pour les outils d'analyse qui ne considèrent pas ce type de contraintes en général. Cependant, celles-ci sont fréquentes dans les applications multitâches. Nous proposons une extension d'un modèle de précédence entre tâches de périodes différentes (communication multi-périodique). Ce modèle de précédence est inspiré du concept de sémaphore et plus spécifiquement le paradigme producteur/consommateur. Le nouveau modèle de précédence proposé consiste à renforcer la sémantique en supportant l'existence des cycles du graphe de précédence. Un outil d'analyse a été développé et adossé au formalisme AADL.

4.1 Introduction

L'exécution des systèmes embarqués nécessite souvent des communications. De nombreux patrons de communication inter-tâches ont été proposés tels que les patrons de communication asynchrone (tableau noir ou module de données) et les patrons de communication synchrone (boîte aux lettres, files de messages, etc.). Les patrons de communication de type producteur/-consommateur, sont appelées synchrones, ou plus précisément synchrones faiblement couplées pour les différencier des "rendez-vous" à la Ada, qui sont synchrones fortement couplées.

Dans le contexte temps réel, les communications doivent être **déterministes**. Un système est **fonctionnellement déterministe** si les sorties obtenues sont toujours les mêmes pour les mêmes entrées tandis qu'un système est **temporellement déterministe** si ses sorties sont toujours produites dans les mêmes intervalles de temps. Sans déterminisme (fonctionnel et temporel), la communication ne peut jamais être validée à cause d'un comportement (fonctionnel ou temporel) inattendu.

Le reste de ce chapitre est organisé comme suit. La section 4.2 présente des concepts et des modèles de communication déterministe. La section 4.3 contribue à renforcer la sémantique de la contrainte de précédence à base de sémaphore (SPC). La section 4.4 présente une nouvelle politique d'ordonnancement et le test associé pour le système utilisant les SPC. La section 4.5 présente une extension d'AADL pour prendre en compte les SPC. Enfin, dans la section 4.6 nous présentons une étude de cas pour illustrer l'utilisation des SPC. Des détails sur l'implémentation de l'outil sont donnés dans la section 4.7. La section 4.8 conclut ce chapitre.

4.2 Modèles de représentation de contraintes de précédence

La communication entre les tâches impose souvent une contrainte de précédence entre la tâche produisant le message et la tâche le consommant suivant le paradigme producteur/consommateur. La tâche productrice est appelée **prédécesseur** et la tâche consommatrice est appelée **successeur**. Plusieurs modèles ont été proposés pour exprimer les précédences entre les tâches (ou instances) : *Simple Precedence Constraint*, *Generalized Precedence Constraint* (GPC), *Repetitive Precedence Constraint* (RPC) et *Semaphore Precedence Constraint* (SPC). Nous présentons par la suite ces différents modèles.

4.2.1 La contrainte de précédence simple

La contrainte de précédence simple (*Simple Precedence Constraint*) [Bla76, CSB90] est le premier modèle traitant ce genre de problèmes. La notation de *Simple Precedence Constraint* : $\alpha_i \rightarrow \alpha_j$ est premièrement appliquée à des instances pour exprimer l'instance α_i précède l'instance α_j (i.e., l'instance α_i doit terminer l'exécution avant que l'instance α_j commence son exécution). Le modèle est alors étendu pour exprimer la précédence entre les tâches de même période. La tâche τ_i précède τ_j implique que $\forall k \in \mathbb{N}$, l'instance $\tau_{i.k}$ précède l'instance $\tau_{j.k}$.

Il existe deux approches pour assurer l'ordre d'exécution des tâches (ou instances) sous contraintes de précédence. La première approche consiste à traduire une implémentation qui utiliserait un mécanisme de synchronisation équivalent aux sémaphores privés. Un sémaphore privé à compte permet d'implémenter une contrainte de précédence entre un prédécesseur et un successeur. Ce sémaphore est initialisé à zéro, chaque instance du prédécesseur en vend une instance, alors que chaque instance du successeur en prend une instance. Cette approche peut mener à des anomalies d'ordonnancement [RRGC02, Gra69] lorsque les précédences ne sont pas consistantes avec les priorités. Les précédences sont **consistantes** avec les priorités si pour n'importe quelle précédence $\tau_{i,k} \rightarrow \tau_{j,k'}$, la priorité de $\tau_{i,k}$ est supérieure à la priorité de $\tau_{j,k'}$. Dans ce cas, elle peut s'analyser avec une extension du calcul de temps de réponse qui va donner une borne supérieure sur les temps de réponse [RRGC02]. La deuxième approche applicable, quand les précédences sont consistantes avec les priorités, consiste à modifier des propriétés des

tâches sous certaines contraintes. Cette approche ne fonctionne que si les tâches sont strictement périodiques. Les tâches sont considérées comme indépendantes après la modification. Dans la suite, on ne considère que la deuxième approche. Chetto et al. [CSB90] montrent que pour respecter les contraintes de précédence simples (i.e, *Simple Precedence Constraint*), nous devons seulement nous assurer que :

— (i) Une instance n'est jamais activée avant ses prédécesseurs.

— (ii) Une instance est plus prioritaire que ses successeurs.

D'après (ii), seules les relations de précédences consistantes avec les priorités sont donc considérées. La propriété (i) peut être assurée en remplaçant la date de réveil d'une instance par la date de réveil maximale de ses prédécesseurs. Par conséquent, la nouvelle date de réveil de l'instance i (i.e., r_i^*) sera calculée par l'équation 4.1 dont $preds(\alpha_i)$ représente l'ensemble de prédécesseurs de l'instance α_i :

$$r_i^* = max(r_i, \max_{\alpha_j \in preds(\alpha_i)} r_j^*) \tag{4.1}$$

L'ajustement de la date de réveil sera effectué en ordre topologique, depuis les tâches qui n'ont pas de prédécesseurs où $r_i^* = r_i$ vers les tâches dont les prédécesseurs sont tous ajustés.

La propriété (ii) est assurée en affectant les priorités de manière cohérente avec les contraintes de précédence. L'ajustement sera effectué en fonction de la politique d'ordonnancement utilisée pour assurer : $\tau_i \rightarrow \tau_j \Rightarrow prio(\tau_i) > prio(\tau_j)$. L'urgence relative d'une tâche dépend non seulement de son délai mais aussi de l'urgence de ses successeurs. Pour la politique DM, les délais relatifs des tâches seront ajustés suivant l'équation 4.2 dont $succs(\tau_i)$ représente l'ensemble de successeurs de la tâche τ_i :

$$D_i^* = min(D_i, \min_{\tau_j \in succs(\tau_i)} (D_j^* - C_j)) \tag{4.2}$$

Tandis que selon la politique EDF, le délai absolu de chaque instance sera ajusté suivant l'équation 4.3 :

$$d_i^* = min(d_i, \min_{\alpha_j \in succs(\alpha_i)} (d_j^* - C_j)) \tag{4.3}$$

Comme la date de réveil, l'ajustement de délai sera également effectué en ordre topologique : depuis les tâches qui n'ont pas de successeurs vers les tâches dont les successeurs sont tous ajustés.

Après l'ajustement des paramètres, les analyses d'ordonnançabilité classiques pour les systèmes de tâches indépendantes peuvent être appliquées.

4.2.2 La contrainte de précédence généralisée

Le modèle à base de contrainte de précédence généralisée (*Generalized Precedence Constraint* - GPC) a été proposé dans [RCR01]. GPC est une extension de la contrainte de précédence simple pour supporter la communication multi-périodique où la période d'une tâche soumise à la contrainte est un multiple de l'autre. GPC $\tau_i \rightarrow \tau_j$ exige que le nombre des instances commencées $S_j(t)$ à tout instant t et le nombre des instances terminées $E_i(t)$ satisfassent l'équation 4.4 :

$$E_i(t) \times T_i \geq S_j(t) \times T_j \tag{4.4}$$

La dépendance entre les tâches peut être modélisée par un graphe orienté dont les noeuds représentent les tâches et les arcs représentent les contraintes de précédence GPC. Le graphe GPC doit être acyclique.

L'analyse d'ordonnançabilité des systèmes sous les contraintes GPC consiste à déplier le graphe GPC afin d'obtenir un graphe de contraintes de précédence simple. Ensuite, il suffit d'appliquer les modifications décrites dans le paragraphe précédent et finalement utiliser les tests

d'ordonnançabilité classiques sur les systèmes de tâches indépendantes. La technique de dépliage consiste à dupliquer les tâches sur l'hyperpériode. Une tâche τ_i de période T_i sera remplacée de manière équivalente par n_i duplicata de période $n_i \times T_i$ où $n_i = \frac{ppcm(T_1,...,T_n)}{T_i}$ avec les contraintes de précédence simple entre les tâches ainsi obtenues. Le $k^{ième}$ duplicata de la tâche τ_i est noté τ_i^k avec $k \in \{1..n_i\}$. La $m^{ième}$ instance du duplicata τ_i^k représente la $((m-1)n_i+k)^{ième}$ instance de la tâche originelle τ_i. La technique de dépliage est illustrée dans la figure 4.1.

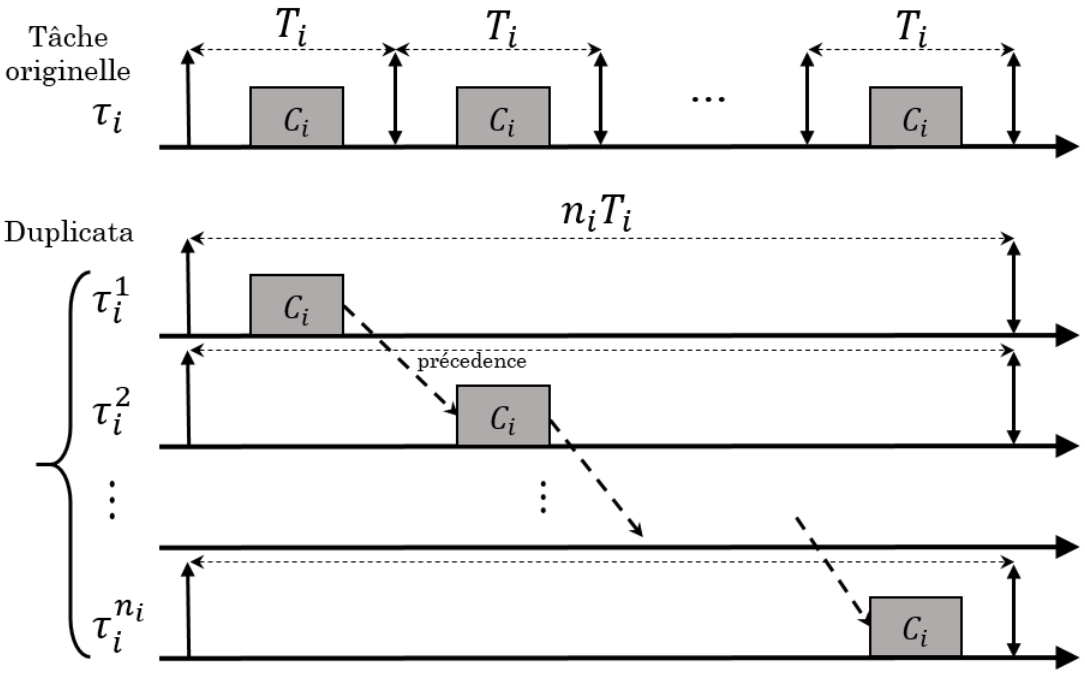

FIGURE 4.1 – Illustration de la technique de dépliage

Soit $\tau_i \rightarrow \tau_j$ une contrainte de type GPC, n_i et n_j sont les nombres de duplicata des tâches τ_i et τ_j respectivement. Chaque contrainte GPC sera remplacée par :

— n_i contraintes de précédence simples si $T_i > T_j$:
$\forall k \in \{1..n_i\} \tau_i^k \rightarrow \tau_j^{a_k}, a_k = \left\lfloor \frac{(k-1)T_i}{T_j} \right\rfloor + 1.$

— n_j contraintes de précédence simples sinon :
$\forall k \in \{1..n_j\} \tau_i^{b_k} \rightarrow \tau_j^k, b_k = \left\lceil \frac{kT_i}{T_j} \right\rceil.$

4.2.3 La contrainte de précédence répétitive

La contrainte de précédence répétitive (*Repetitive Precedence Constraint* - RPC) a été proposée par Cucu et al. [CKS02]. Chaque dépendance entre la tâche τ_i et la tâche τ_j est encodée par une matrice $Code_{ij}$ de taille $N \times M$ dont $N = \frac{ppcm(T_i,T_j)}{T_i}$ et $M = \frac{ppcm(T_i,T_j)}{T_j}$. Le prédicat $Code_{ij}(n,m) = 1$ si la $n^{ième}$ instance de la tâche τ_i précède la $m^{ième}$ instance de la tâche τ_j, le prédicat $Code_{ij}(n,m) = 0$ sinon. La représentation de la relation de précédence utilise donc un espace non-polynomial. Aucune restriction n'est imposée aux prédicats de précédence, ce qui peut entraîner l'incohérence et la redondance. Les auteurs ont proposé une politique d'ordonnancement hors-ligne et non-préemptive pour les systèmes de tâches sous cette contrainte de précédence. Cette politique d'ordonnancement est prouvée pour être optimale dans la classe des politiques hors-ligne non-préemptives, elle représente également un test d'ordonnançabilité.

4.2.4 La contrainte de précédence à base de sémaphore

La contrainte de précédence à base de sémaphore (*Semaphore Precedence Constraint* - SPC) a été proposée par Forget et al. [FBG+10]. SPC s'inspire du concept de sémaphore et notamment du paradigme de m-n producteurs/consommateurs. Concrètement, une contrainte SPC est désignée par : $\tau_i \xrightarrow{h} \tau_j$, le comportement des tâches soumises à la contrainte dépend d'un sémaphore à compte, initialisé à $h \in \mathbb{N}$, on peut aussi le voir de façon équivalente comme le nombre de messages initialement présents dans la file de messages du producteur/consommateur. La tâche source joue le rôle du producteur. A chaque fois que la tâche source termine son exécution, elle ajoute une valeur égale à sa période au compteur du sémaphore. La tâche cible joue le rôle du consommateur. A chaque fois que la tâche cible commence son exécution, elle déduit une valeur égale à sa période à partir du compteur du sémaphore. L'idée principale est que le consommateur à l'instant t ne peut pas prendre plus que la valeur du compteur du sémaphore à cet instant. Lorsqu'une tâche consommatrice essaie de prendre plus de valeur que la valeur du compteur, elle sera bloquée jusqu'au moment où la valeur du compteur devient suffisante. Comme GPC, le graphe de SPC proposé par [FBG+10] ne peut pas avoir de cycles (i.e., *Directed Acyclic Graph*).

Pour ordonnancer les systèmes de tâches soumises aux SPC, chaque contrainte SPC sera remplacée par un ensemble de contraintes de précédence simple. Ainsi, pour chaque SPC $\tau_i \xrightarrow{h} \tau_j$, les contraintes de précédence simple seront ajoutées dans le graphe des instances selon l'une des deux manières suivantes :

— Pour l'instance $\tau_{j,k'}$, une contrainte $\tau_{i,Pred_{i,j}(k)} \rightarrow \tau_{j,k'}$ est ajoutée
$Pred_{i,j}(k') = \left\lceil \frac{(k'+1)T_j - h}{T_i} \right\rceil - 1$
L'instance $\tau_{i,Pred_{i,j}(k)}$ est appelée le prédécesseur direct de $\tau_{i,k}$.

— Pour l'instance $\tau_{i,k}$, une contrainte $\tau_{i,k} \rightarrow \tau_{j,Succ_{i,j}(k)}$ est ajoutée
$Succ_{i,j}(k) = \left\lfloor \frac{kT_i + h}{T_j} \right\rfloor$
L'instance $\tau_{j,Succ_{i,j}(k)}$ est appelée le successeur direct de $\tau_{i,k}$.

Les systèmes peuvent être ordonnancés à priorités fixes aux tâches ou à priorités fixes aux travaux. Pour des priorités fixes aux tâches, une politique d'ordonnancement adaptée de l'algorithme OPA [Aud91] a été proposée [FBG+10]. Cette politique tente d'affecter les priorités en ordre croissant et topologique (les successeurs seront affectés en premier) aux tâches et vérifie l'ordonnançabilité de la tâche affectée par simulation. Si la tâche est ordonnançable sous une priorité, alors la priorité lui sera affectée officiellement. Cette politique d'ordonnancement est aussi un test exact d'ordonnançabilité. Pour des priorités fixes aux travaux, EDF peut être appliqué directement au système après l'ajustement des paramètres temporels à la Chetto et Chetto [CSB90] selon les contraintes de précédence simple. Les tests classiques d'ordonnançabilité pour EDF restent valables dans ce cas [FGPR11].

4.2.5 Patrons de communication déterministes et modèles de représentation

Plusieurs patrons de communication multi-périodique déterministes ont été proposés dans la littérature. Certains sont présentés dans la figure 4.2, dont les arcs orientés représentent des précédences entre les instances des tâches. Par exemple, au patron (a) toutes les $3k^{ième}$ instances de la tâche τ_i précèdent les $k^{ième}$ de la tâche τ_j (k commence par 0). Les patrons (g) et (h) représentent les précédences entre les tâches de périodes premières entre elles (e.g., (3,5)). Chaque patron de communication peut être exprimé à l'aide d'un des modèles de contraintes de précédence que nous avons abordés auparavant. L'expressivité des modèles de contraintes de précédence par rapport aux patrons de communication est présentée dans le tableau 4.1.

Les SPC (*Semaphore Precedence Constraint*) se basent sur l'ensemble de concepts classiques et un mécanisme simple qui assure une implémentation facile et efficace. En comparaison avec RPC, SPC est un peu moins expressif puisqu'il ne peut pas exprimer un patron de communication

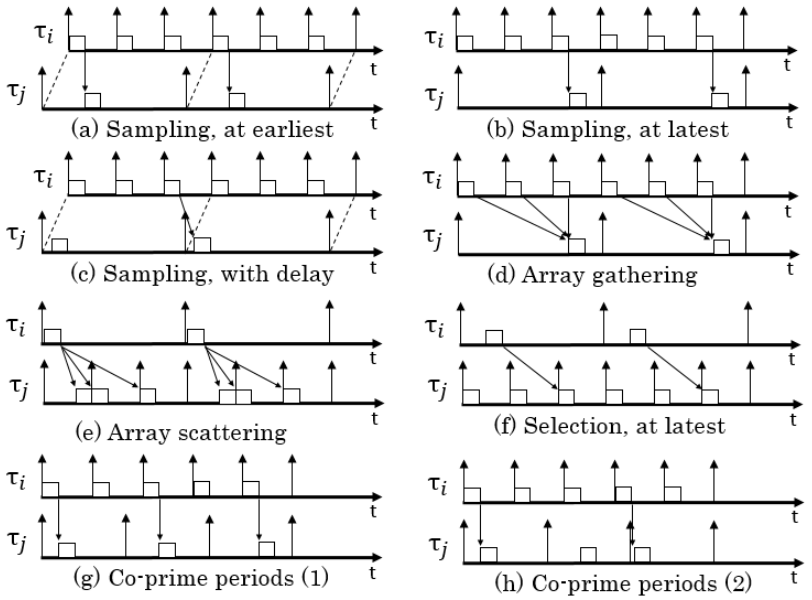

FIGURE 4.2 – Les patrons de communication déterministe

TABLEAU 4.1 – Modèles de précédence exprimant les patrons de communication

Patron	RPC	GPC	SPC
(a)	✓	✗	✓
(b)	✓	✓	✓
(c)	✓	✗	✓
(d)	✓	✓	✓
(e)	✓	✓	✓
(f)	✓	✗	✓
(g)	✓	✗	✓
(h)	✓	✗	✗

que l'on peut souhaiter entre tâches de périodes premières entre elles (i.e., le dernier patron dans la figure 4.2). Ce cas paraît cependant très spécifique et peu conforme à ce qu'une application réelle pourrait vouloir exprimer. Rappelons aussi que les RPC requièrent une modélisation de taille exponentielle, alors que les SPC sont un modèle de précédences quadratique (tâche à tâche) dont chaque relation est caractérisée uniquement par un entier (valeur initiale du sémaphore privé). Par conséquent, le modèle SPC est le modèle le plus expressif que l'on puisse exprimer dans une taille raisonnable, offrant un bon compromis entre l'expressivité et la taille. De plus, concernant l'aspect analyse temporelle, une politique d'ordonnancement qui est aussi un test d'ordonnançabilité pour les priorités fixes aux tâches a été proposée en adaptant l'OPA (*Optimal Priority Assignment* [Aud91]) dans [FBG+10]. Aussi, des ajustements ont été proposés dans [FGPR11] pour supporter EDF comme ordonnanceur d'un système de tâches soumises à des SPC.

4.2.6 Communication déterministe en AADL

Néanmoins, les langages de conception tels que AADL ne supportent que les patrons de communication entre les tâches de même période. La figure 4.3 présente les deux patrons de communication supportés par AADL où les tâches τ_i, τ_j sont de même période. Ces deux patrons sont (i) la **communication immédiate** et (ii) la **communication retardée**.

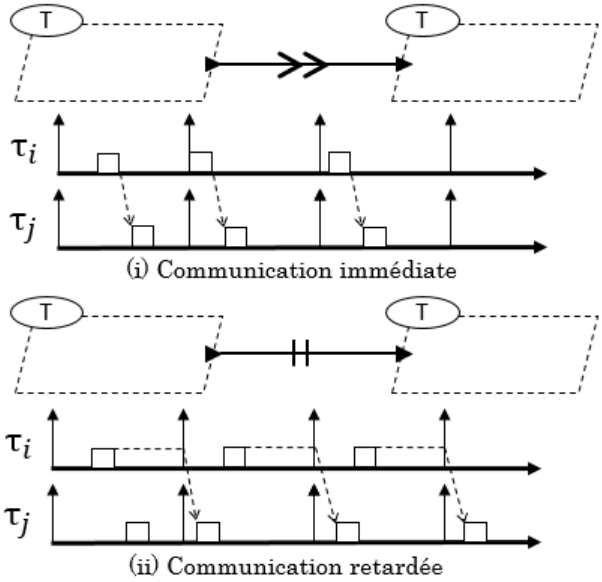

FIGURE 4.3 – Les patrons de communication d'AADL

Une communication immédiate signifie que le résultat produit par la tâche productrice après son exécution est immédiatement disponible pour la tâche consommatrice. Cependant, une communication retardée signifie que le résultat de la tâche productrice n'est disponible pour la tâche consommatrice qu'après son prochain réveil. Dans le cas du multi-périodique, ces patrons de communication ne sont pas définis.

4.2.7 Contributions

Dans ce chapitre, nous allons :

— Renforcer la sémantique des SPC en (i) permettant au sémaphore d'avoir une valeur initiale négative, ce qui permet d'augmenter les patrons de communications modélisables, et (ii) en permettant des cycles dans le graphe de précédences.

— Présenter une approche basée sur le dépliage du graphe de SPC pour l'analyse d'ordonnançabilité des systèmes utilisant ces SPC étendues.

— Implémenter le SPC dans AADL pour supporter les patrons de communication multi-périodiques.

4.3 Renforcement de la sémantique des SPC

4.3.1 Valeur initiale négative du sémaphore

Dans [FBG+10], le compteur sémaphore n'a que des valeurs non-négatives. Nous avons cependant rencontré des cas dans lesquels on avait besoin d'une valeur initiale négative du sémaphore. Cela signifie qu'un certain nombre d'instances de la tâche source (productrice) doivent se produire avant que la valeur du sémaphore devienne non-négative et à partir de ce moment, la communication agit périodiquement. Ce type de comportement peut se trouver par exemple dans des systèmes pour lesquels un capteur doit donner plusieurs valeurs avant que le filtrage des valeurs envoyées ne permette leur exploitation. La figure 4.4 présente un exemple de valeur négative du sémaphore, τ_i, τ_j ont la même période qui est égale à 1. Un exemple concret de ce type de communication pourrait être que τ_1 fait une acquisition sur un capteur et applique un filtre moyenneur sur trois valeurs.

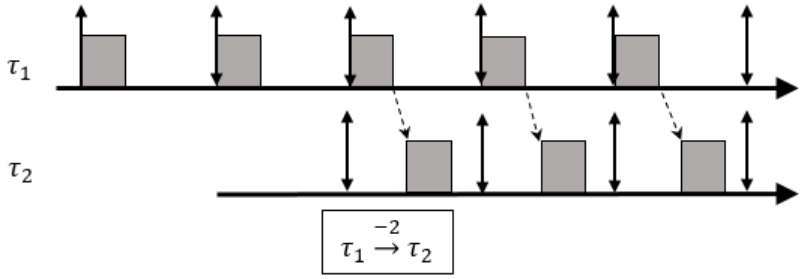

FIGURE 4.4 – Valeur initiale négative

4.3.2 Cycles dans le graphe de SPC

Dans [FBG+10], le graphe des SPC ne peut pas avoir de cycle. Dans cette thèse, nous allons supprimer cette contrainte. Par conséquent, le graphe de SPC est autorisé à avoir des cycles. Deux questions se posent. Bien entendu, on ne peut tolérer de dépendance cyclique, qui entraînerait un blocage. De plus, dans le cas de priorité fixe aux tâches, puisque les méthodes d'analyse développées imposent d'avoir des précédences consistantes avec les priorités, il est évident qu'un cycle dans le graphe de SPC (entraînant qu'une tâche est à la fois prédécesseur et successeur d'une tâche) pose un problème d'affectation de priorités. Cependant, dans le cas de priorités fixes aux travaux, il est possible d'avoir des cycles dans le graphe de SPC tant que le graphe déplié des instances ne contient pas de cycle, ainsi chaque instance qui en précède une autre peut obtenir une priorité supérieure à son ou ses successeurs. La figure 4.5 présente un exemple du graphe de SPC ayant un cycle.

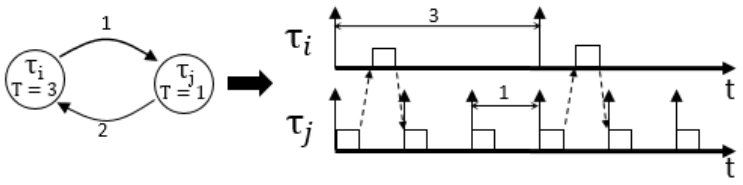

FIGURE 4.5 – Ordre total entre les instances de deux tâches

4.3.3 Dépliage du graphe de SPC

Un système de tâches contraint par SPC peut être représenté par un graphe où chaque tâche est représentée par un noeud et chaque SPC est représentée par un arc orienté. Le graphe de SPC peut être déplié en un graphe infini des instances. Dans le graphe déplié, chaque noeud représente une instance et chaque arc orienté représente une contrainte de précédence simple.

La figure 4.6 présente un exemple de dépliage du graphe de SPC en un graphe d'instances où chaque SPC est remplacée de manière équivalente par un ensemble de contraintes de précédence simple.

Évidemment, nous ne pouvons pas effectuer une analyse sur un graphe infini des instances. Nous allons démontrer que le graphe infini des instances est ultimement périodique. Cela veut dire que le graphe infini des instances se compose de deux parties : une partie initiale non répétitive et une partie suivante répétitive. Nous considérons un SPC $\tau_i \xrightarrow{h} \tau_j$. Nous montrons alors que le graphe déplié correspondant aux contraintes de précédence simple se comporte de manière ultimement périodique avec la période $H_{ij} = ppcm(T_i, T_j)$. Algébriquement, les arcs du graphe déplié représentent une fonction bijective $Succ_{i,j}(k) = k'$ dont la fonction inverse est $Pred_{i,j}(k') = k$, représentée dans le graphe par un arc allant de $\tau_{i,k}$ à $\tau_{j,k'}$. Le graphe déplié pour un seul SPC est ultimement périodique si et seulement si les fonctions $Succ_{i,j}$ et $Pred_{i,j}$ sont

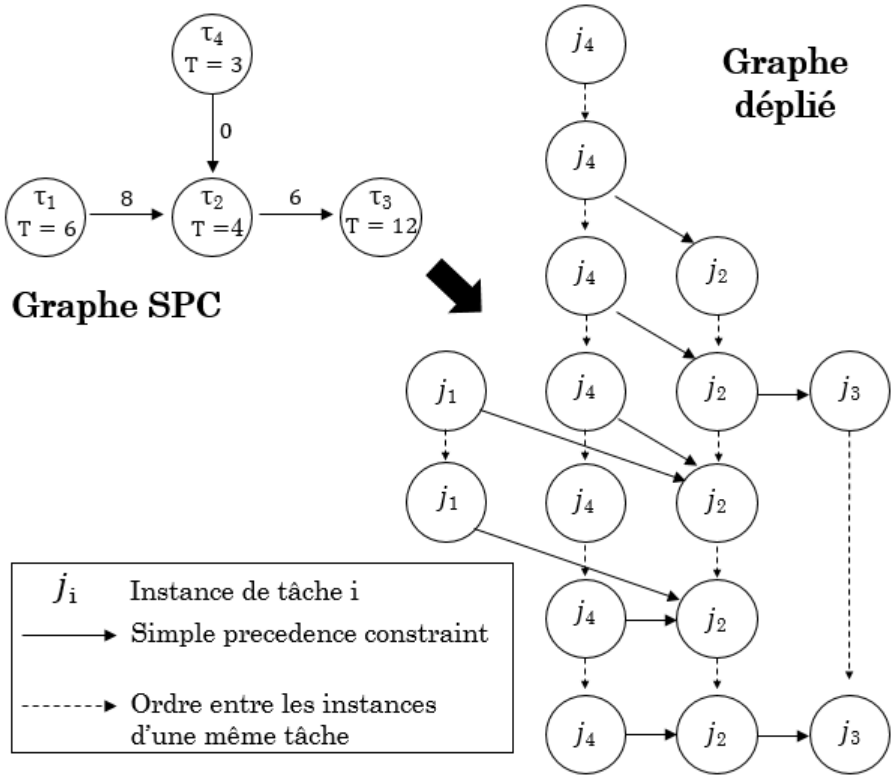

FIGURE 4.6 – Dépliage du graphe de SPC sur un intervalle de temps

ultimement périodiques. Par composition, le graphe déplié d'un graphe de SPC est également périodique avec une période égale au plus petit commun multiple (*ppcm*) des périodes de chacun des SPC.

Lemme 1 *La fonction successeur est périodique de période* $\frac{H_{ij}}{T_i}$ *pour toute instance d'indice supérieur ou égal à* $-\frac{h_{i,j}}{T_i}$*, formellement,* $\forall k \in \mathbb{N} \wedge k \geq -\frac{h_{i,j}}{T_i}, Succ_{i,j}(k + \frac{H_{ij}}{T_i}) = Succ_{i,j}(k) + \frac{H_{ij}}{T_j}$

Preuve 1 *Nous le prouvons algébriquement.* $Succ_{i,j}(k+\frac{H_{ij}}{T_i}) = \left\lfloor \frac{(k+\frac{H_{ij}}{T_i})T_i+h_{ij}}{T_j} \right\rfloor = \left\lfloor \frac{kT_i+H_{ij}+h_{ij}}{T_j} \right\rfloor$. *Or puisque* $k \geq -\frac{h_{i,j}}{T_i}$ *et* $H_{ij} = PPCM(T_i, T_j)$*, on peut écrire :* $\left\lfloor \frac{kT_i+H_{ij}+h_{ij}}{T_j} \right\rfloor = \left\lfloor \frac{kT_i+h_{ij}}{T_j} \right\rfloor + \frac{H_{ij}}{T_j}$.

Lemme 2 *La fonction prédécesseur est périodique de période* $\frac{H_{ij}}{T_j}$ *pour toute instance d'indice supérieur ou égal à* $\frac{h_{i,j}}{T_j} - 1$*, formellement,* $\forall k \in \mathbb{N} \wedge k \geq \frac{h_{i,j}}{T_j} - 1, Pred_{i,j}(k + \frac{H_{ij}}{T_j}) = Pred_{i,j}(k) + \frac{H_{ij}}{T_i}$

Preuve 2 *Nous le prouvons algébriquement.* $Pred_{i,j}(k + \frac{H_{ij}}{T_j}) = \left\lceil \frac{(k+\frac{H_{ij}}{T_j}+1)T_j-h_{i,j}}{T_i} \right\rceil - 1 = \left\lceil \frac{(k+1)T_j+H_{ij}-h_{i,j}}{T_i} \right\rceil - 1 = Pred_{i,j}(k) + \frac{H_{ij}}{T_i}$ *(Puisque* $k \geq \frac{h_{i,j}}{T_j} - 1$*)*

4.4 Politique d'ordonnancement et analyse d'ordonnançabilité

Les ajustements originaux des paramètres (date de réveil et échéance relative) dans [FGPR11, FBG+10] sont effectués en suivant l'ordre topologique au niveau des tâches. Par exemple, l'ajustement de la date de réveil est effectué progressivement à partir de tâches sans prédécesseurs, vers des tâches dont tous les prédécesseurs sont déjà ajustés. Malheureusement, nous ne pouvons

pas appliquer cet ordre à un graphe de SPC qui comporte des cycles car il existe toujours une tâche dont le prédécesseur (ou le successeur) n'a pas encore été ajusté.

Nous adaptons le processus original (i.e., les équations 4.1 et 4.3 du chapitre 3) en changeant le niveau d'ajustement des tâches aux instances dans le graphe déplié des instances. Bien entendu, nous ne pouvons qu'effectuer l'ajustement sur un **intervalle fini** de temps. Puisque le graphe déplié est finalement périodique (i.e., se compose d'une partie non-répétitive et de la répétition d'une partie répétitive), l'intervalle de test sera donc la partie non-répétitive plus une fois la partie répétitive. L'absence des cycles dans le graphe déplié est assurée en utilisant l'algorithme de Kahn [Kah62]. Le principe de l'algorithme de Kahn consiste à enlever du graphe au fur et à mesure les noeuds qui n'ont que les arcs d'entrée (ou que les arcs de sortie) jusqu'à ce qu'il n'y ait plus de noeud disponible, si le nombre de noeuds visités n'est pas le même que le nombre de noeud du graphe, le graphe déplié contient des cycles et n'est pas valide.

Considérons un exemple de graphe sur la figure 4.7. Au début, il y a les noeud E et F qui n'ont pas d'arcs d'entrée. On enlève ces deux noeuds du graphe, le nombre des noeuds visités est deux. Le graphe contient maintenant les noeuds C, A, D et B. Les noeuds C et A n'ont donc plus d'arcs d'entrée donc on les enlève du graphe. Le nombre des noeuds visités est quatre. Le noeud D n'a pas d'arcs d'entrée donc on l'enlève, le nombre des noeuds visités est 5. Il ne reste que le noeud B sur le graphe. On l'enlève, le nombre des noeuds visités est six, égal au nombre total des noeuds. Le graphe déplié n'a donc pas de cycle.

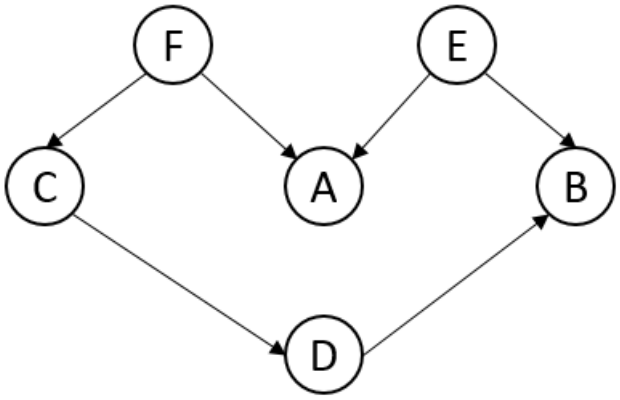

FIGURE 4.7 – Exemple de l'algorithme de Kahn

L'algorithme de détection de cycle possède une complexité linéaire à la taille du graphe.

Une fois l'ensemble des paramètres temporels des instances est ajusté à la Chetto et Chetto, on peut analyser le système comme s'il était constitué de tâches indépendantes. La politique d'ordonnancement EDF peut être appliquée directement aux instances ajustées. Nous pouvons alors effectuer une analyse d'ordonnançabilité classique (comme le test basé sur la *Demand Bound Function* [JS93a]) sur l'ensemble des instances ajustées.

4.5 Implémentation de SPC en AADL

AADL fournit un mécanisme d'extension via les *Property Sets*. Les *Property Sets* prédéfinis d'AADL supportent les propriétés de base des composants d'AADL. Basé sur les composants et les propriétés prédéfinis, les utilisateurs peuvent définir d'autres composants avec leurs propres propriétés. Nous modélisons les SPC en AADL par une propriété de type *Time*. Cette propriété peut être appliquée aux composants AADL de type *port connection*. La figure 4.8 présente l'extension de *Property Sets* pour exprimer les SPC. Dans le modèle SPC, la valeur h_{ij} du compte initial du sémaphore n'a pas d'unité. Cependant, le fait que les périodes de la tâche source et de la tâche cible peuvent être exprimées en différentes unités dans le modèle rend

SemaphorePrecedenceConstraint.aadl

```
1 property set SemaphorePrecedenceConstraint is
2     SPC : Timing_Properties::Time
3     applies to (port connection);
4 end SemaphorePrecedenceConstraint;
```

FIGURE 4.8 – Implémentation de SPC en *Property Sets*

nécessaire l'ajout d'une unité au niveau du compte. Celui-ci sera alors converti en une seule unité afin d'éviter toute ambiguïté.

4.6 Étude de cas

4.6.1 Description

Nous réutilisons un exemple industriel, qui est le *Flight Application Software* (FAS) pour réalimenter la station spatiale internationale (ISS [NAS19]). La figure 4.9 présente une vue globale de l'architecture logicielle du FAS.

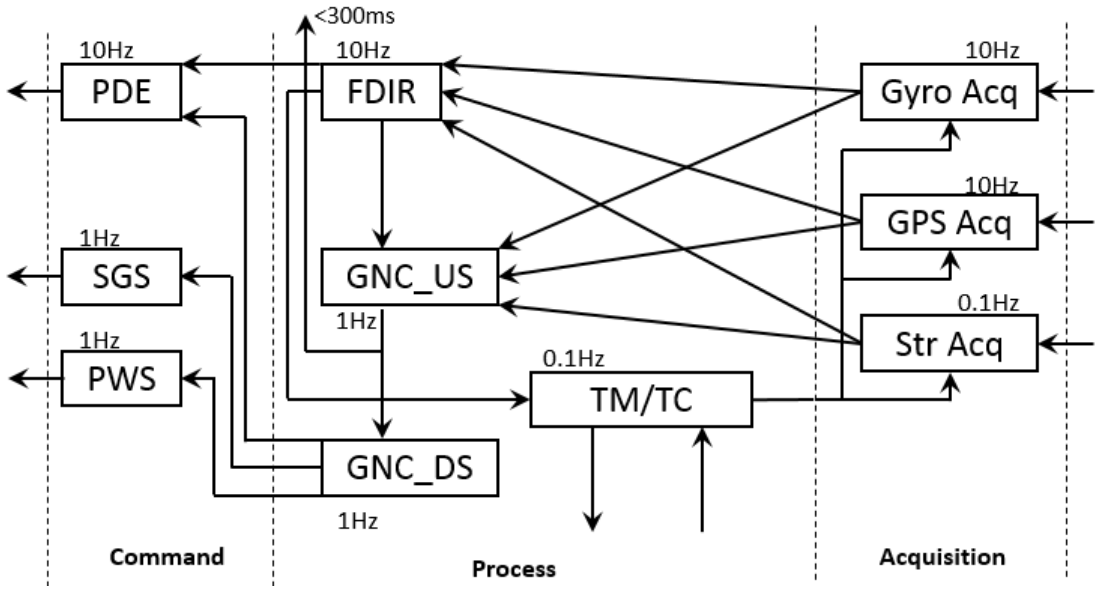

FIGURE 4.9 – FAS architecture

Chaque boîte représente une fonctionnalité de haut niveau (tâche d'un processus) et les arcs représentent les communications des données. Le système acquiert d'abord les données à l'aide des fonctions dédiées : l'orientation et la vitesse angulaire (Gyro Acq), la position (GPS Acq et *star-tracker* Str Acq) et la télécommande de la station sol (TM/TC). La fonction de navigation et de contrôle du guidage (divisée en GNC_US et GNC_DS) calculent les commandes à appliquer, tandis que la fonction FDIR (*Failure Detection Isolation and Recovery*) vérifie l'état du système FAS et recherche d'éventuelles défaillances. Les commandes sont envoyées au dispositif de contrôle : les commandes de poussée (PDE), les commandes de distribution de puissance (PWS), les commandes de positionnement du panneau solaire (SGS) et la télémétrie vers la station au sol (TM/TC). Les fréquences des tâches varient entre 0.1Hz et 10Hz. Les tâches de fréquences différentes peuvent communiquer les unes avec les autres. Il existe également une contrainte de délai intermédiaire qui exige que les données générées par GNC_US soient disponibles au plus tard 300ms après le début de leur période.

4.6.2 Modèle FAS en AADL

Le graphe de SPC pour FAS est présenté dans la figure 4.10. Il y a deux cycles sur le graphe de SPC, ce sont les cycles : $\tau_3 \rightarrow \tau_{10} \rightarrow \tau_9 \rightarrow \tau_3$ et $\tau_5 \rightarrow \tau_3 \rightarrow \tau_{10} \rightarrow \tau_9 \rightarrow \tau_5$.

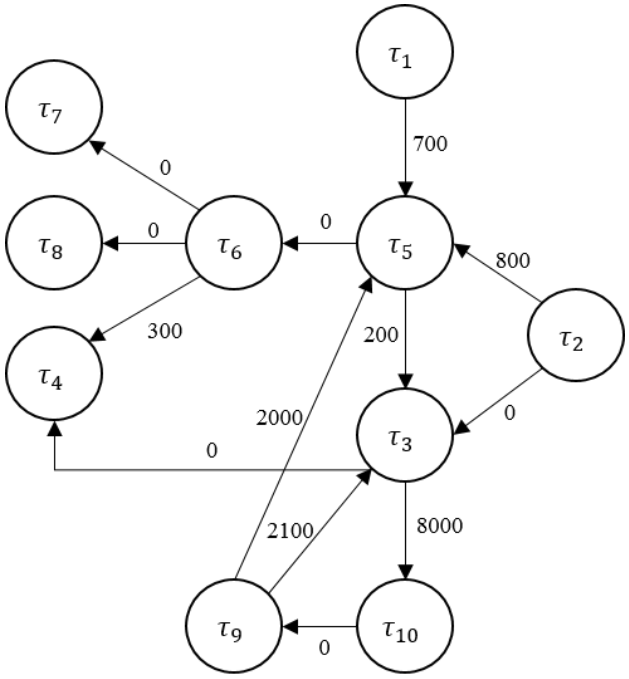

FIGURE 4.10 – Le graphe SPC avec cycles pour l'architecture FAS

Les propriétés de chaque tâche du FAS sont présentées dans le tableau 4.2.

id	tâche	date de réveil	échéance relative	pire temps d'exécution	période
τ_1	Gyro_Acq	10	100	10	100
τ_2	GPS_Acq	0	100	10	100
τ_3	FDIR	0	100	20	100
τ_4	PDE	0	100	10	100
τ_5	GNC_US	50	300	20	1000
τ_6	GNC_DS	0	1000	100	1000
τ_7	PWS	0	1000	20	1000
τ_8	SGS	0	1000	20	1000
τ_9	Str_Acq	1000	10000	200	10000
τ_{10}	TM/TC	500	10000	500	10000

TABLEAU 4.2 – Paramètres temporels des tâches de FAS

L'architecture du système FAS exprimée en AADL est présentée dans la figure 4.11 où chaque boîte est modélisée comme un *thread*. La communication entre les composants FAS se fait via des *data ports* utilisant notre extension SPC.

Pour des raisons de lisibilité seule une partie du graphe déplié des précédences du FAS est présentée sur la figure 4.12.

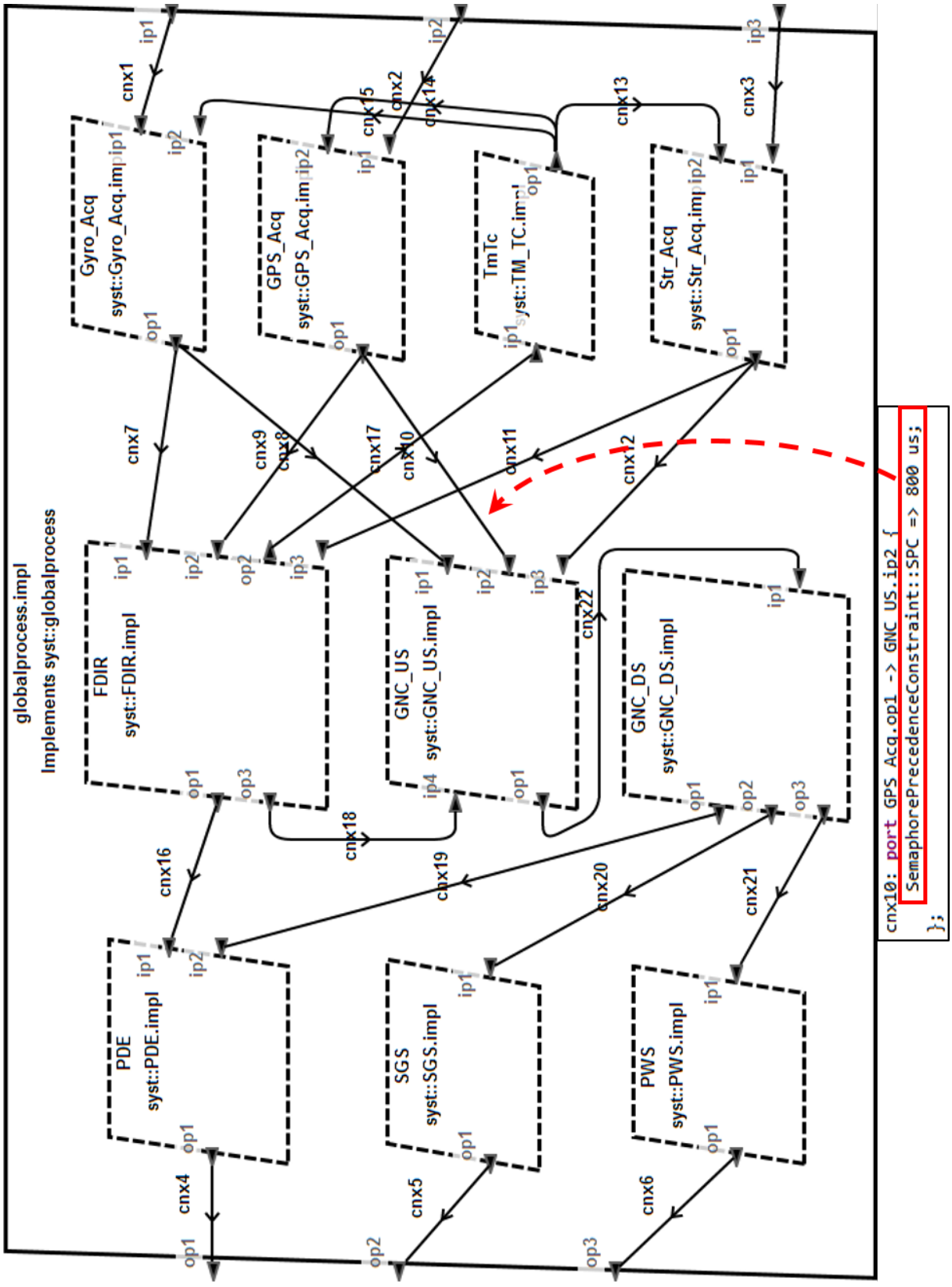

FIGURE 4.11 – Extrait du modèle FAS en AADL

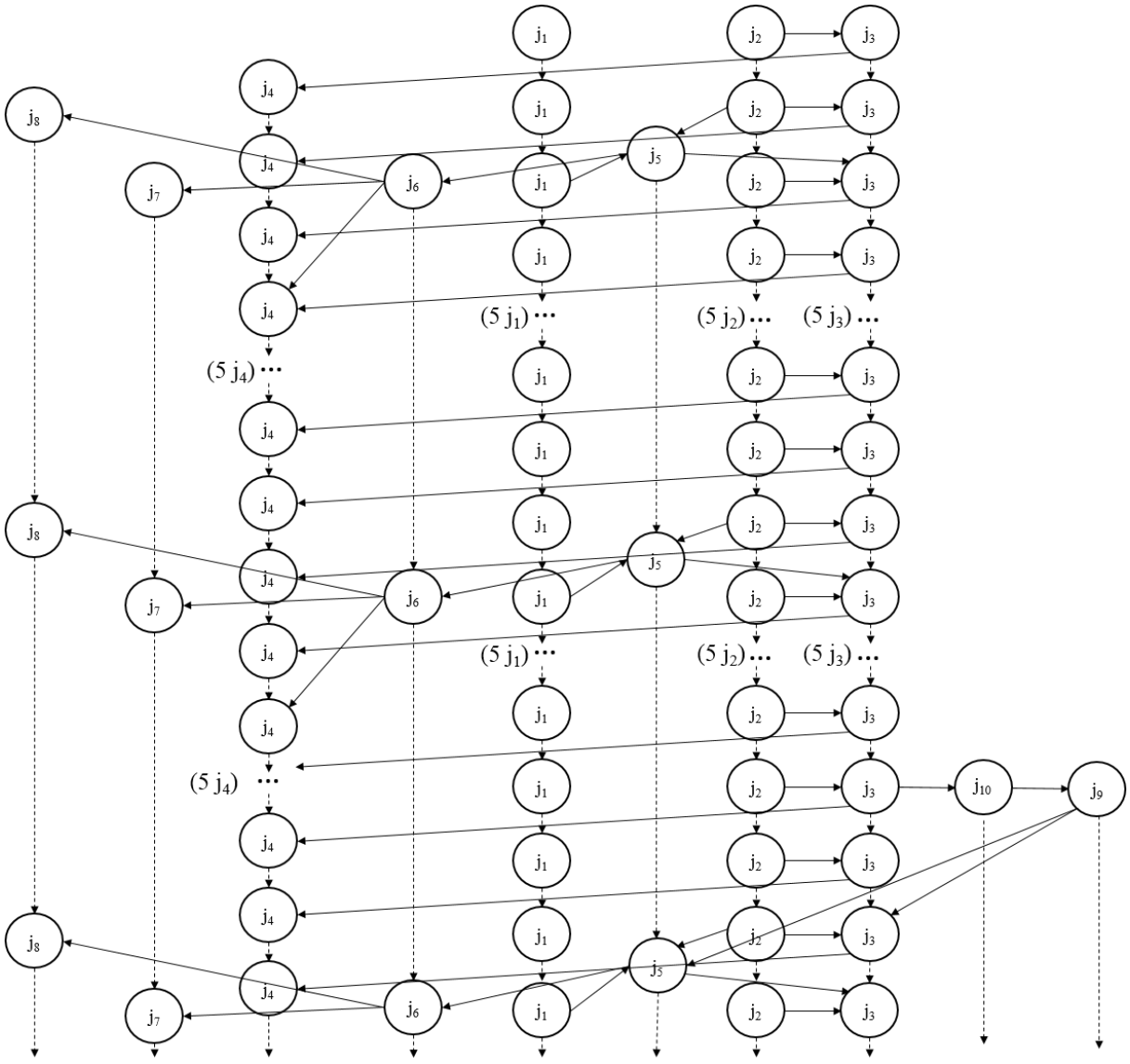

FIGURE 4.12 – Le graphe déplié du graphe de SPC du système FAS

4.7 Outillage

L'interface graphique de l'outil d'ordonnancement des tâches soumises à SPCs (*Semaphore Precedence Constraint*) est présentée dans la figure 4.13. Elle se compose d'une zone d'affichage des systèmes de tâches et les contraintes, d'une zone de contrôle, d'une zone d'analyses, d'une zone de résultat et d'une zone de simulation. L'outil supporte des modèles exprimés en AADL (i.e., **.aaxl2*) utilisant la propriété SPC et offre également la possibilité de saisir des systèmes sous un format XML propriétaire. La zone d'analyses offre trois possibilités : (i) analyser le système avec le test OPA [Aud91] et simuler l'exécution du système de tâches sous la configuration trouvée, (ii) analyser le système avec le test RBF [JS93b] et simuler l'exécution du système de tâches sous la politique d'ordonnancement EDF [Hor74, Der74], (iii) analyser le système avec le test RTA [JP86b] et simuler le système sous la politique d'ordonnancement DM [LW82]. Le résultat de la simulation est affiché dans la zone de simulation. La zone de résultat permet d'afficher les résultat des tests d'ordonnançabilité et propose de télécharger le résultat sous format spécifique selon le test utilisé.

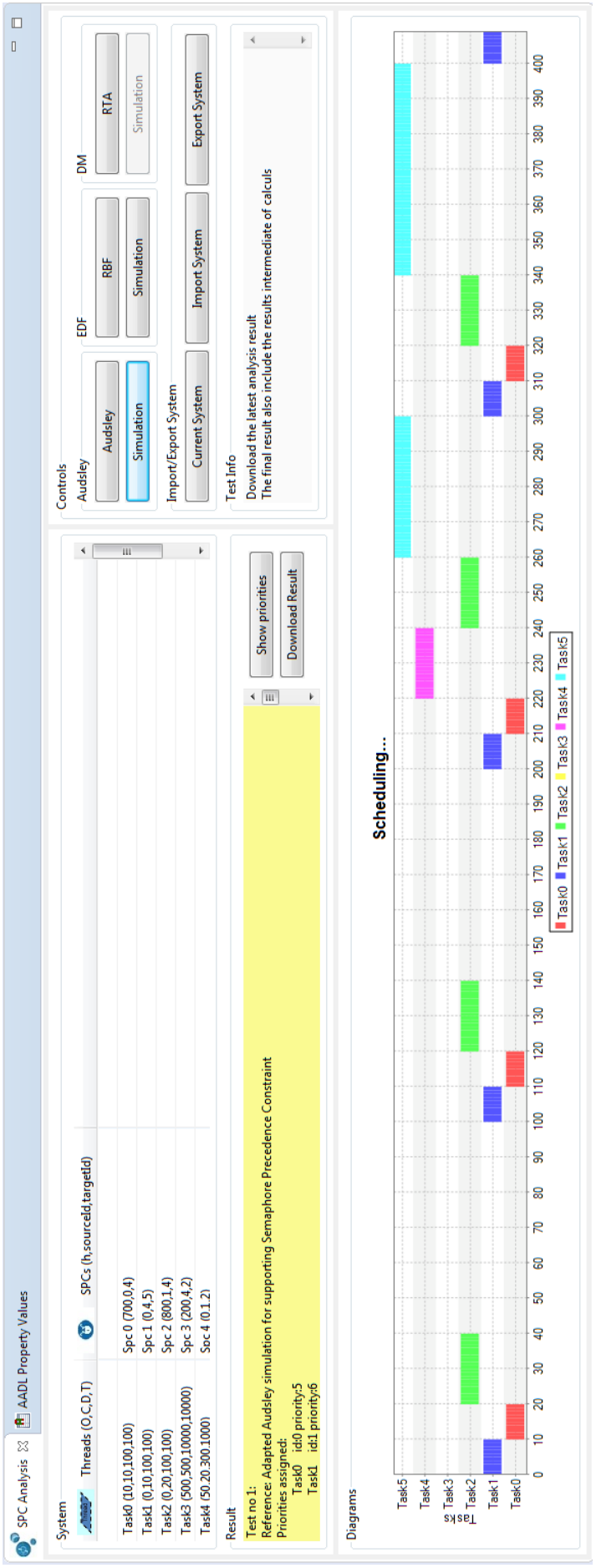

FIGURE 4.13 – L'interface graphique de l'outil d'ordonnancement des tâches soumises à SPCs

4.8 Conclusion

Dans ce chapitre, nous avons étendu les contrainte de précédence à base de sémaphore. Le comportement de ce type de contrainte dépend des périodes des communicants et de la valeur initiale du sémaphore à compte. Les contraintes peuvent être exprimées à l'aide du graphe de SPC. Notre extension permet d'exprimer des valeurs initiales négatives du sémaphore à compte, et permet d'avoir des cycles dans le graphe de SPC. Nous avons implémenté les SPC sous une propriété de *connection port* dans AADL et un outil d'analyse pour les systèmes de tâches soumises à des contraintes SPC.

Chapitre 5

Test exact d'ordonnançabilité de tâches dépendantes sous G-FP

Sommaire

Les méthodes d'analyse de temps de réponse, très répandues dans les outils d'analyse, se basent sur la prise en compte d'un instant critique non réaliste dès lors que certaines tâches sont dépendantes. De plus, l'analyse de l'ordonnançabilité de systèmes de tâches indépendantes à priorités fixes dans le cas de tâches dépendantes que nous avons pu trouver dans la littérature peut se montrer optimiste, ce que nous montrons dans ce chapitre. En utilisant une analyse exacte exhaustive des comportements des systèmes de tâches basée sur les réseaux de Petri temporels, notre objectif est double. D'une part, proposer une méthode efficace d'analyse exacte de système temps réel de tâches dépendantes en monoprocesseur, limité à des systèmes de petite taille, l'analyse requérant une durée exponentielle de la taille du modèle d'entrée. D'autre part, fournir une analyse exacte de calcul de pire temps de réponse pour des tâches dépendantes avec les algorithmes globaux à priorités fixes aux tâches sur systèmes multiprocesseurs.

5.1 Introduction

Dans le domaine industriel, domaine dans lequel le projet FUI WARUNA s'inscrit, la plupart des ordonnanceurs de systèmes d'exploitation temps réel sur étagère sont à priorités fixes aux tâches (*Fixed Priority* - FP) . C'est le cas par exemple des ordonnanceurs temps réel de base dans la norme POSIX 1003.1, ainsi que dans la norme AUTOSAR et Osek OS, ou dans les interfaces propriétaires de VxWorks, RTEMS, ou FreeRTOS. De plus, chacun de ces RTOS propose le protocole à priorité plafond dans sa version immédiate, ou le protocole à priorité héritée pour le cas particulier de VxWorks, afin d'éviter les inversions de priorité lorsqu'il existe des exclusions mutuelles. Dans les systèmes industriels, les systèmes de tâches indépendantes n'existent pas : elles sont souvent soumises à des exclusions mutuelles, et des contraintes de précédence. Or, les tests d'ordonnançabilité outillés et applicables au cas FP sont le plus souvent basés calcul de temps de réponse. Pour les tâches dépendantes, même en monoprocesseur, des hypothèses très conservatives sont émises afin de permettre aux tests d'avoir une complexité pseudo-polynomiale et ainsi de passer à l'échelle. Ainsi, l'impact des sections critiques de priorité inférieure sur des tâches de priorité supérieure est modélisée par le facteur de blocage B_i, et les contraintes de précédence sont modélisées par une gigue d'activation J_i représentant la date au plus tard à laquelle une tâche successeur peut être activée par sa tâche prédécesseur. Ainsi, si τ_i précède τ_j, la gigue d'activation de τ_j est le pire temps de réponse de τ_i. A partir de la gigue, lors du calcul de temps de réponse, on suppose que l'instant critique, que l'on pose à l'instant 0, utilisé pour étudier le pire temps de réponse d'une tâche τ_i est constitué par :

— La tâche τ_i s'est réveillée à l'instant critique avec sa gigue maximale, elle s'est donc activée à l'instant $-J_i$;

— Chaque tâche τ_j plus prioritaires est réveillée à l'instant critique avec sa gigue maximale, elle a donc été activée à la date $-J_j$, c'est-à-dire qu'on suppose que si elle a une tâche prédécesseur, celle-ci a eu son pire temps de réponse (bien qu'elle soit elle-même considérée comme se réveillant à l'instant critique) ;

— La tâche τ_i souffre de son pire facteur de blocage B_i. Cela correspond, dans le cas du protocole à priorité plafond, au fait que parmi toutes les sections critiques moins prioritaires sur des ressources critiques utilisées par une tâche au moins aussi prioritaire que τ_i, la section critique la plus longue vient juste de commencer. Dans le cas du protocole à priorités héritées, ce sont toutes les plus longues sections critiques de tâches moins prioritaire sur des ressources susceptibles d'être demandées par des tâches de priorité au moins égale à celle de τ_i qui viennent toutes juste de commencer.

Il est clair que ces conditions peuvent ne jamais être réunies simultanément, et par conséquent que le pire temps de réponse calculé peut être lourdement pessimiste. Ce pessimisme est le prix à payer pour avoir une complexité acceptable, cependant, pour l'étude d'un processeur avec relativement peu de tâches (de l'ordre de moins d'une dizaine), il peut être intéressant de faire un calcul exact de temps de réponse, même si la complexité spatiale ou temporelle de l'étude est exponentielle.

5.2 Exemple motivationnel

Afin d'illustrer ce point, considérons le système donné sur la Figure 5.1. Trois tâches sont ordonnancées par Rate Monotonic sur un processeur. τ_1, la plus prioritaire, est périodique, et active τ_3, qui partage une ressource critique avec τ_2, sporadique de période 12, la tâche la moins prioritaire. Notons qu'une communication par boîte aux lettres nécessite aussi que les tâches puissent accéder au contenu de la boîte aux lettres en exclusion mutuelle, ce qui implique une exclusion mutuelle entre τ_1 et τ_3. Nous supposons que chaque exclusion mutuelle de type tableau noir dure 0.5 unité de temps, alors que les exclusions mutuelles sur objets de type boite aux lettres dure 0.6 unités de temps.

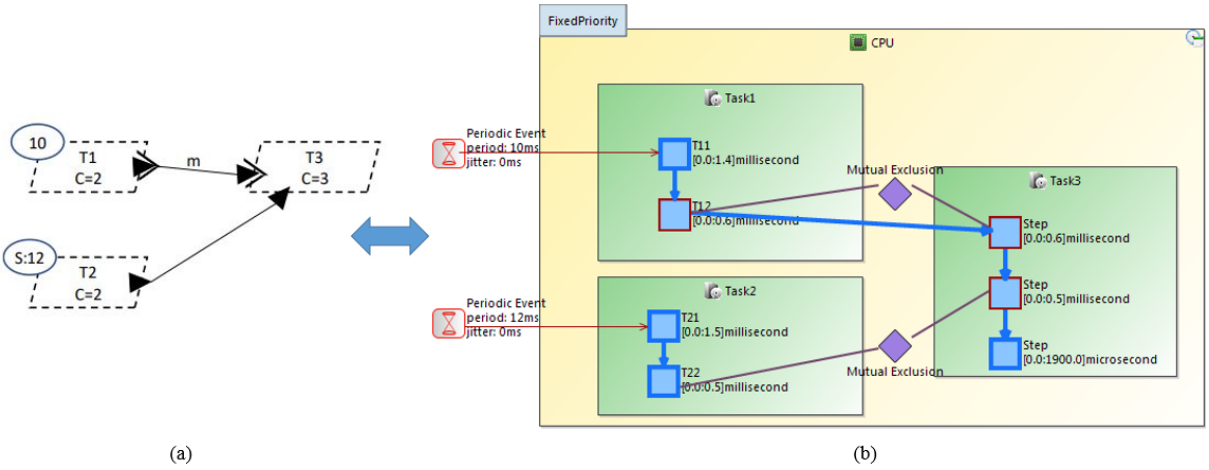

FIGURE 5.1 – Système exemple à étudier

Lorsque l'on utilise des outils de calcul de temps de réponse basés RTA, en supposant l'utilisation du protocole à priorité plafond, ceux-ci donnent pour τ_1 un pire temps de réponse de 2.6 (sa durée, plus la plus longue des sections critiques de priorité inférieure pouvant interférer avec elle), ce qui permet d'obtenir la gigue d'activation de τ_3, $J_3 = 2.6$. Le pire temps de réponse de τ_3 est alors obtenu en supposant un instant critique pendant lequel τ_3 est activée à la date -2.6, en même temps que τ_1 s'active, alors que τ_2 vient de commencer sa section critique sur le tableau noir. Cela donne donc une période d'activité du niveau de priorité de τ_3 qui dure 5.5 unités de temps, mais comme τ_3 s'est activée à la date de -2.6, cela lui donne un pire temps de réponse de 8.1. Rappelons cependant que ce temps de réponse est relatif à la date d'activation de la tâche τ_1. Enfin, l'instant critique de τ_2 est obtenu en supposant un réveil de τ_1 à l'instant 0, simultanément à un réveil de τ_3 à l'instant -2.6 avec son activation à l'instant 0. Cela permet de calculer un pire temps de réponse de 7.

Une modélisation par réseau de Petri (RdP) temporel (voir Figure 5.2 - l'unité de temps du RdP est le dixième d'unité de temps du système considéré afin de ne faire apparaître que des entiers sur les bornes d'intervalles de tir) est un moyen d'effectuer, à un coût exponentiel, une analyse exacte du système. Dans cette modélisation, toutes les durées possibles sont considérées entre le BCET (*Best Case Execution Time* - 0 par défaut) et le WCET de chaque partie de tâche, les activations d'une tâche sporadique sont aussi toutes considérées, de la période de la tâche à l'infini. Nous explicitons dans la suite de cette section les éléments de modélisation par réseau de Petri d'un système de tâche ordonnancé en FP ou en G-FP (*Global Fixed Priority* pour du multiprocesseur).

L'outil Romeo de modélisation et d'analyse de réseaux de Petri temporels paramétriques à inhibiteurs et *stopwatches* contient un *model checker* très puissant et très simple à intégrer à une suite d'outils. Il permet d'analyser simplement ce réseau avec une logique temporelle à la CTL permettant l'utilisation de variables (paramètres) et d'intervalles temporels. Il permet de démontrer que le pire temps de réponse effectif de τ_1 est en réalité de 2, ce qui montre qu'en aucun cas, τ_1 ne peut subir de durée de blocage due à la section critique partagée avec τ_3. Il montre aussi que le pire temps de réponse de τ_3 est en réalité de 3.5. Il faut comprendre que la RTA, lorsqu'on calcule le pire temps de réponse de τ_3, calcule le temps de réponse de la chaîne de tâches allant de l'activation de la première tâche à la terminaison de la tâche étudiée, i.e., le temps de réponse de la chaîne $\tau_1 \rightarrow \tau_3$. Le modèle RdP permet aussi de calculer le pire temps de réponse de cette chaîne, qui est en réalité de 5.5. Notons que dans le test RTA, la tâche τ_1 apparaît deux fois dans le pire temps de réponse de τ_3 : la première fois sous forme de gigue, la seconde sous forme de réveil à l'instant critique, ce qui ne peut pas arriver. Le calcul RTA du pire temps de réponse de τ_2, était, lui, exact, puisque son pire temps de réponse effectif est bien

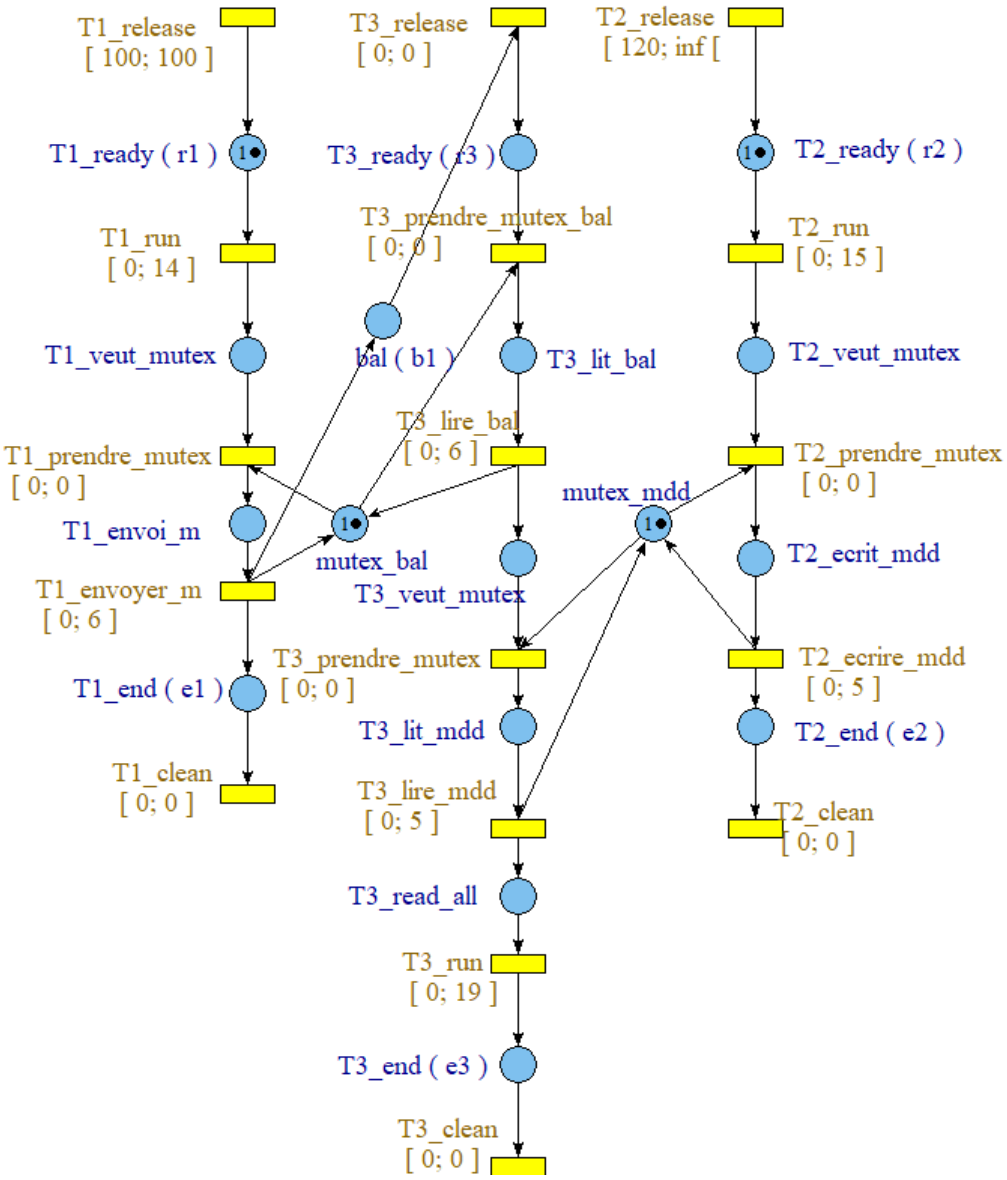

FIGURE 5.2 – Réseau de Petri temporel modélisant le système - arcs inhibiteurs à stopwatch non représentés

de 7 unités de temps.

5.3 Analyse exacte de temps de réponse sous ordonnanceur FP sur plate-forme monoprocesseur

5.3.1 Travaux connexes

Il existe de nombreuses études d'ordonnançabilité basées sur une représentation par RdP ou automate temporisé. T. Amnell et al. [AFM+02] ont proposé l'outil TIMES pour vérifier l'ordonnançabilité des systèmes monoprocesseurs. La construction de l'outil TIMES est basée sur des automates temporisés, plus précisément, sur le *model-checker* de UPPAAL [LPY97]. TIMES fournit un éditeur graphique permettant de spécifier un ensemble de tâches avec leurs paramètres tels que la priorité, le pire temps d'exécution ou l'échéance, etc. TIMES fournit également un simulateur pour que l'utilisateur puisse observer le comportement dynamique du système selon ses paramètres et l'ordonnanceur choisi. Un vérificateur est aussi fourni par TIMES permettant des analyses exactes sur des ensembles de tâches synchrones, périodiques ou sporadiques. Cependant, TIMES présente quelques points faibles : il ne prend pas en compte les dates de réveil différentes des tâches (tâches différées) et ne permet pas d'analyser les ensembles de tâches avec la présence de ressources partagées. E. Grolleau et A. Choquet-Geniet ont proposé dans [GCG00b] de modéliser des systèmes de tâches sur monoprocesseur à l'aide de RdP coloré [Jen81] sous la règle de tir maximal. Les points forts de leurs travaux sont qu'ils permettent de prendre en compte les ressources partagées, les contraintes de précédence, et les tâches différées. Le point faible de leurs travaux est que le RdP génère une séquence hors-ligne à exécuter, et ne permet pas d'utiliser un ordonnanceur en-ligne.

Enfin, on peut citer G. Behrmann et al. qui ont proposé dans [BLR05] un modèle d'automate temporisé appelé *Priced Timed Automata* en associant à chaque transition et chaque location respectivement un coût et un taux de coût. L'analyse consiste à chercher un ordonnancement optimal hors-ligne de coût minimal à l'aide de *model-checker* de UPPAAL.

L'avantage des réseaux de Petri temporels est qu'ils permettent l'étude d'un comportement en-ligne de système, en assurant la C-viabilité (toutes les durées possibles de tâches sont considérées), et la T-viabilité (pour les sporadiques, toutes les périodes, de la période minimale à l'infini, sont considérées). De plus, le temps est considéré comme continu, ce qui évite d'être optimiste, contrairement à ce que nous allons montrer dans le cas des travaux de T.P. Baker et M. Cirinei [BC07].

5.3.2 Modélisation existante avec observateurs complexes

L'idée d'utiliser réseau de Petri temporel dans un contexte industriel s'inspire des travaux de B. Parquier et al. dans [PRH+16]. L'idée est de modéliser un système de tâches périodiques sur architecture monoprocesseur à l'aide de réseau de Petri temporel avec arcs inhibiteurs à stopwatch. La figure 5.3 représente les règles de transition du système en réseau de Petri temporel tel qu'ils les ont proposées.

L'une des limites de ce modèle est qu'il ne peut pas exprimer la première période de chaque tâche, autrement dit, selon le modèle traduit, l'exécution de chaque tâche ne commence pas immédiatement mais à la deuxième itération, ce fait est illustré par la figure 5.4.

Pour calculer le pire temps de réponse des tâches, un observateur est utilisé. L'observateur est un autre réseau de Petri lié au modèle initial permettant d'observer quelques propriétés désirées. Il n'impacte pas le comportement du modèle initial et grâce à la bonne propriété demandée au *model-checker*, on peut obtenir les pire temps de réponse des tâches observées.

Le processus de calcul du temps de réponse dépend du nombre maximal de réentrances (i.e., combien d'activations une chaîne étudiée peut avoir à un moment). C'est pour cela que deux observateurs sont utilisés. Le premier observateur consiste à compter le nombre maximal de réentrances, il nécessite le lancement du model-checker pour analyse. On obtient alors le nombre maximal de réentrances, qui permet de construire le second observateur, dont la profondeur

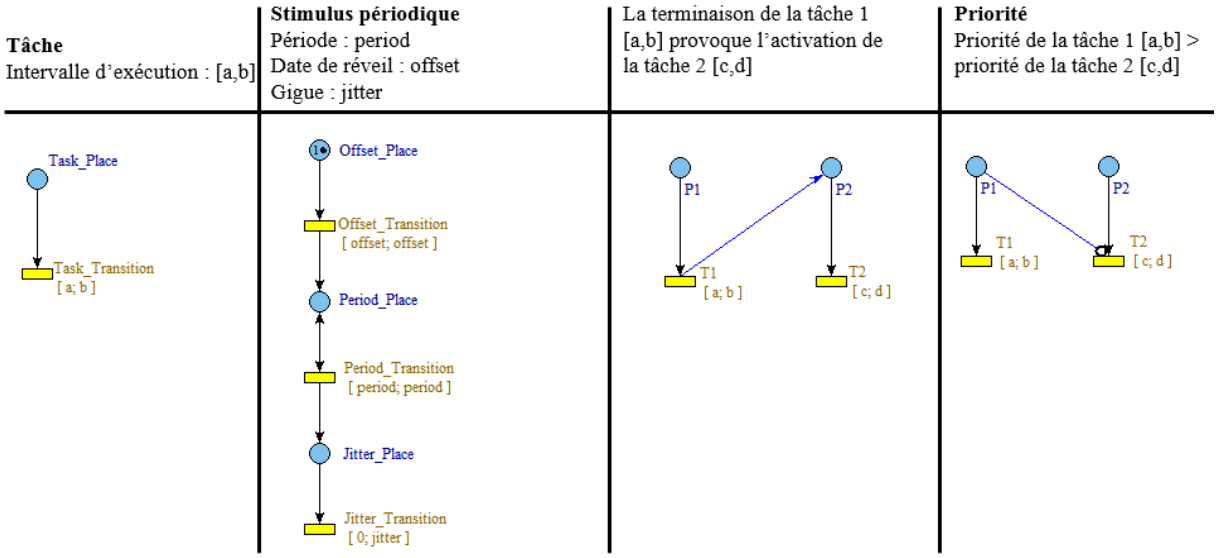

FIGURE 5.3 – Traduction du système en RdP temporel [PRH+16]

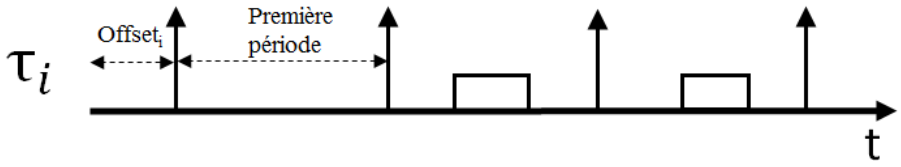

FIGURE 5.4 – Exécution du modèle traduit

dépend du nombre de réentrances, qui va permettre lors d'une seconde analyse de déduire le pire temps de réponse d'une tâche ou d'une chaîne de tâches. Le premier observateur se compose de deux places et une transition. Les deux places se relient aux transitions $Jitter\_Transition$ et $Task\_Transition$, tel que présenté sur la figure 5.5. Les arcs entrant sur $P10$ et $P11$ marquent

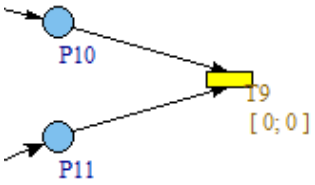

FIGURE 5.5 – Structure de l'observateur de réentrance

ces places respectivement lorsque la première tâche de la chaîne observée s'active, et lorsque la dernière tâche de la chaîne observée se termine. Dès que les deux places sont marquées, $T9$ est tirée. Par construction, le marquage maximal atteint par $P10$ donne le nombre maximal de chaînes démarrées simultanément.

Pour calculer le nombre maximal de jetons présents au même moment dans la place $P10$, il faut demander *model-checker* si la formule ParamTPN-PTCTL 5.1 est vraie pour un entier nb donné. Cette formule exprime le fait que sur tout comportement possible du RdP, sur un horizon temporel infini, le marquage de $P10$ est borné par nb. Malheureusement, il faut proposer des valeurs pour nb, ce qui amène à appeler pour chaque valeur testée le model-checker, par exemple en faisant croître nb aux valeurs 1, 2, 3, 4, etc. jusqu'à obtenir une réponse positive permettant de trouver le marquage maximal nb_{max} de cette place. Le nombre de réentrances est nb_{max}-1. Notons que ce processus peut prendre du temps, la complexité de chaque analyse ParamTPN-PTCTL n'étant pas polynomiale.

$$AG[0, inf]P10 \leq nb \tag{5.1}$$

L'observateur permettant de calculer un temps de réponse utilise un paramètre, c'est à dire une variable, que le model-checker de Roméo va chercher pour que la formule soit vraie. Nous allons présenter le cas simple où il n'y a pas réentrance. La structure de l'observateur de temps de réponse dans ce cas est présentée dans la figure 5.6. Cet observateur est relié aux modèle initial par les mêmes transitions que l'observateur de réentrances.

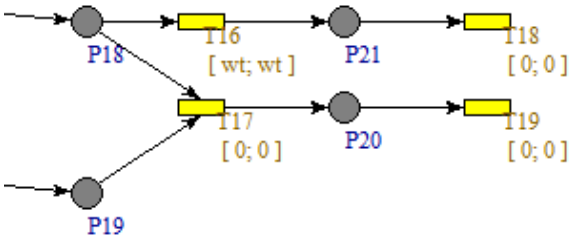

FIGURE 5.6 – Structure de l'observateur de temps de réponse sans réentrance

Pour obtenir le pire temps de réponse, nous demandons au *model-checker* la formule ParamTPN-PTCTL 5.2. Cette formule recherche le domaine de valeurs du paramètre wt telles qu'aucun jeton ne puisse franchir la transition $T16$. Quand la première tâche de la chaîne étudiée commence son exécution, un jeton est envoyé à la place $P18$, la transition $T16$ est sensibilisée, donc l'horloge associée à la transition $T16$ commence à compter. Quand la dernière tâche de la chaîne étudiée finit son exécution, un jeton est envoyé à la place $P19$, la transition $T17$ est sensibilisée. La transition $T17$ est immédiatement franchie, donc $T16$ n'est plus sensibilisée, l'horloge associée s'arrête au temps d'écart entre l'instant où la première tâche commence son exécution et l'instant où la dernière tâche termine son exécution, i.e. le temps de réponse de la chaîne étudiée.

$$AG[0, inf]M(P21) < 1 \tag{5.2}$$

5.3.3 Modélisation proposée

Nous allons poser quelques hypothèses sur le comportement des applications modélisées. Nous supposons, comme dans le profil Ravenscar [Bur99] que les tâches en attente de message commencent par attendre un message. On ne peut donc pas trouver d'attente de message provenant d'une boîte aux lettres en milieu de tâche par exemple, ou à l'intérieur d'une section critique. Nous supposons des priorités fixes aux tâches, affectées de manière unique aux tâches, i.e. les tâches possèdent toutes une priorité distincte. Les priorités d'une tâche peuvent varier par application d'un protocole de gestion de ressources, en particulier nous considérerons le protocole à priorité plafond dans sa version immédiate, qui est le plus répandu dans les RTOS.

5.3.3.1 Modélisation des rythmes d'activation

Dans la colonne de gauche de la figure 5.7, nous voyons que le modèle proposé reprend le principe proposé dans [PRH+16]. Une transition de durée comprise entre le BCET (par défaut 0) et le WCET représente une tâche (ou une portion de tâche comme nous le verrons par la suite). De même, colonne de droite, nous voyons que le déclenchement d'une tâche par une autre prend la modélisation classique d'une synchronisation dans un RdP.

Une première différence vient de la modélisation des réveils. Les horlogeries d'activation des tâches périodiques, qu'elles soient à offset nul ou non nul, sont représentées dans la seconde colonne de la figure. A la date d'offset (qui est nulle si la date de réveil est nulle), la transition *Offset_Transition* est tirée, créant un jeton dans *First_Release_Place*, qui est marquée à la

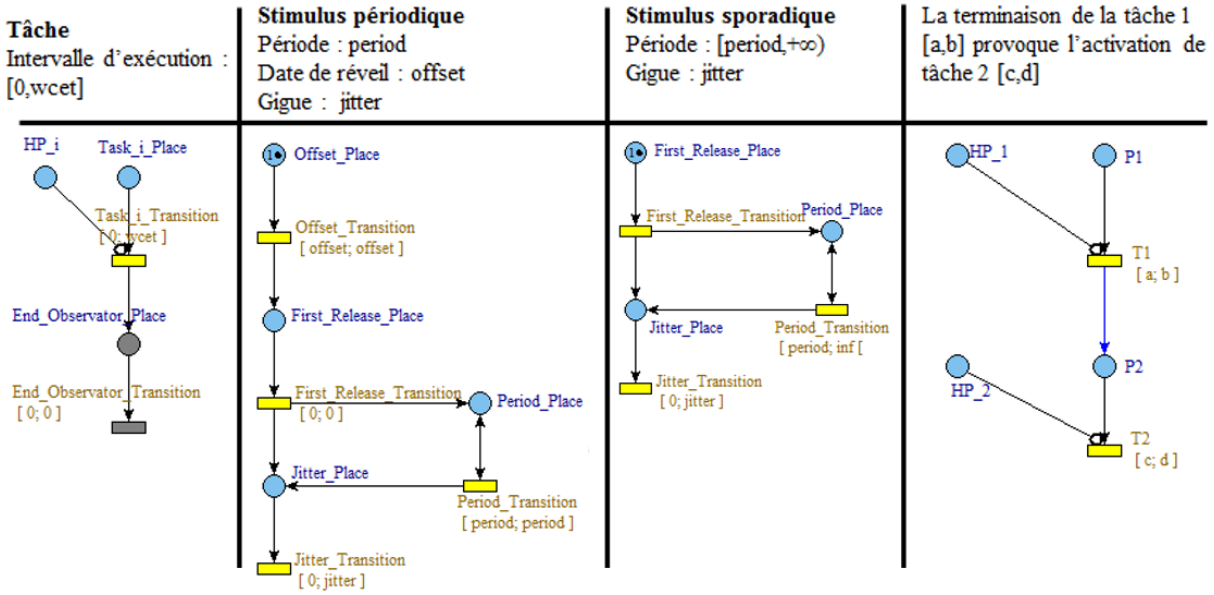

FIGURE 5.7 – Règles de construction du système monoprocesseur en RdP

première activation de la tâche, ce qui permet de franchir immédiatement la transition *First_Release_Transition*, qui va marquer deux places : *Jitter_Place* qui permet de représenter le délais dû à la gigue d'activation (qui peut être nulle), ainsi que la place *Period_Place* qui va permettre, périodiquement, d'activer à nouveau la tâche en marquant *Jitter_Place* par le tir de *Period_Transition*. Ce faisant, la place *Period_Place* est marquée de nouveau, permettant un prochain réveil une période plus tard. Une place *End_Observator_Place* et une transition *End_Observator_Transition* sont aussi ajoutées en fin de tâche (marquées en gris dans la première colonne de la figures 5.7). Cet observateur permet de marquer la date de fin de la tâche. La plus grande différence entre la date d'activation et la date de fin donne le pire temps de réponse de la tâche.

Les tâches sporadiques ($3^{ième}$ colonne) fonctionnent suivant le même principe, excepté qu'elles n'ont pas de date de réveil, celle-ci étant remplacée par une transition non contrainte *First_Release_Transition* (l'intervalle par défaut est de 0 à l'infini), permettant d'activer la première instance à n'importe quel moment de la vie de l'application. Ensuite, la période (dans ce cas, le délai minimal inter-activation) est représentée par la transition *Period_Transition* qui, dans le cas sporadique, a un intervalle de tir statique [Période,+∞[.

5.3.3.2 Modélisation de la concurrence et des priorités

Si l'on reprenait le principe de la modélisation de la concurrence proposé dans [PRH+16], étant donné que nous considérons des tâches dépendantes, intégrant des exclusions mutuelles qui nous obligeront à représenter chaque tâche par plusieurs transitions, il faudrait de nombreux arcs inhibiteurs à stopwatch, de chaque place précédant une transition d'exécution d'une tâche plus prioritaire à chaque transition d'exécution de toutes les tâches moins prioritaires. Le modèle serait totalement illisible. L'idée ici, qui permettra en plus de passer très facilement à l'ordonnancement multicœur global, est plutôt de munir chaque tâche d'une place HP_i dans laquelle chaque tâche qui s'active et est plus prioritaire que la tâche va produire un jeton, qu'elle retirera à sa terminaison. Pour une tâche τ_i, il suffira alors de mettre un inhibiteur à stopwatch de la place HP_i à chacune de ses transitions d'exécutions. Nous verrons que nous altérerons un peu ce fonctionnement pour la prise en compte de l'héritage provoqué par le protocole de gestion de ressources.

Deux principes de modélisation de la concurrence et des priorités sont illustrés à travers un exemple montré dans la figure 5.8. Dans l'exemple, il y a trois tâches τ_i, τ_j et τ_k et leurs

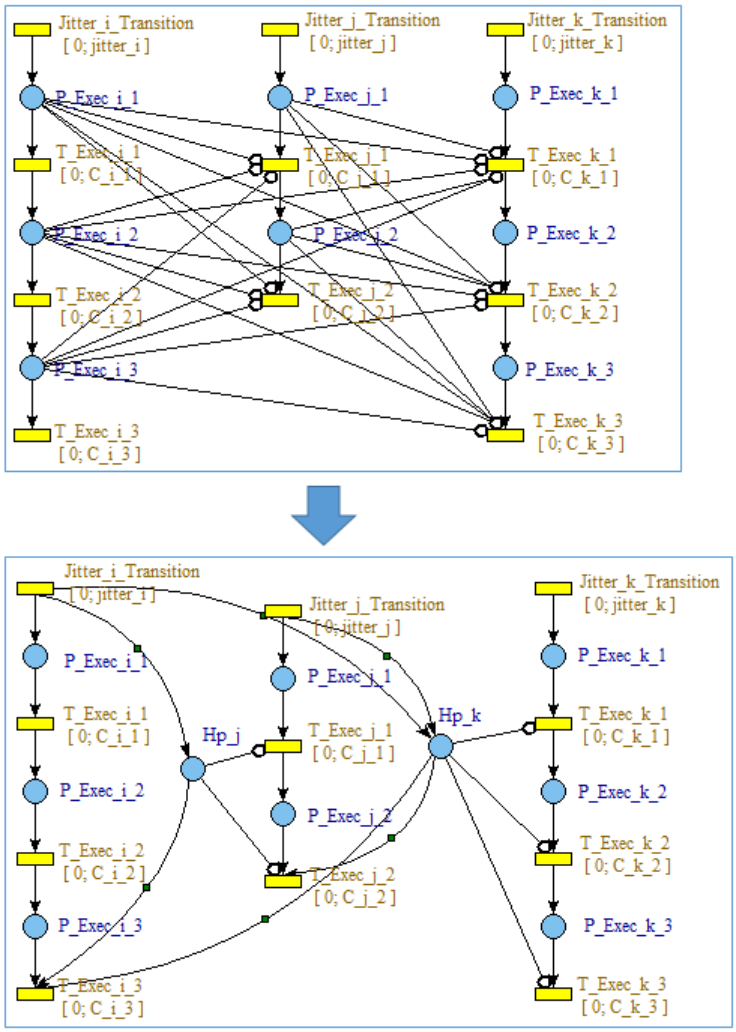

FIGURE 5.8 – Concurrence et priorités sur une plateforme monoprocesseur

priorités : prio(τ_i)>prio(τ_j)>prio(τ_k). Chaque tâche se compose de plusieurs sections (celles-ci représentent des parties de la tâche, séparées typiquement par des points de synchronisation comme des exclusions mutuelles par exemple) : la tâche τ_i comprend trois sections (Exec_i_1, Exec_i_2, Exec_i_3), la tâche τ_j comprend deux sections (Exec_j_1, Exec_j_2) et τ_k comprend trois sections (Exec_k_1, Exec_k_2, Exec_k_3) et pour simplifier, nous cachons les stimuli d'activation des tâches. Noter que τ_i est la tâche qui sera toute la vie de l'application la plus prioritaire donc il n'est pas utile de présenter la place HP_i qui serait toujours vide, puisqu'elle contient le cardinal de l'ensemble des tâches plus prioritaires que la tâche τ_i.

5.3.3.3 Modélisation des exclusions mutuelles

Dans cette partie, nous allons présenter la modélisation des exclusions mutuelles. Nous commençons en modélisant les exclusions mutuelles de façon classique. Pour faciliter la compréhension, nous présentons un exemple simple illustrant le principe. Le système se compose de deux tâches : τ_1, τ_2 partageant une ressource critique.

La figure 5.9 représente la modélisation des tâches partageant des ressources critiques. Les tâches sont modélisées en trois étapes : (1) demande d'accès aux ressources critiques, (2) prise du sémaphore d'exclusion mutuelle et exécution de la section critique, une fois qu'une tâche a obtenu l'accès, d'autres tâches partageant la même ressource sont bloquées et (3) finit l'exécution et libère les ressources.

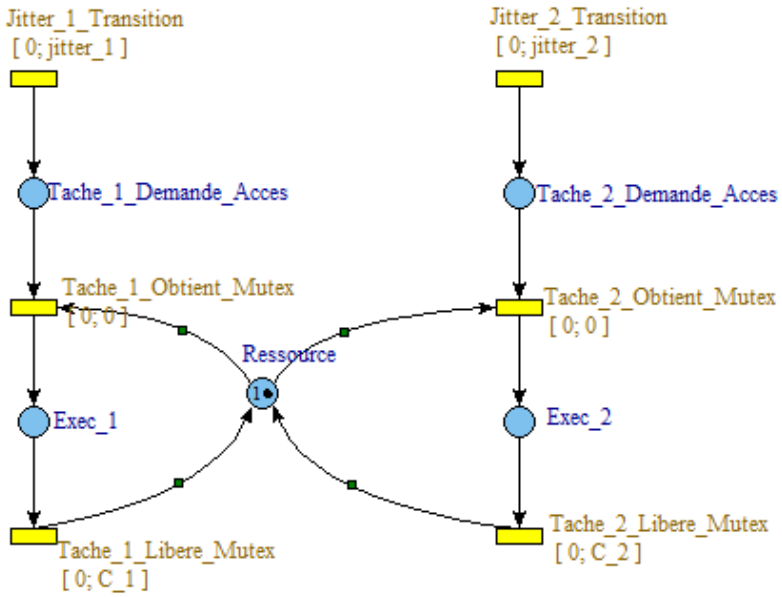

FIGURE 5.9 – Modélisation des exclusions mutuelles sans protocole de gestion

Dans les systèmes temps réel, afin d'éviter les inversions de priorité, on utilise des protocoles de gestion ressources. Parmi les protocoles de gestion de ressources, le protocole à priorité plafond immédiat est très utilisé, et proposé par la plupart des RTOS sur étagère. Dans la suite, nous allons proposer la modélisation des exclusions mutuelles sous le protocole à priorité plafond immédiat. La figure 5.10 présente un exemple illustrant le principe du protocole à priorité plafond immédiat. Le système a deux tâches τ_1, τ_2, la tâche τ_1 est plus prioritaire que la tâche τ_2, ces deux tâches partagent une ressource critique. La tâche τ_2 entrant dans la section critique hérite pendant toute la section critique de la priorité plafond de la ressource.

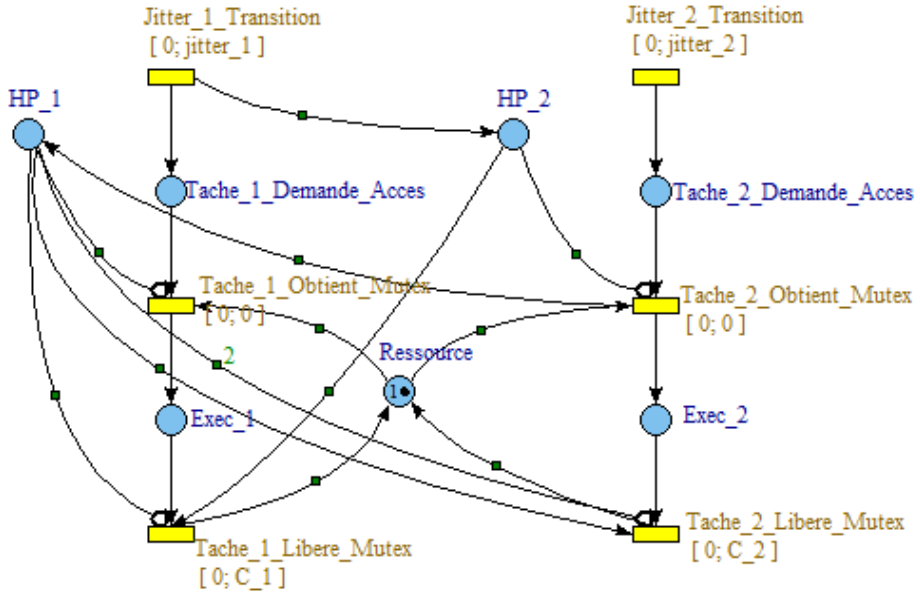

FIGURE 5.10 – Modélisation des exclusions mutuelles sous le protocole à priorité plafond immédiat

5.3.3.4 Exemple

Dans cette partie, nous allons reprendre l'exemple de la figure 5.2 en appliquant le principe de modélisation des ressources critiques sous protocole de priorité plafond immédiat présenté dans les parties précédentes.

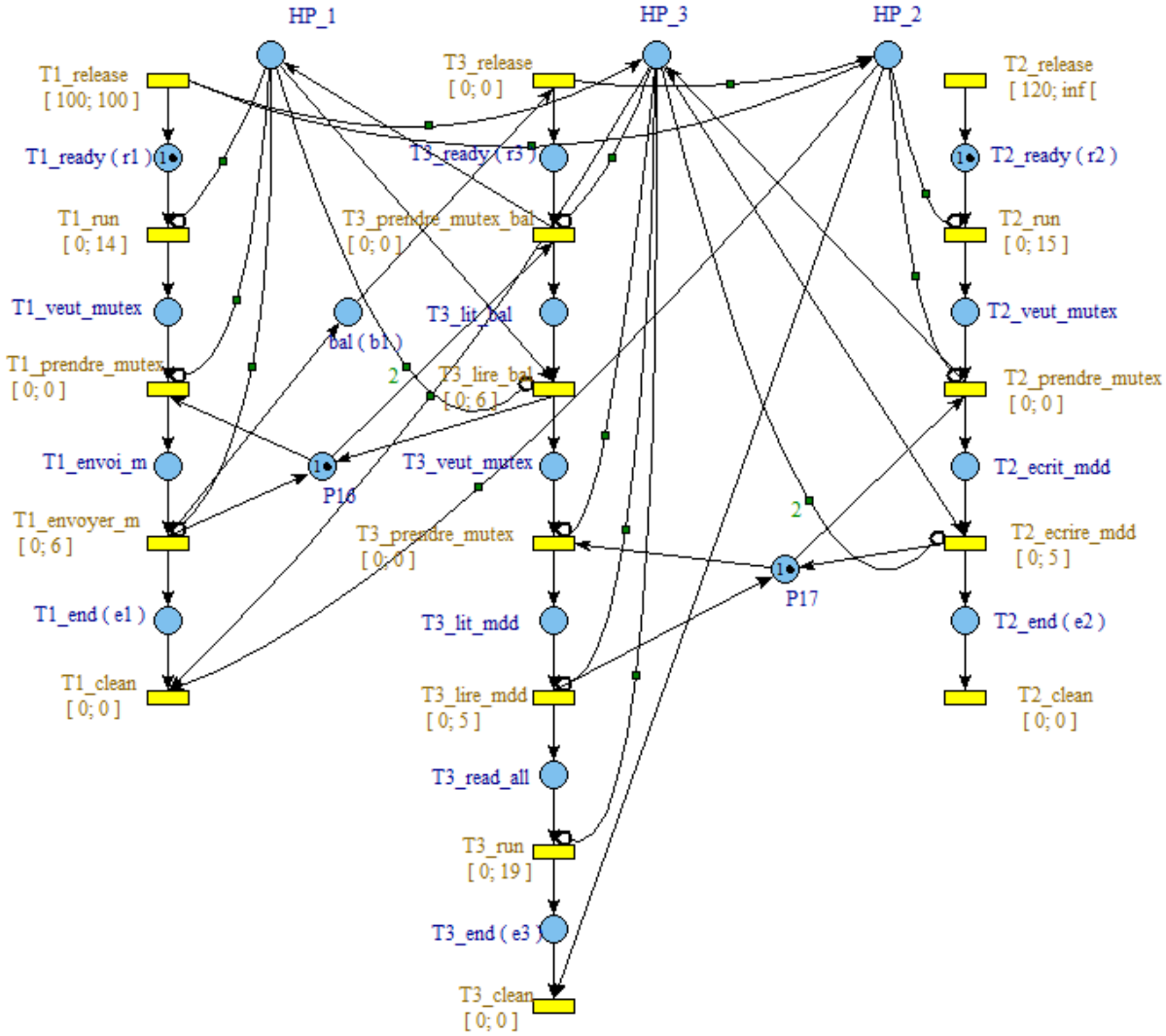

FIGURE 5.11 – Réseau de Petri temporel avec arcs inhibiteurs à stopwatch

5.4 Analyse exacte de temps de réponse sous ordonnanceur G-FP sur plate-forme multiprocesseur identique

Les systèmes multiprocesseurs attirent récemment l'attention de la communauté d'analyse, à cause de la fin de la loi de Moore, les transistors ayant atteint la taille linéaire de quelques atomes. Par conséquent, l'ordonnancement temps réel et l'analyse d'ordonnançabilité des systèmes multiprocesseurs deviennent un domaine de recherche important. Dans ce chapitre, nous ciblons l'ordonnançabilité des systèmes de tâches dépendantes sur architecture multiprocesseur identique.

5.4.1 Travaux connexes

T.P. Baker et M. Cirinei ont proposé dans [BC07] une analyse d'ordonnançabilité exhaustive d'un système de tâches sporadiques par un algorithme global à priorités fixes aux tâches sur multiprocesseur identique. Cette approche consiste à construire une machine à états finis représentant toutes les combinaisons possibles entre les dates de réveil et les séquences d'exécution des tâches. Dans leurs travaux, ils considèrent un temps discret, ainsi, une tâche sporadique de période T se verra attribuer une date de réveil à T, $T+1$, $T+2$, etc. jusqu'à atteindre un espace d'état qui boucle. Cette approche fait l'hypothèse que le temps est discret, et souffre d'explosion combinatoire. L'aspect discret ou non du temps en ordonnancement a été assez peu abordé, car en monoprocesseur, dans la plupart des problèmes d'ordonnancement, le fait qu'il soit continu ne nuit pas aux méthodes proposées sur du temps discret. Cependant, nous verrons ici que cela n'est pas vrai dans le cas multiprocesseur. On peut bien entendu, le cycle machine se ramenant finalement à un temps discret, supposer que le temps est discrétisé au grain d'un cycle machine, cependant, cela augmente d'autant plus les durées des tâches, et donc l'espace d'états à construire dans la méthode de [BC07], la rendant encore moins susceptible d'être utilisée sur des cas réels. On peut prendre un exemple pour montrer la différence entre le temps discret et le temps continu : soit un système de tâches $S = \{\tau_i(C,T,P)\} = \{\tau_1(1,4,4), \tau_2(1,4,3), \tau_3(1,4,2), \tau_4(1,4,1)\}$ (où P représente la priorité d'une tâche) s'exécutant sur deux processeurs.

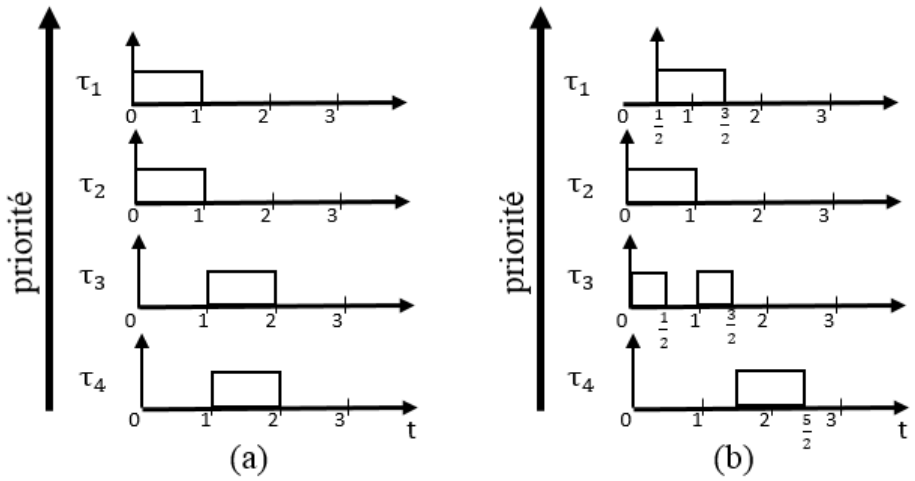

FIGURE 5.12 – Temps discret vs temps continu

La figure 5.12 présente l'ordonnancement du système de tâches dans deux cas : temps discret (a) et temps continu (b). Nous voyons que le pire temps de réponse de la tâche 4 dans le cas de temps discret est 2, cependant il peut atteindre 5/2 dans le cas de temps continu, cas se produisant lorsque la tâche τ_1 est activée à l'instant $\frac{1}{2}$. Par conséquent, mis à part si le modèle de temps considéré tombe à la granularité du cycle machine, la méthode proposée dans [BC07], qui ne considère que des dates de réveil entières, peut se montrer optimiste. Cependant, si chaque durée de tâche est exprimée sur son nombre de cycles, l'explosion combinatoire rend cette méthode totalement inutilisable.

N. Guan et al. ont proposé dans [GGD+07] un test exact pour les systèmes de tâches synchrones périodiques sous G-FP en utilisant un automate temporisé [Alu99]. Le temps considéré est discret et peut donc lui aussi se montrer optimiste si la granularité n'est pas celle du cycle machine. Enfin, G. Geeraerts et al. ont amélioré les travaux de T.P. Baker et M. Cirinei dans [GGL13] en utilisant une technique d'antichain. En fait, ils ont proposé une relation de simulation permettant de détecter les états d'erreur depuis d'autres états d'erreur. Cela permet de restreindre les états à tester. Y. Sun et G. Lipari a attaqué le problème d'ordonnançabilité des systèmes de tâches sporadiques sous G-FP sur une architecture multiprocesseur identique en

proposant dans [SL16] un *Linear Hybrid Automaton* (LHA) basé sur un automate temporisé à chronomètre (i.e., *Stopwatch Timed Automata*). Dans leurs travaux, ils ne considèrent cependant que les tâches indépendantes.

Dans cette thèse, nous analysons l'ordonnançabilité des systèmes de tâches sporadiques dépendantes sur architecture multiprocesseur identique à l'aide de réseaux de Petri temporels. Pour la modélisation des tâches, nous verrons qu'il s'agit d'une simple extension du modèle monprocesseur proposé, sauf que les arcs inhibiteurs à stopwatch sont pondérés par m, le nombre de cœurs. En effet, en multiprocesseur global, m tâches au plus sont exécutées simultanément sur m cœurs, ce qui fait que si pour une tâche, il existe au moins m tâches plus prioritaires actives, elle ne peut pas s'exécuter. Cependant, il y a une différence dans la gestion des exclusions mutuelles, qui nous amène à faire un petit état de l'art sur la gestion de ressources en environnement multiprocesseur.

5.4.2 Modélisation des ordonnancements G-FP sur plate-forme multiprocesseur identique

5.4.2.1 Cas des exclusions mutuelles

Les systèmes multiprocesseurs sont de plus en plus répandus dans la pratique et nécessitent des échanges des données entre les tâches, cela pose le problème de gestion des ressources critiques sur une architecture multiprocesseur. La littérature propose plusieurs approches pour faire face au problème.

R. Rajkumar a proposé dans [Raj12] le protocole MPCP (*Multiprocessor Priority Ceiling Protocol*) en étendant le protocole à priorité plafond (i.e., *Priority Ceiling Protocol* [SRL90]). MPCP est utilisé pour la synchronisation d'un ensemble de tâches partitionnées partageant des ressources critiques et ordonnancées en priorités fixes aux tâches (i.e., *Partitioned Priority Fixed*). Sous MPCP, les ressources sont classifiées en ressources locales et ressources globales. Les ressources locales sont partagées par des tâches sur un processeur, les ressources globales sont partagées par les tâches de processeurs différents. La priorité d'une tâche dans une section critique globale, dans laquelle elle demande une ressource globale, est affectée une priorité supérieure à la priorité la plus élevée parmi toutes les tâches locales. Cette priorité est appelé **plafond distant** (i.e., *Remote Ceiling*). Or, une section critique globale ne peut être préemptée que par d'autres sections critiques globales qui ont le plafond distant plus élevé. Les ressources locales sont protégées en utilisant les protocoles de synchronisation monoprocesseur (e.g., PCP).

Gai et al. ont proposé dans [GLDN01] le protocole MSRP (*Multiprocessor Stack Resource Policy*) en étendant le protocole d'allocation de la pile (*Stack Resource Policy*-SRP). MSRP est utilisé pour synchroniser un ensemble de tâches partageant des ressources critiques sous EDF partitionné (P-EDF). Comme MPCP, les ressources sont aussi classifiées en locales ou globales. Les tâches sont synchronisées sur les ressources locales en utilisant SRP. Une différence significative entre MSRP et MPCP est que lorsqu'une tâche est bloquée sur une ressource globale sous MSRP, elle effectue un *busy wait* (c-à-d le processeur est occupé sans aucun travail) et n'est pas préemptive. Les tâches bloquées sur une ressource globale sont ajoutées à une file d'attente FIFO. Les sections critiques globales ne peuvent pas être imbriquées sous MSRP.

Block et al. ont proposé dans [BLBA07] le protocole FMLP (*Flexible Multiprocessor Locking Protocol*). FMLP peut être appliqué à toutes les politiques d'ordonnancement quelque soit partitionnées ou globales (e.g., P-EDF ou G-EDF). Dans FMLP, les ressources sont classifiées en ressources courtes et ressources longues selon l'utilisateur. La demande d'accès aux ressources longues ne peut pas être imbriquée dans la demande d'accès aux ressources courtes. FMLP peut éviter l'impasse en regroupant ressources qui peuvent être imbriquées et garantissant qu'un seul travail peut accéder aux ressources d'un groupe à tout moment. Un groupe inclut des ressources, deux ressources sont dans le même groupe si une demande pour l'une est imbriquée dans une demande pour l'autre. Un verrou est attribué à chaque groupe et une seule tâche peut contenir

le verrou.

A. Easwaran et B. Andersson ont proposé dans [EA09] un protocole de synchronisation sous G-FP appelé P-PCP (i.e., *Parallel-Priority Ceiling Protocol*). P-PCP est la généralisation de PCP pour G-FP.

5.4.2.2 Modélisation des rythmes d'activation

La figure 5.13 présente les règles de construction du système en réseau de Petri temporel en multiprocesseur.

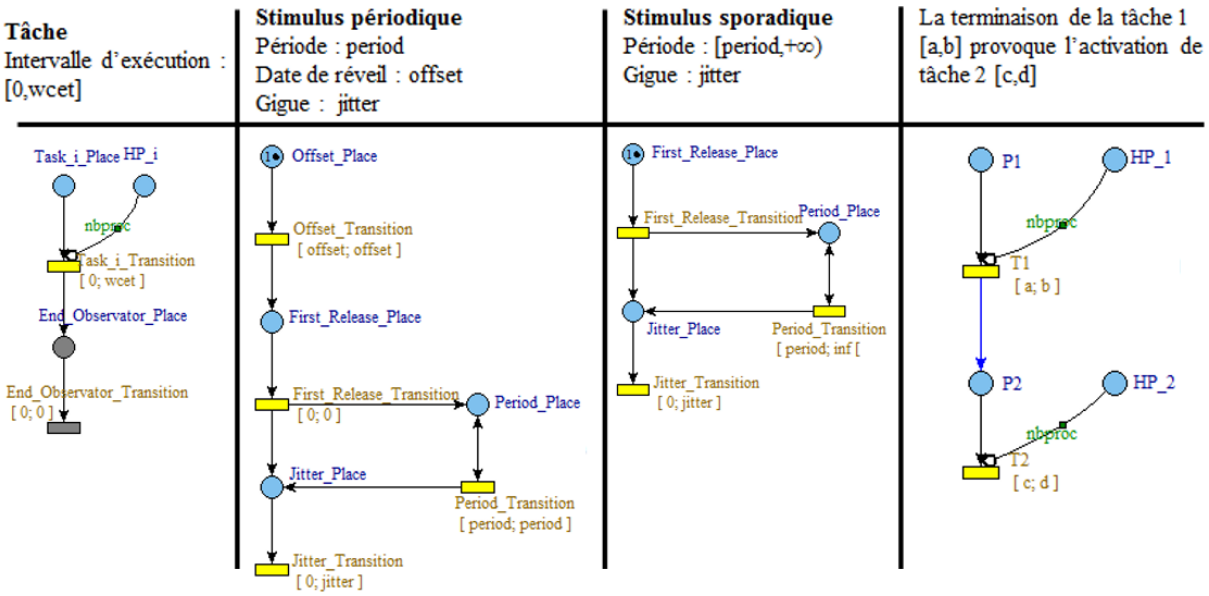

FIGURE 5.13 – Règles de construction du système sur une plateforme multiprocesseur

Les stimuli restent les mêmes que ceux du monoprocesseur (voir la figure 5.7). La différence vient des arcs inhibiteurs des places HP_i vers des transitions des tâches moins prioritaires, le poids de ces arcs inhibiteurs est égal au nombre de processeurs. Une tâche i est inhibée seulement si tous les processeurs sont occupés par des tâches plus prioritaires, i.e., le marquage HP_i est supérieur ou égal au nombre de processeurs. La figure 5.14 présente un exemple de trois tâches indépendantes sur deux processeurs : la tâche 1 est plus prioritaire que la tâche 2 et la tâche 3 est la moins prioritaire.

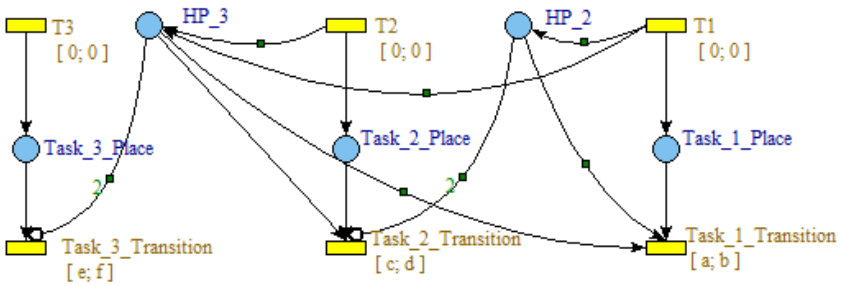

FIGURE 5.14 – Priorité entre les tâches

5.4.2.3 Modélisation des exclusions mutuelles

Malgré quelques protocoles proposés dans la littérature, à l'heure actuelle, la plupart des RTOS n'en proposent pas encore. Or, on se limite à pas de protocole. Il peut donc y avoir des

inversions de priorité mais elles seront calculées par le modèle, et donc prédites dans le pire temps de réponse obtenu.

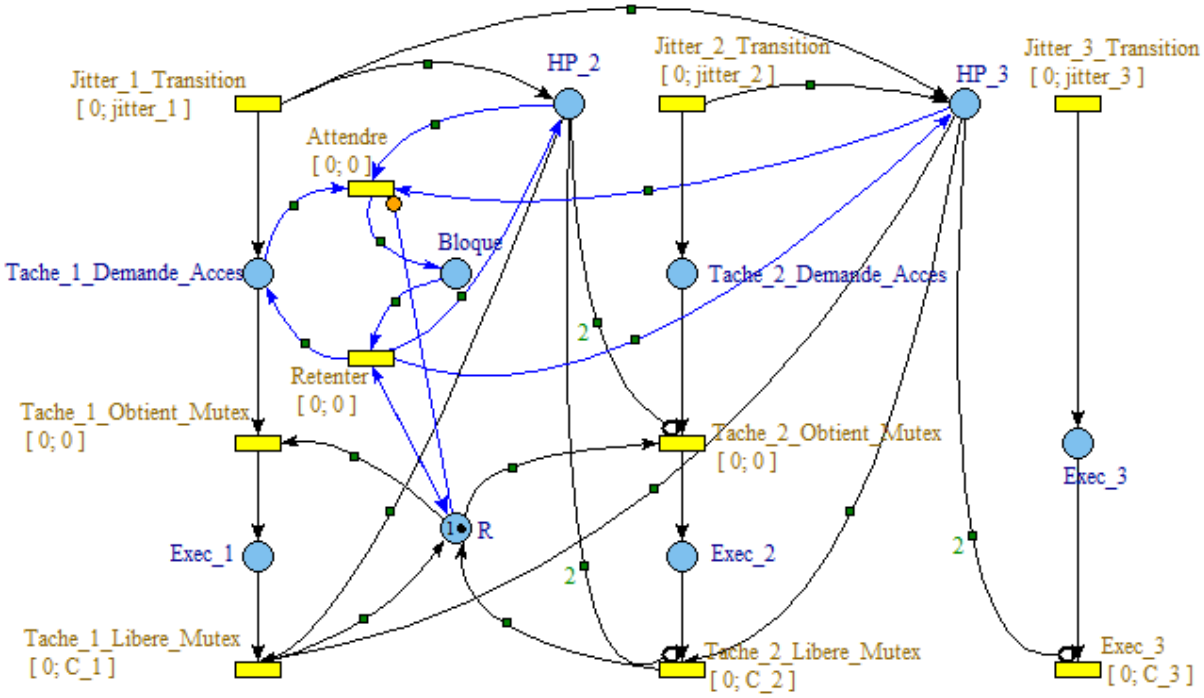

FIGURE 5.15 – Exclusion mutuelle sur multiprocesseur sans protocole de gestion

La figure 5.15 présente un exemple d'exclusion mutuelle sans protocole de gestion sur deux processeurs. L'exemple a trois tâches : τ_1, τ_2 et τ_3 (priorité $\tau_1 > \tau_2 > \tau_3$), les tâches τ_1 et τ_2 partagent une ressource critique. Les arcs nouvellement ajoutés par rapport au cas de monoprocesseur sans protocole sont marqués en bleu.

L'idée générale est que lorsqu'une tâche est bloquée en entrée de section critique, elle enlève un jeton des places HP_i de toutes les tâches moins prioritaires, exprimant ainsi le fait qu'elle est bloquée. Dans cet exemple, la tâche τ_1 se réveille, elle marque un jeton dans la place HP_2 et un dans la place HP_3 et passe à la place $Tache\_1\_Demande\_Acces$. Depuis cette place, il n'y a que la transition $Tache\_1\_Obtient\_Mutex$ sensibilisée, la transition *Attendre* est inhibée par l'arc inhibiteur logique (non stopwatch) depuis la ressource R. Lorsque la tâche τ_2 se réveille, elle marque d'un jeton la place HP_3, la tâche τ_3 est donc inhibée dès lors que les deux tâches plus prioritaires sont actives. Supposons que la tâche τ_2 obtienne le mutex avant la tâche τ_1 (i.e., la transition $Tache\_2\_Obtient\_Mutex$ est tirée avant la transition $Tache\_1\_Obtient\_Mutex$). Elle prend le jeton de la ressource R, la transition *Attendre* est désinhibée et sensibilisée. On tire la transition *Attendre*, cela retire un jeton depuis les places HP_2 et HP_3, la tâche τ_3 est donc désinhibée et un jeton est produit à la place *Bloque*. Ensuite, la transition $Tache\_2\_Libere\_Mutex$ est sensibilisée, on la tire, un jeton est rendu à la place R. La transition *Retenter* est sensibilisée, on la tire, un jeton est rendu aux places HP_2, HP_3 et $Tache\_1\_Demande\_Acces$. La tâche τ_1 n'est plus bloquée.

Le protocole P-PCP peut, sur le même modèle que le protocole PCP en monoprocesseur, être modélisé de façon similaire, en modélisant l'héritage de priorité lors de l'entrée en section critique d'une tâche.

5.5 Vérification temporelle

5.5.1 Formule ParamTPN-PTCTL

Pour observer le pire temps de réponse d'une tâche, nous utilisons la formule ParamTPN-PTCTL suivante :

$$P_{Task\_i\_Place} == 1 \rightsquigarrow_{[0,a]} P_{End\_Observator\_Place} == 1 \tag{5.3}$$

La syntaxe $(p) \rightsquigarrow_{[0,b]} (q)$ est équivalente à la propriété signifiant $AG((q)\, implique\, AF\,[0,b]\,(q))$. Cette formule est valable si et seulement si chaque fois que p est vraie, q sera également vraie dans l'intervalle de temps $[0,b]$. Cette formule utilise un paramètre a comme la limite maximale pour le pire temps de réponse. Le résultat de la vérification paramétrique du modèle pour cette formule est $a \geq \alpha$, si α est inférieur à la période de la tâche correspondante, α sera le pire temps de réponse de cette tâche. Si α est supérieur ou égal à la période, il y a des réentrances. On incrémente la valeur de $P_{Task\_i\_Place}$ d'une unité et on re-teste successivement jusqu'à ce que nous obtenions un temps de réponse inférieur ou égal à la période. On calcule donc successivement le pire temps de réponse observé pour un tâche en ne prenant aucune réentrance en compte, puis, si celui-ci est plus grand que la période, le pire temps de réponse d'une instance faisant partie d'une exécution où une instance est réentrante, etc.

5.5.2 Étude de cas

Dans cette section, nous présentons un exemple de l'ordonnancement multiprocesseurs. Donné un système de tâches sporadiques : $S = \{\tau_i < C_i, T_i, P_i >\} = \{\tau_1 < 1,4,4 >, \tau_2 < 1,4,3 >, \tau_3 < 1,3,2 >, \tau_4 < 1,2,1 >\}$ (avec P=4 est la plus prioritaire) ordonnancé sous FP sur deux processeurs. La modélisation du système est présentée dans la figure 5.16 suivante :

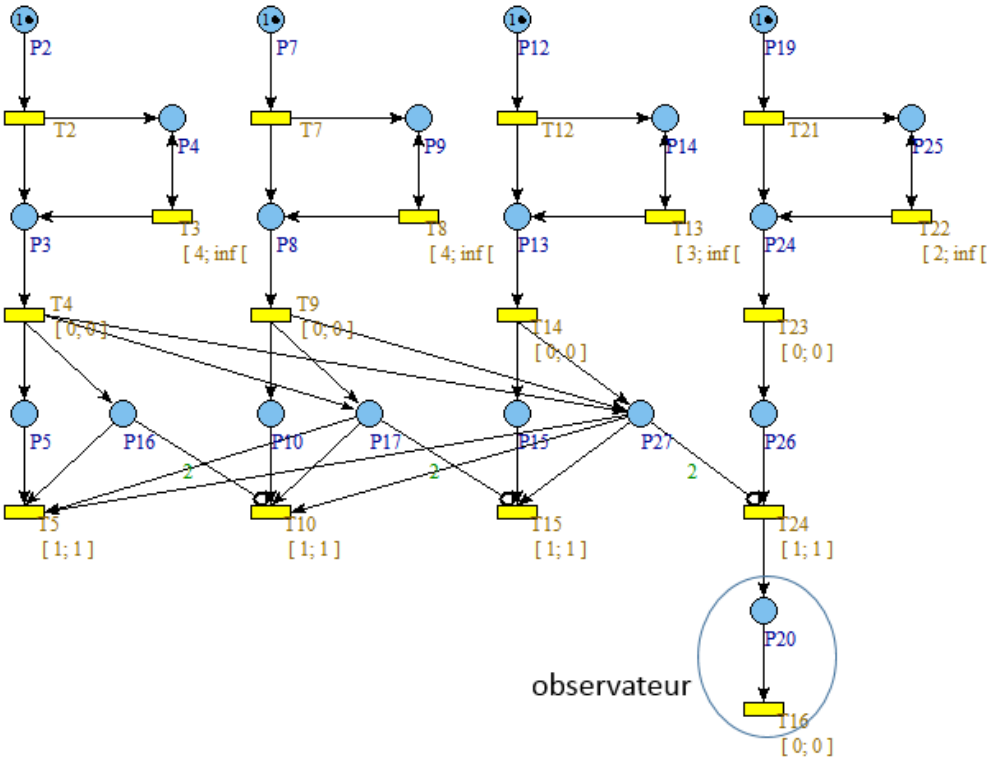

FIGURE 5.16 – Modélisation du système en Roméo

Nous pouvons calculer le pire temps de réponse de la tâche 4 avec la formule TCTL paramétrique 5.4 suivante :

$$P26 > 0 \rightsquigarrow_{[0,a]} P20 > 0 \tag{5.4}$$

Cette formule utilise un nouveau paramètre a qui représente le temps de réponse. Le résultat de model-checking paramétrique de cette formule est 5/2, supérieure à la période de cette tâche donc nous devons vérifier la deuxième fois avec la formule 5.5 suivante :

$$P26 > 1 \rightsquigarrow_{[0,a]} P20 > 0 \tag{5.5}$$

Le résultat de model-checking paramétrique de cette formule est 1/2, inférieure à la période de cette tâche, le pire temps de réponse de la tâche 4 est donc 1*période+1/2 = 2+1/2 = 5/2.

De manière équivalente, nous avons calculé le pire temps de réponse des autres tâches. Le pire temps de réponse de la tâche 3 est 2, le pire temps de réponse de la tâche 2 est 1 et le pire temps de réponse de la tâche 1 est 1.

5.6 Conclusion

Dans ce chapitre, d'abord nous avons proposé un test exact calculant le pire temps de réponse des tâches dépendantes au cas monoprocesseur en se basant sur le réseau de Petri temporel. Le test prend en compte le protocole priorité plafond immédiat pour la gestion des ressources critiques. Ensuite, nous avons étendu le test au cas multiprocesseur identique. Nous avons fait un état de l'art sur les protocoles de gestion de ressource critique multiprocesseur. Néanmoins, la plupart des protocoles dans la littérature ne sont pas proposés par les RTOS donc nous nous limitons à pas de protocole de gestion.

Chapitre 6

Vers un référentiel d'analyse générique

Sommaire

La contribution présentée dans ce chapitre consiste à proposer une démarche à base de modèles permettant d'un côté d'aider les concepteurs à faire le bon choix de tests et de l'autre côté de permettre aux chercheurs et aux analystes de partager leur savoir faire en termes de tests d'analyses et leurs conditions d'utilisation. L'objectif est non seulement de permettre aux concepteurs d'agir d'une manière autonome et sûre lors du processus d'analyse mais aussi d'augmenter et faciliter l'utilisation des tests d'analyse dans les pratiques d'ingénierie.

6.1 Introduction

Comme nous l'avons évoqué à plusieurs reprises dans ce manuscrit, la vérification temporelle est une étape importante dans la conception des systèmes embarqués temps réel. L'évolution technologique des architectures matérielles et logicielles a toujours été un facteur stimulateur de l'évolution des tests de la vérification temporelle. La communauté de la validation temporelle a toujours accompagné cette évolution en proposant des nouveaux tests basés sur la simulation, les méthodes formelles et/ou les méthodes algébriques. Cette multitude de tests d'analyse a rendu l'étape de la validation, au sein du cycle de vie de conception, à la fois plus pertinente et compliquée. (i) La pertinence vient du fait que les tests proposés mènent de plus en plus vers des conceptions très raffinées (proches de l'exécution du système dans le monde réel) prenant en considération plusieurs facteurs pratiques (comme la précédence, la préemption, suspension, etc.). (ii) La complexité est due à la multitude des tests existants, ce qui rend le choix de l'analyse appropriée, permettant d'analyser et/ou de valider correctement une conception, une tâche très difficile même pour les experts vu le caractère critique des domaines d'applications. Récemment, plusieurs outils gratuits et commerciaux, implémentant des tests d'analyses, ont été proposés afin de faciliter la tâche aux concepteurs. Cependant, un outil est souvent basé sur un modèle d'analyse particulier (modèle de transactions, modèle de composants, etc.) et ne propose pas tous les tests d'analyses possibles. De plus, malgré que l'inclusion des outils dans un processus à base de modèles permet de lier les entrées d'outils aux différents langages de modélisation (notamment les standards), le choix du test est souvent dirigé par l'expérience du concepteur. Notre participation, aux projets avec des partenaires industriels, nous a permis de constater que les pratiques industrielles favorisent souvent la réutilisation des tests qui fonctionnaient lors des anciens projets. En effet, ce principe de la réutilisation pousse souvent les concepteurs à adapter en amont leurs conceptions "métiers" pour qu'elles correspondent aux tests habituels au lieu d'explorer de nouveaux tests. Certes, les pratiques industrielles évoluant dans un milieu concurrentiel préoccupé par le délai de commercialisation (*time-to-market* en anglais) privilégient des tests d'analyse déjà expérimentés. Cependant, ces pratiques peuvent mener à des choix de conception conservatifs et par conséquent demandant plus de ressources (ce qui génère un coût additionnel). La non-exploration des nouveaux tests est aussi due à l'écart sémantique qui se trouve entre les modèles de tâches des tests d'analyses et les artefacts de modèles "métiers" utilisés pour représenter les systèmes à concevoir.

L'objectif de ce chapitre est de capitaliser les efforts effectués dans le but de faciliter l'utilisation des tests de la validation temporelle. En effet, ce chapitre propose trois contributions complémentaires. La première contribution consiste en la proposition d'un langage permettant de décrire les contextes d'analyse (introduits dans la section 3.6) indépendemment des langages de modélisation. La deuxième contribution présente une approche à base de modèles permettant d'augmenter la fiabilité et d'obtenir des référentiels d'analyse corrects par construction. La troisième contribution permet de transformer un référentiel d'analyse générique pour qu'il soit adapté à un langage de modélisation précis. Cela permettra d'inclure l'analyse temporelle au moment de la modélisation mais aussi de faciliter le prototypage de certains tests d'analyse sans passer par les outils classiques. L'objectif du prototype rapide est de faire face à la lenteur du processus de transfert technologique des tests d'analyses proposés généralement par des chercheurs académiques.

Le reste du chapitre est organisé comme suit. La section 6.2 présente en détail la problématique abordée en plus des travaux existants. La section 6.3 présente le positionnement du travail. La section 6.4 est dédiée au langage de de description des règles d'identification afin d'expliciter le contexte d'analyse relatif à un test donné. La section 6.5 présente le méta-modèle des référentiels d'analyse ainsi que les différentes fonctionnalités permettant de renforcer l'exactitude des contenus. La section 6.6 présente l'utilisation des référentiels d'analyse avec les langages de modélisation comme étant un moyen d'aide à la décision permettant de donner plus d'autonomie aux concepteurs non-experts du domaine de la vérification temporelle. Dans cette section, nous

montrons également les différents scénarios d'utilisation ainsi qu'une solution du prototypage afin de pouvoir intégrer un test de validation temporelle (basé sur une méthode algébrique) dans le processus de conception sans passer par un outil externe nécessitant la transformation de modèles. Enfin, la section 6.7 conclut ce chapitre.

6.2 Travaux connexes : méthodes d'aide à la conception

Récemment, quelques travaux d'aide à la conception ont été proposées pour orienter les concepteurs vers les bonnes analyses temporelles. Par la suite, nous introduisons cinq approches.

6.2.1 L'approche MoSaRT

Y. Ouhammou a proposé dans [Ouh13] une méthode de développement collaborative liant la modélisation et la validation. La figure 6.1 présente une vue d'ensemble du framework MoSaRT.

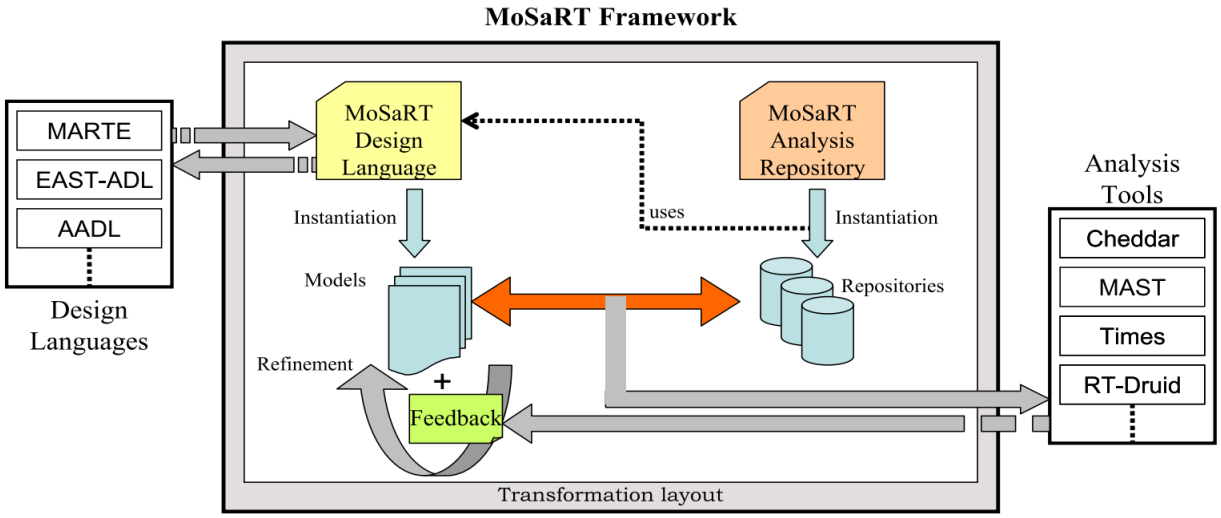

FIGURE 6.1 – L'utilisation du framework MoSaRT [Ouh13]

Le framework offre à la fois la modélisation des systèmes temps réel en proposant *MoSaRT Design Language* pour les concepteurs et la construction des référentiels d'analyses pour les analystes (instances de MoSaRT Analysis Repository). Le *MoSaRT Design Language* peut aussi être utilisé comme un langage intermédiaire pivot pour d'autres langages de modélisation. Les experts d'analyse instancient le référentiel d'analyse (i.e., *MoSaRT Analysis Repository* dans la figure 6.1) pour stocker les analyses accompagnées par leurs contextes d'utilisation et les outils qui supportent ces analyses.

Les référentiels d'analyses permettent à tout stade de la conception de déterminer les bons tests d'analyses qui correspondent au modèle ainsi que les outils les supportant. Dans le cas où le modèle ne correspond à aucun test d'ordonnançabilité, le concepteur obtient des informations sur l'incompatibilité (i.e., les hypothèses qui ne sont pas satisfaites) pour adapter sa conception s'il le faut. Néanmoins, le point faible de cette approche c'est que le processus d'identification est basé sur des règles d'identification écrites en OCL (Object Constraint Language), exprimées forcément sur le langage de modélisation de MoSaRT. L'analyste doit donc maîtriser à la fois la syntaxe OCL et la structure du langage de modélisation de MoSaRT (c'est à dire le métamodèle de MoSaRT). De plus, l'utilisation du référentiel d'analyses est limitée aux modèles conçus en langage de modélisation de MoSaRT.

Pour déterminer les analyses et outils qu'il faut utiliser, le modèle est soumis à un processus d'identification. L'objectif du processus d'identification est de vérifier la conformité entre le modèle (en cours de conception) et les contextes d'analyse. En effet, chaque contexte est doté

d'un ensemble de règles d'identification. Une **règle d'identification** est une hypothèse relative à l'architecture (logicielle et matérielle) ou au comportement temporel. L'évaluation de chaque règle d'identification peut s'avérer vraie ou fausse selon les caractéristiques du système étudié. La figure 6.2 résume le processus d'identification en deux étapes : la première étape consiste à évaluer les règles d'identification sur le système étudié (i.e., l'étape "Evaluation Process" de la figure), la deuxième étape consiste à comparer le résultat d'évaluation des règles d'identification avec leurs valeurs attendues dans chaque contexte (l'étape "Comparison Process" de la figure). Le contexte choisi est le contexte dont toutes les valeurs attendues sont satisfaites après évaluation. Le framework offre ensuite la possibilité de transformer le modèle conçu en MoSaRT vers le formalisme d'entrée des outils d'analyse (e.g., Cheddar, MAST, etc.).

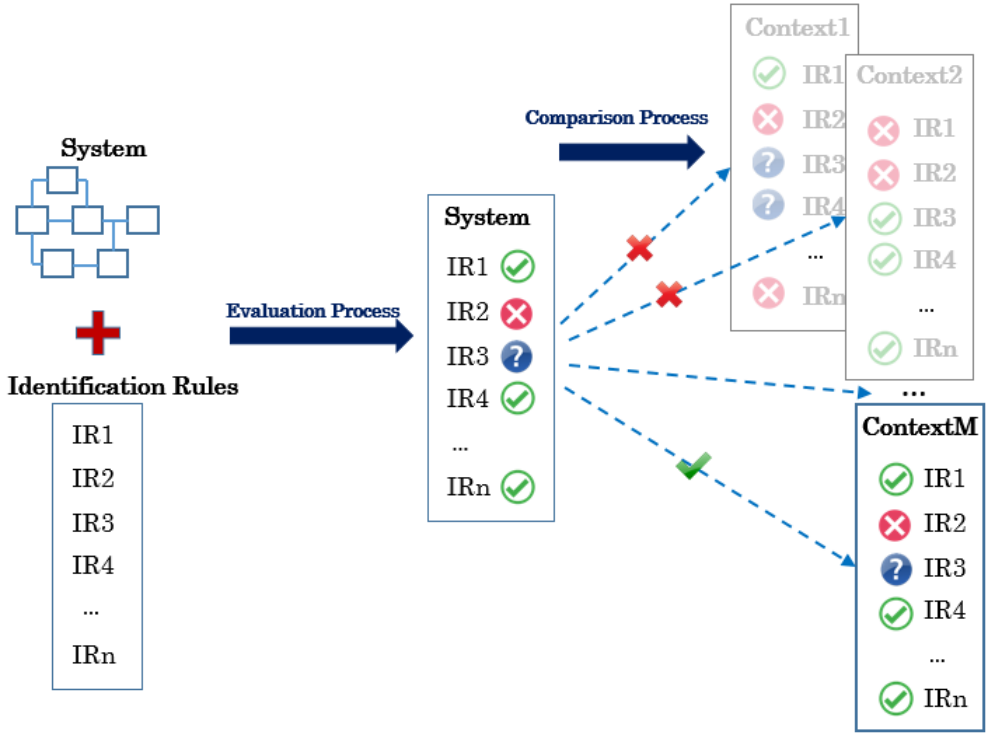

FIGURE 6.2 – Processus d'identification

6.2.2 L'approche des patrons de conception

Gaudel et al. [GSP+14] ont proposé une approche à base de patrons de conception (*Design pattern* en anglais) implémenté dans l'outil Cheddar. Contrairement à *MoSaRT analysis repository* qui propose des contextes orientés modèles d'analyse, les patrons de conception de Cheddar sont orientés architecture logicielle. Par conséquent, chaque patron de conception peut correspondre à un ensemble d'analyses d'ordonnançabilité. Parmi les patrons de conception supportés on trouve le profil Ravenscar. La figure 6.3 présente le processus de détection des tests applicables pour le système étudié. Les **contraintes d'applicabilité** représentent les hypothèses à garantir.
Ce processus de détection ressemble au processus d'identification de l'approche MoSaRT. Néanmoins, dans le cas où la conception du système ne correspond à aucun patron de conception, aucune information sur l'incompatibilité n'est fournie. Ce processus n'est pas explicite comme dans MoSaRT. En effet, le concepteur ne sait donc pas comment corriger sa conception pour s'adapter aux patrons de conception existants. En outre, cette approche est implémentée d'une façon statique (*hard coded*). Autrement dit, l'ajout de nouveaux patrons de conception demande un effort considérable pour la compréhension et la modification du code source.

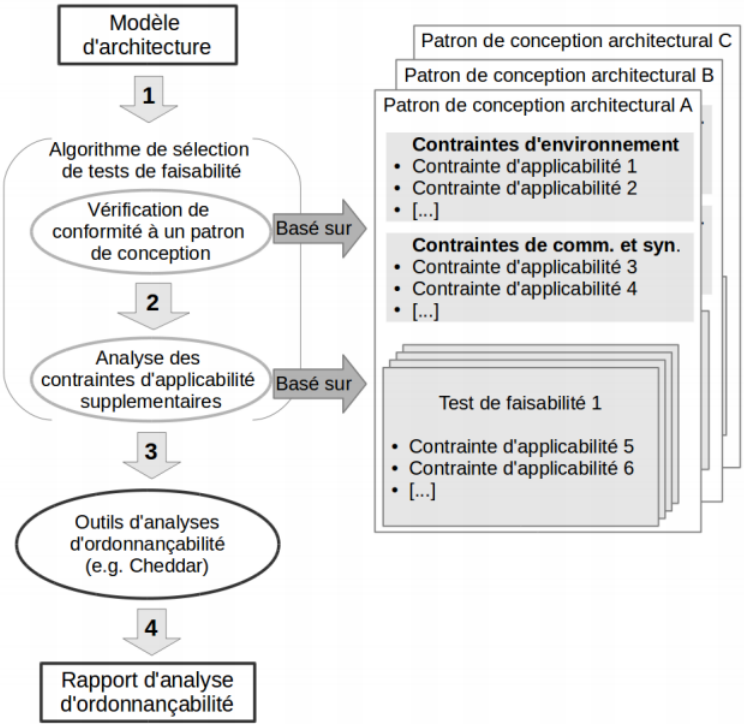

FIGURE 6.3 – Processus de sélection de tests de faisabilité [GAU14]

6.2.3 Approche MAIWEn

Brau et al. ont proposé MAIWEn (***M**odeling and **A**nalysis* **I**ntegration **W**orkbench for the **En**gineering of embedded systems) [BNH17].MAIWEn se base sur le concept des **contrats** [BHN15]. Un contrat définit formellement l'interface d'une analyse en termes de quatre éléments : les données nécessaires (*Inputs*) pour l'analyse, les résultats fournis par l'analyse (Outputs), les propriétés (sous forme d'hypothèses) nécessaires (*Assumptions*) pour l'analyse et les propriétés garanties (*Garantees*) par l'analyse. La figure 6.4 représente le modèle de contrat.

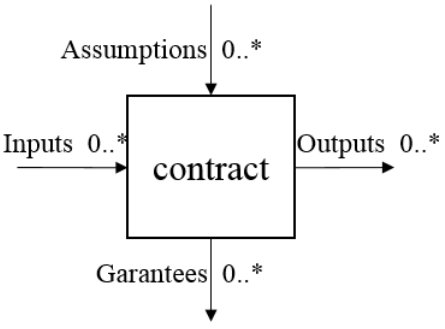

FIGURE 6.4 – Modèle de contrat [BHN15]

Les résultats fournis par une analyse peuvent être les données d'entrée (nécessaires) pour d'autres analyses (idem pour les propriétés). De plus, deux types de relations peuvent exister entre les contrats : la précédence verticale et la précédence horizontale. Deux contrats sont en précédence verticale si les propriétés garanties par un contrat font partie des propriétés nécessaire pour l'autre contrat (idem pour la précédence horizontale). On peut alors déterminer à travers les contrats tous les enchaînements et les combinaisons possibles entre les tests d'analyse. Cependant, cette approche ne se focalise pas sur la compatibilité du système (en cours de conception) avec les tests. De plus, les informations nécessaires pour l'orchestration (les données d'entrée, de sortie, les propriétés nécessaires et garanties) sont saisies de manière ad-hoc ce qui rend

l'extension de MAIWEn difficile.

6.2.4 Approche CONSERT

CONSERT (*CONServative Endogenous Repository based Transformations*) proposé par Bui Long et al. [LOG+17] a repris le principe du référentiel d'analyse de MoSaRT. CONSERT est en fait un référentiel qui stocke les transformations accompagnées des règles de détection permettant de déterminer les transformations appropriées pour un système.

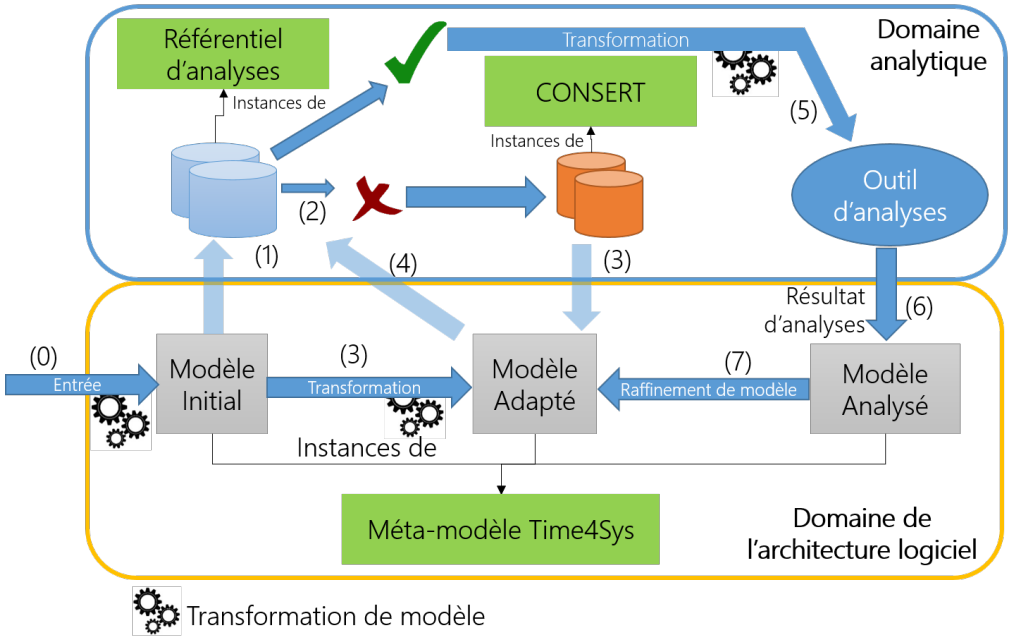

FIGURE 6.5 – Utilisation globale de *CONSERT* [LOG+17]

En cas d'absence de test d'analyse compatible avec le système en cours de conception, CONSERT propose d'adapter la conception par une suite de transformations conservatives endogènes vers des modèles plus abstraits mais analysables.

6.2.5 Approche TEMPO

TEMPO est un framework développé par *Thalès Research & Technology*. L'approche TEMPO est assez similaire à celle de CONSERT. TEMPO propose un méta-modèle pivot capturant la vue métier, un méta-modèle pivot capturant la vue analyse et un ensemble de règles de transformation entre eux. Le méta-modèle pivot orienté conception métier est un sous-profil du langage MARTE adapté pour les architectes de Thalès, permettant ainsi de représenter les propriétés de performance de manière plus simple et plus intuitive. Le méta-modèle pivot d'analyse se base sur l'ensemble de concepts génériques des analyses temps réel classiques utilisées au sein de Thalès. L'utilisation globale de TEMPO est présentée dans la figure 6.6. Le modèle d'entrée de TEMPO peut être conçu par le méta-modèle de conception de TEMPO ou peut être importé d'autres langages de conception. Le modèle de conception est ensuite transformé en modèle d'analyse et puis exporté vers le formalisme d'entrée des outils d'analyse. Le résultat d'analyse est injecté dans le modèle de conception d'origine. Cette approche n'explicite aux utilisateurs la compatibilité entre le modèle et l'analyse. L'usage donc de TEMPO est dirigé par l'expertise des architectes.

6.2.6 Méthodologie Optimum

Optimum [MTPG11] est une méthodologie basé sur le langage MARTE permettant de mettre à disposition des concepteurs les artefacts nécessaires à la modélisation des systèmes temps réel

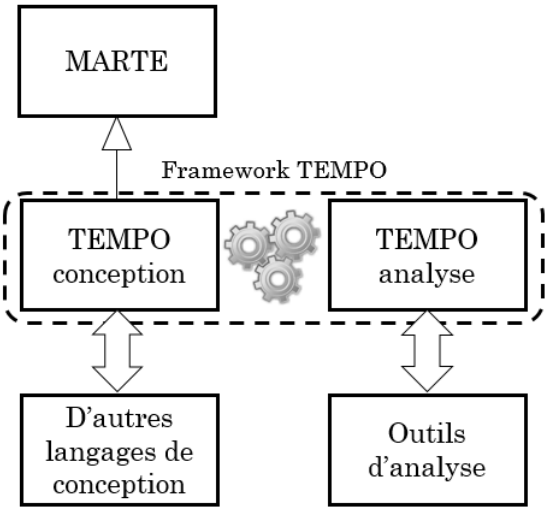

FIGURE 6.6 – Utilisation globale de TEMPO

en partant du modèle fonctionnel. En conséquent, la méthodologie propose un processus à deux étapes. La première étape consiste à spécifier, à partir d'un modèle fonctionnel en entrée, un modèle de flux de bout-en-bout (appelé workload model). La deuxième étape consiste à raffiner le workload modèle afin d'obtenir un modèle d'analyse suite à l'application d'une stratégie d'allocation des tâches. Cependant, la méthodologie Optimum ne porte que sur la partie modélisation et les artefacts de MARTE à utiliser dans chacune des étapes du processus, et ne propose aucun moyen d'aide à la détection de stratégie adéquate d'allocation des tâches ni d'aide à l'analyse d'ordonnançabilité.

6.2.7 Discussion

La figure 6.7 est une extension de la figure 2.16 où les travaux, approches et outils discutés ci-dessus ont été placés tout au long du processus de modélisation afin d'avoir une vue d'ensemble. On peut remarquer qu'il n y a aucune approche qui couvre l'ensemble des processus à savoir l'exploration, le dimensionnement et la validation. Une combinaison de deux ou plusieurs approches (ou leur inclusion dans une chaîne de valeur) peut alors être intéressant.

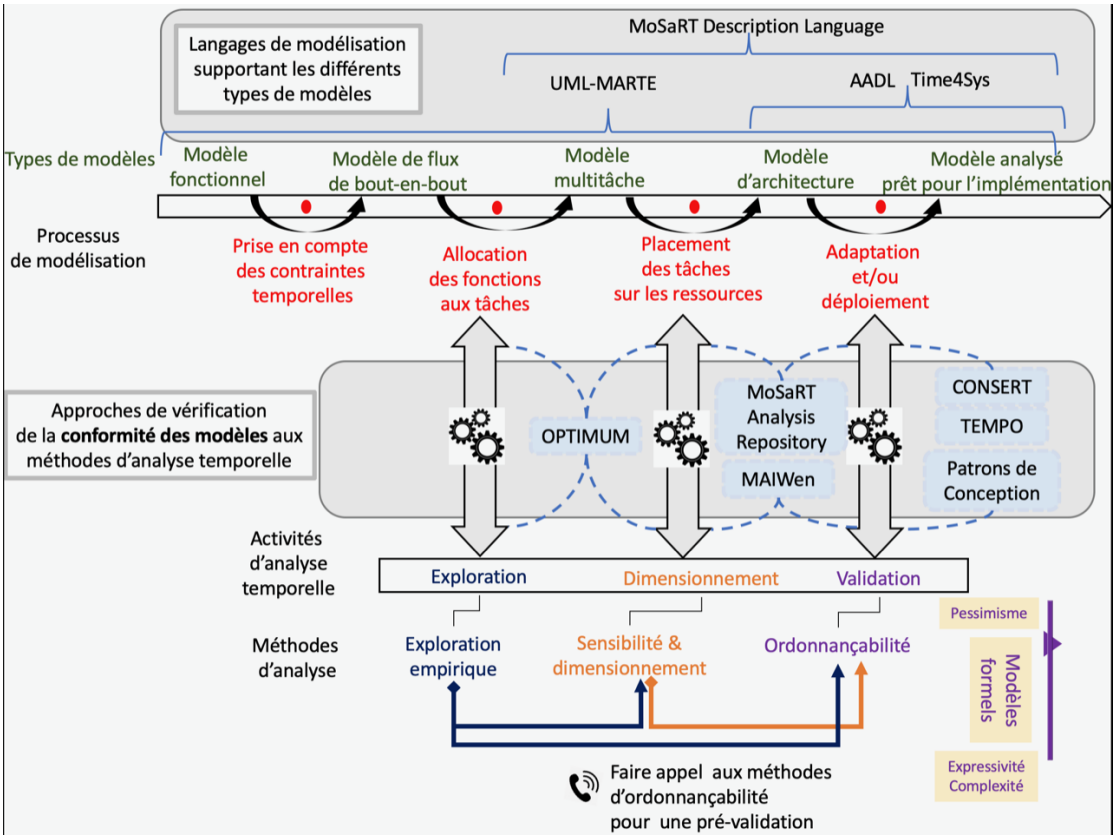

FIGURE 6.7 – Modélisation dirigée par l'analyse d'ordonnançabilité

6.3 Positionnement

Plusieurs travaux ont traité le recensement des tests existants et leurs contextes d'utilisation pour faciliter leur usage. On peut classifier ces travaux en trois catégories. (i) La première catégorie est la catégorie des travaux de *Surveys* qui ont été publiés dans plusieurs revues du domaine afin de présenter un état de l'art sur l'ensemble des tests proposés pour un type de système particulier. Parmi ces travaux on peut trouver à titre d'exemple [SAÅ+04] de L. Sha et al., [DB11c] de R. Davis et al., et [SY15] de Martin Stigge et Wang Yi. Ce dernier est une étude qui recense une douzaine de modèles d'analyse, à base de graphe, dédiés uniquement à des systèmes de tâches indépendantes à priorités fixes sur une architecture monoprocesseur. (ii) La deuxième catégorie est celle des outils d'analyse. Comme nous l'avons évoqué dans l'introduction de ce chapitre, malgré la panoplie d'outils existants (voir la sous-section 3.5), ils ne couvrent pas tous les tests académiques et théoriques. De plus, ils sont plus dédiés à l'automatisation du calcul du test qu'à l'orientation des concepteurs vers les tests les plus adaptés à leurs besoins. (iii) La troisième catégorie regroupe les travaux qui ont été proposés pour faciliter l'usage et augmenter l'utilisation des tests théoriques dans une démarche dirigée par les modèles (voir la section 6.2). Parmi les travaux de cette catégorie on trouve à titre d'exemple l'approche MAIWEn [BNH17] et l'approche des patrons de conception. Cette catégorie qui est la plus en phase avec la contribution de ce chapitre. Néanmoins, le point commun manquant entre ces différentes catégories est l'**évolution**. Autrement dit, tandis que l'évolution technologique nécessite de revoir les tests d'analyse existants pour plus de précisions voire même les revisiter, elle impacte les outils d'analyses car l'ajout d'un nouveau test ne se fait pas systématiquement. Ceci nécessite, en général, la maîtrise du code source de l'outil accueillant avant de coder en dur la nouvelle méthode. Pour toutes ces raisons, la contribution de ce chapitre se basera essentiellement sur l'extension des travaux initiés au laboratoire par Y. Ouhammou à savoir le framework *MoSaRT analysis repository* (référentiel d'analyse en français) [Ouh13]. Ce framework fait partie des tra-

vaux de la troisième catégorie, mais contrairement aux travaux présentés ci-dessus, il permet de gérer l'évolution. En effet, un travail d'explicitation et de modélisation du domaine des analyses temporelles a été fait et a donné lieu à un méta-modèle qui peut être instancié et enrichi d'une manière évolutive.

6.3.1 MoSaRT analysis repository

En raison de l'exactitude temporelle d'un système temps réel, l'analyse d'ordonnancement est devenue l'une des analyses principales. Le choix des tests et des techniques de dimensionnement appropriés a un impact important sur le cycle de développement d'où l'utilité de l'approche MoSaRT (voir 6.2.1). Cette approche est basée sur le référentiel d'analyse [Ouh13] qui a contribué à améliorer la manière dont les concepteurs vérifient la conception de leurs systèmes. Dans cette section, on présente brièvement ce référentiel d'analyse. La figure 6.8 présente les concepts de base sur le référentiel d'analyse de Y. Ouhammou.

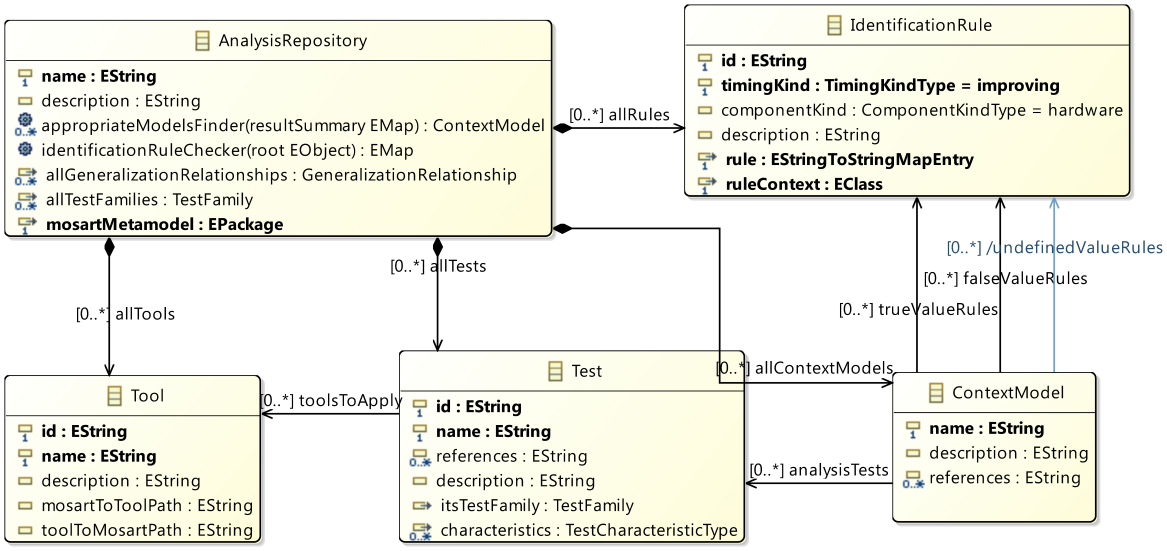

FIGURE 6.8 – Les concepts de base du référentiel d'analyse de Y. Ouhammou

Le référentiel d'analyse se compose de quatre éléments : les règles d'identification, les contextes, les tests et les outils.

— La règle d'identification est la notion fondamentale du référentiel d'analyse. Une règle d'identification représente une hypothèse sur le système. Par exemple, "le système ne contient que des tâches périodiques", "le système a une architecture de monoprocesseur", etc. sont des règles d'identification. La valeur de chaque règle d'identification dépend du système étudié, elle peut être évaluée comme vraie ou fausse.

— Le contexte représente le modèle d'analyse caractérisant le système et permet d'identifier les analyses pouvant être appliquées. En fait, un contexte correspond à un ensemble d'hypothèses. Par exemple, le contexte de Liu&Layland [LL73a] repose sur les hypothèses suivantes : "le système a une architecture monoprocesseur", "toutes les tâches sont indépendantes", "toutes les tâches sont caractérisées par le pire temps d'exécution et la période", etc. Les hypothèses constituant le contexte sont représentées par des règles d'identification. Pour un contexte donné, les règles d'identification sont caractérisées par leurs valeurs attendues, suite à l'évaluation, qui peuvent être : soit vraie (*trueValueRules*), soit fausse (*falseValueRules*), soit indéfini (*undefinedValueRules*) qui veut dire que la véracité de cette règle n'a pas d'importance pour le contexte en question.

— Le test représente la technique d'analyse qui permet de conclure sur l'ordonnançabilité du système. Puisque le contexte représente la situation d'analyse des systèmes, un certain nombre de tests sont applicables dans un contexte. Pour un contexte spécifique, un test peut être exact, suffisant ou nécessaire. Les tests sont également caractérisés par l'aspect de viabilité. Un test d'analyse est dit viable (*sustainable*) par rapport à un système, si le système (jugé valide par le test) reste valide après la modification des paramètres suivants (présentés dans la figure 6.9) : la décroissance du temps d'exécution (C-viable), la croissance de la période (P-viable), la décroissance de la gigue (J-viable) ou la croissance de l'échéance relative (D-viable).

— L'outil représente l'implémentation d'une technique d'analyse. En cas de besoin, la transformation du modèle de conception en formalisme d'entrée de cet outil est aussi stockée au niveau du référentiel. Le résultat de test peut aussi être transformé (transformation inverse) et injecté dans le modèle initial.

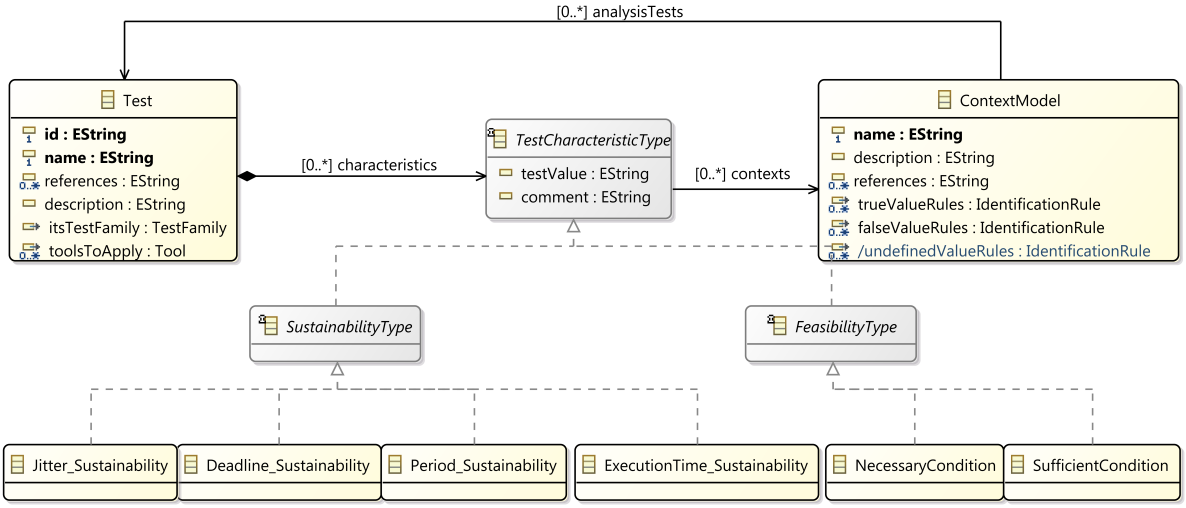

FIGURE 6.9 – Extrait du méta-modèle exprimant le test, le contexte et leur relation

6.3.2 Discussion et démarche souhaitée

Dans le cycle de développement des systèmes temps réel, la tâche principale des concepteurs est de créer les modèles avec suffisamment d'information pour prendre en charge les exigences temporelles dans différentes étapes de la conception. L'objectif est d'utiliser les modèles de conception pour piloter les analyses d'ordonnancement afin vérifier les exigences temporelles et prévoir le comportement temporel du système étudié. Pour effectuer une analyse, le modèle de conception doit d'abord être transformé en modèle d'analyse qui permet, à travers les outils d'analyse, pour produire des résultats qui seront ensuite utilisés pour raffiner successivement le modèle de conception. Aujourd'hui, grâce à l'ingénierie dirigée par les modèles, la transition du modèle de conception en modèle d'analyse a été raccourcie. Différents langages de modélisation ont été proposés pour concevoir les modèles de conception tels que MARTE [OMGa], AADL [AAD19] et MoSaRT [Ouh13]. La figure 6.10 présente notre démarche globale.

Après la conception du système, le concepteur utilise les référentiels comme conseiller d'analyse. Afin de séparer les référentiels d'analyse des langages de modélisation, assurer leur indépendance ainsi que leur enrichissement par des analystes non experts en langages de modélisation, nous proposons un langage appelé *Identification Rule Language* (IRL). Il est dédié aux analystes des systèmes temps réel pour décrire les règles d'identification. Le nouveau langage est indépendant de tous langages de conception et se base sur un dictionnaire de concepts extraits

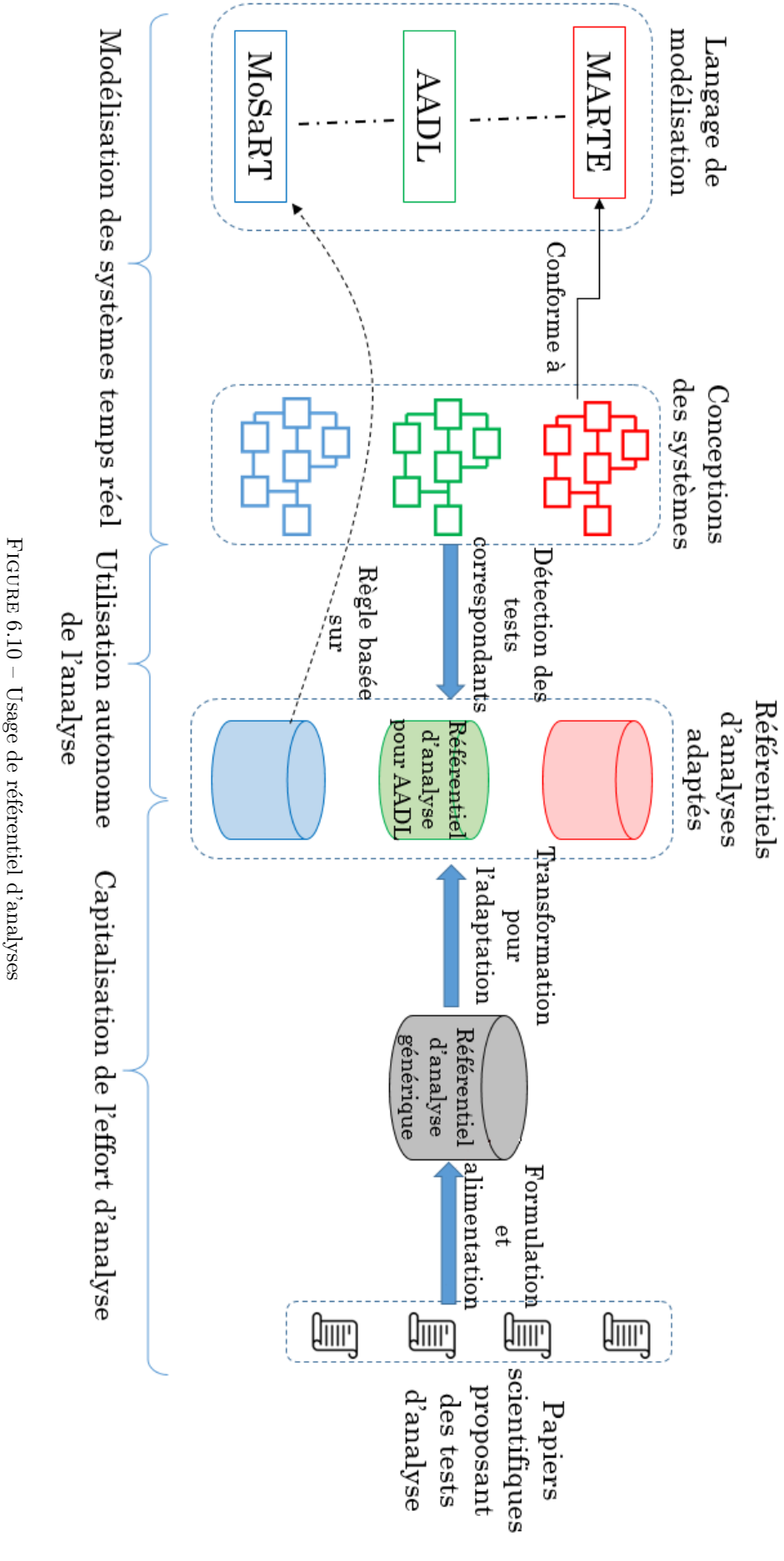

FIGURE 6.10 – Usage de référentiel d'analyses

de la littérature. L'idée derrière ce langage n'est pas de créer un nouveau langage de contrainte tel que OCL, mais de créer un langage pivot avec lequel les utilisateurs finaux (par exemple, chercheurs universitaires et analystes) peuvent définir facilement et avec précision les hypothèses sans devoir maîtriser OCL ou le langage de conception cible. Le référentiel d'analyse qui utilise IRL pour exprimer ses règles sera donc générique.

6.4 Identification Rule Language (IRL)

Le modèle le plus simple permettant de définir des hypothèses est la logique propositionnelle. Par exemple, "le système a une architecture monoprocesseur" est une proposition. Cependant, la logique propositionnelle présente certaines limites. L'une des principales limites est qu'il est impossible de parler de propriétés qui s'appliquent à des catégories de sujets. Par exemple, "il existe au moins n tâches dont le temps de réponse est inférieur à l'échéance" est une hypothèse qui ne peut pas être exprimée à l'aide de la logique propositionnelle. Ainsi, l'expressivité de la logique des prédicats est nécessaire dans notre situation. La différence principale est que la logique des prédicats, contrairement à la logique des propositions, contient des quantificateurs (existentiel, universel, ...) et permet de travailler avec des variables. Pour les raisons mentionnées ci-dessus, on s'est inspiré du paradigme de la logique des prédicats pour formaliser notre langage de définition des règles d'identification.

On rappelle que l'objectif de notre travail n'est pas de proposer une autre implémentation de la logique du premier ordre (comme OCL ou la logique de description (*Description Logic* - DL [Baa02]) qui est utilisée principalement dans les domaines de l'ingénierie basée sur la connaissance et les ontologies), mais un langage contrôlé et expressif se focalisant ainsi sur un domaine spécifique. IRL étant un langage dédié, on présente ci-après sa syntaxe abstraite (méta-modèle). La figure 6.11 montre un extrait du méta-modèle IRL (exprimé en Ecore [Ecl]) où les classes de couleur grise sont des classes abstraites. Le méta-modèle contient deux parties. La première partie (voir la partie A de la figure 6.11) décrit la structure des prédicats qui est fixe, tandis que la seconde partie (voir la partie B de la figure 6.11) est évolutive et dédiée aux concepts temps réel utilisés pour les prédicats.

6.4.1 Syntaxe abstraite

6.4.1.1 Structure du prédicat (logique du premier ordre)

L'élément central de la première partie du langage IRL est la classe `Expression` (voir la figure 6.11) dont les instances représentent les prédicats en réalité. Chaque instance de la classe `Expression` se compose d'un ensemble de variables (instances de la classe `Variable`) et d'une proposition (instance de la classe `Proposition`).

Chaque variable (instance de la classe `Variable`) est caractérisée par un quantificateur (*exist* ou *forAll* qui sont représentés par les énumérations `ExistentialQuantifier` et `UniversalQuantifier`) et un nom. Une variable peut être soit naturelle (instance de la classe `NaturalInteger`), soit réelle (instance de la classe `Real`) ou appartient à un ensemble restrictif des éléments défini par l'utilisateur (instance de la classe `UserDefinedDomain`).

Chaque proposition a un opérateur principal. Un opérateur peut être une fonction personnalisée en tant qu'instance de la classe `BinaryFunction` ou `UnaryFunction`. Sinon, l'opérateur est une instance de la classe `BasicArithmeticOperator` (par exemple : addition, soustraction,...). Selon le type de la fonction utilisée, une proposition comporte un ou plusieurs opérandes (instance de la classe `Operand`). Un opérande peut être une instance des classes suivantes : `Constant`, `PropertyValue`, `Proposition` ou `Accessor`. Chaque instance de la classe `Accessor` peut faire référence à un élément instance de la classe abstraite `ReferenceableElement`. Concrètement, l'élément référençable peut être une variable (instance de la classe `Variable`), un primitif (instance de la classe `PredefinedSet`) ou un ensemble défini par l'utilisateur (instance de la classe

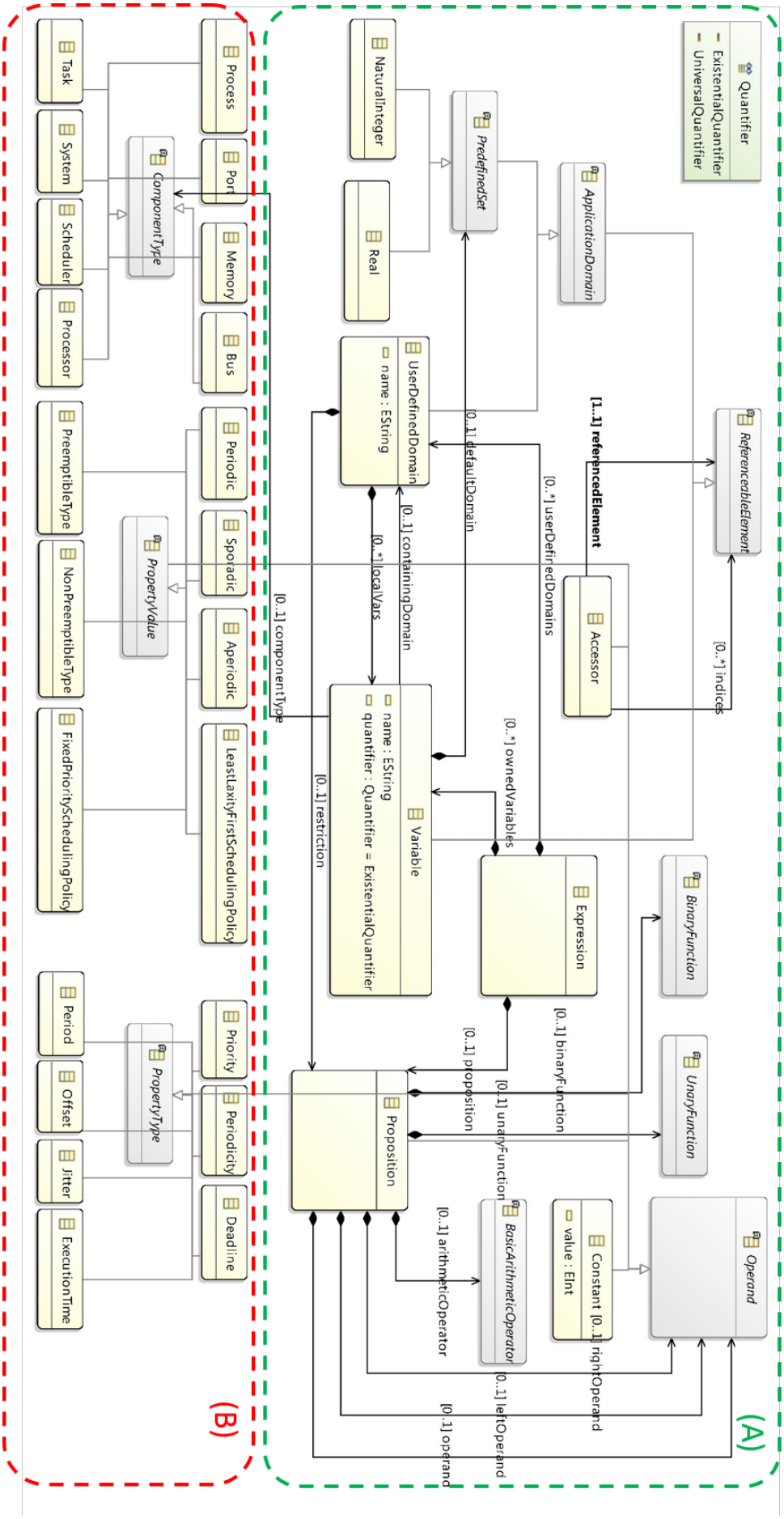

FIGURE 6.11 – Extrait du méta-modèle de prédicat : (A) : structure de prédicat (B) : Concepts de domaine temps réel

`UserDefinedDomain`).

6.4.1.2 Dictionnaire de concepts

Alors que notre objectif est d'obtenir un langage pivot indépendant de tous les langages de conception, nous avons choisi de séparer le méta-modèle en deux partie. Cela permettra d'assurer l'évolution des concepts utilisés dans les systèmes temps réel. Dans cette perspective, contrairement aux autres langages de modélisation standard, qui sont des modèles prescriptifs, nous avons opté à ce que la deuxième partie du méta-modèle IRL soit descriptive [Hes06]. Par conséquent, nous avons conçu la deuxième partie comme un dictionnaire de concepts des systèmes temps réel en nous inspirant de la notion de l'ontologie. Notons qu'à notre connaissance, il n'existe aucune ontologie consensuelle du domaine de système temps réel. Actuellement, notre proposition d'ontologie contient plus de 90 concepts classés en trois catégories. (i) Catégorie `ComponentType` qui contient la plupart des entités utilisées dans les système temps réel. (ii) Catégorie `PropertyType` qui regroupe les propriétés courantes des système temps réel, utilisées pour la validation temporelle. (iii) La valeur de chaque élément de la catégorie `PropertyType` peut être une valeur littérale ou un élément du groupe `PropertyValue`.

La partie (B) de la figure 6.11 représente une transposition de notre ontologie dans le méta-modèle IRL. Il faut noter que toutes les classes sont liées par des relations d'héritage à leurs classes principales abstraites correspondantes. Cette structure permettra d'étendre le méta-modèle en ajoutant de nouvelles classes sans impacter les instances existantes. En d'autres termes, l'évolution de la partie (B) ne devrait pas avoir d'impact sur les hypothèses ayant déjà été exprimées en IRL avant son évolution.

6.4.2 Syntaxe concrète

Notre syntaxe concrète est textuelle et proche de la langue naturelle. Cette syntaxe qui sert à instancier IRL est développée grâce à l'outil Xtext [EB10]. Le listing 6.1 présente un extrait de la grammaire de la syntaxe concrète textuelle.

```
IdentificationRuleRepository returns IdentificationRuleRepository:
    {IdentificationRuleRepository}
    'IdentificationRuleRepository' '{'
        ( allRules+=Rule ( allRules+=Rule)* )?
        ( allConstraints+=Constraint ( allConstraints+=Constraint)* )?
    '}';

Rule returns Rule:
    'Rule' name=EString '{'
        ('description' ':' description=EString';')?
        ('references' ':' references+=EString (',' references+=EString)* ';')?
        'content' ':' content=Expression';'
        '}';

Constraint returns Constraint: 'Constraint' name=EString ':' relation=Relation ';';

Relation returns Relation: Requires | Conflicts |Equivalent;

Expression returns Expression:
    {Expression}
        (userDefinedDomains+=UserDefinedDomain ( "," userDefinedDomains+=UserDefinedDomain)* ';')?
        (ownedVariables+=Variable ( "|" ownedVariables+=Variable)* '|')? proposition=BooleanProposition;

Variable returns Variable:
    {Variable}
    (( quantifier=Quantifier)? name=EString 'in'
          (defaultDomain=PredefinedSet|containingDomain=[UserDefinedDomain|EString])
        |(( quantifier=Quantifier)?  name=EString ':' componentType = ComponentType));
```

```
BooleanProposition returns Proposition :
   {Proposition}
   unaryFunction=Not'('operand=BooleanProposition')'
      | '(' leftOperand=BooleanProposition arithmeticOperator=(And|Or|Implies)
           rightOperand=BooleanProposition')'
      |leftOperand=NonBooleanProposition
         arithmeticOperator=(Is|Diff|Eq|Geq|Gt|Lt|Leq) rightOperand= NonBooleanProposition
      |binaryFunction = (TypeOf | GetProperty | PropertyExist)
           '('leftOperand=VariableAccessor','rightOperand=PropertyType ')'
      |binaryFunction = (Member | Is_Bound_To | Is_Direct_Subcomponent_Of | Is_Subcomponent_Of)
           '('leftOperand=VariableAccessor','rightOperand=VariableAccessor')';

NonBooleanProposition returns Proposition :
   {Proposition}
   (unaryFunction=(Source|Destination|Owner) '('operand=(VariableAccessor)')'
      |unaryFunction=Cardinal '('operand=(UserDefinedSetAccessor)')'
      |binaryFunction=BinaryFunction'('leftOperand=VariableAccessor','rightOperand=PropertyType')'
      |operand=(Constant|Accessor|Property)
      |leftOperand=NonBoolOperand arithmeticOperator=(Add|Subtract|Multiply|Divide)
           rightOperand=NonBoolOperand);

NonBoolOperand returns Proposition:
   {Proposition}
   (unaryFunction=(Source|Destination|Owner) '('operand=(VariableAccessor)')'
      |unaryFunction=Cardinal '('operand=(UserDefinedSetAccessor)')'
      |binaryFunction=BinaryFunction'('leftOperand=VariableAccessor','rightOperand=PropertyType')'
      |operand=(Constant|Accessor|Property));

ComponentType returns ComponentType:
   Connection| Bus | Memory | Switch | Task |Process | Router | Processor | MutualExclusionResource |
        CommunicationResource | Port | System |Scheduler;
```

Listing 6.1 – Extrait de la syntaxe concrète en Xtext

Le listing 6.2 présente un exemple d'une règle d'identification exprimée en IRL.

```
forAll t : Task | TypeOf(t,Periodicity) is Periodic;
```

Listing 6.2 – Une règle d'identification exprimée en IRL

L'hypothèse exprimée dans le listing 6.2 signifie que "toutes les tâches du système sont périodique". La règle contient une variable caractérisée par un quantificateur `forAll`, un nom est t et elle est un élément de l'ensemble de `Task`. La proposition est une instance de la classe `BooleanProposition`, elle se compose d'un opérateur principal (i.e., is) et deux arguments. Le premier argument a une instance de fonction binaire : `TypeOf` avec deux opérandes (i.e., t et le type de propriété `Periodicity`, le deuxième argument est le type de la propriété `Periodic`. Cette règle d'identification est alors beaucoup plus facile à lire par rapport à son expression en OCL avec les concepts de MoSaRT (voir le listing 6.3).

```
SbTimeTrigger.allInstances()->forAll(t:SbTimeTrigger|t.rtpPeriodicity
                              .oclIsTypeOf(MoSaRT::RealTimeProperties::RtpTypes::RtpPeriodicType))
```

Listing 6.3 – Exemple d'une hypothèse exprimée en OCL sur le méta-modèle MoSaRT

6.5 Le référentiel d'analyse générique

6.5.1 Formalisation

Dans cette section, nous introduisons la structure du méta-modèle du référentiel d'analyse. Un référentiel d'analyse $\mathcal{A}_{rep} = <\mathcal{R}, \mathcal{C}, \mathcal{T}>$, tel que

— $\mathcal{T} = \{t_1, t_2, .., t_n\}$ est un ensemble de tests proposés dans la littérature.

— $\mathcal{R} = \{r_1, r_2, .., r_n\}$ est l'union de toutes les règles d'identification exprimées en IRL, définissant $\mathcal{C}$. $\mathcal{R}$ doit être évalué sur chaque système. Le résultat d'évaluation d'une règle peut être vrai ou faux, et dépend du système étudié.

— $\mathcal{C} = \{c_1, c_2, .., c_n\}$ est un ensemble de contextes (domaine d'application d'un test). Chaque contexte $c_i = < G_i, T_i >$. G_i est un groupe racine, qui contient des sous-groupes, reliant des règles d'identification. Chaque (sous-)groupe est caractérisé par l'attribut *junctionType* qui peut être *and* ou *or*. En outre, $T_i \subset \mathcal{T}$ est un sous-ensemble de tests qui peuvent être appliqués dans le contexte c_i.
$G_i =< G_{ij}, \mathcal{R}_i^{true}, \mathcal{R}_i^{false}, \mathcal{R}_i^{neutral} >$ dont G_{ij} est un ensemble de sous-groupes directs du groupe G_i, G_{ij} lui-même peut contenir des sous-groupes reliant des règles. $\mathcal{R}_i^{true}$ est l'ensemble de règles du groupe G_i dont la valeur attendue est vraie, $\mathcal{R}_i^{false}$ est l'ensemble de règles du groupe G_i dont la valeur attendue est fausse, $\mathcal{R}_i^{neutral}$ est l'ensemble de règles du groupe G_i dont la valeur attendue peut être soit vraie soit fausse. Si le groupe est du type *and*, cela veut dire qu'au moment de l'évaluation toutes les règles du groupe doivent avoir des valeurs d'évaluation qui correspondent aux valeurs attendues. Autrement dit, $\forall r \in \mathcal{R}_i^{true} \mid c_i \models r$ **et** $\forall r \in \mathcal{R}_i^{false} \mid c_i \not\models r$. Si le groupe est du type *or*, il suffit d'avoir au moins une règle respectant la valeur attendue. Formellement, $\exists r \in \mathcal{R}_i^{true} \mid c_i \models r$ **ou** $\exists r \in \mathcal{R}_i^{false} \mid c_i \not\models r$. La présence des groupes de type *and*, *or* permet d'éviter la redondance des règles et de mieux gérer la variabilité des contextes $\mathcal{C}$.

6.5.2 Méta-modèle du référentiel d'analyse

La structure générale du référentiel d'analyse est présentée dans la figure 6.12. Elle reflète la formalisation présentée ci-dessus.

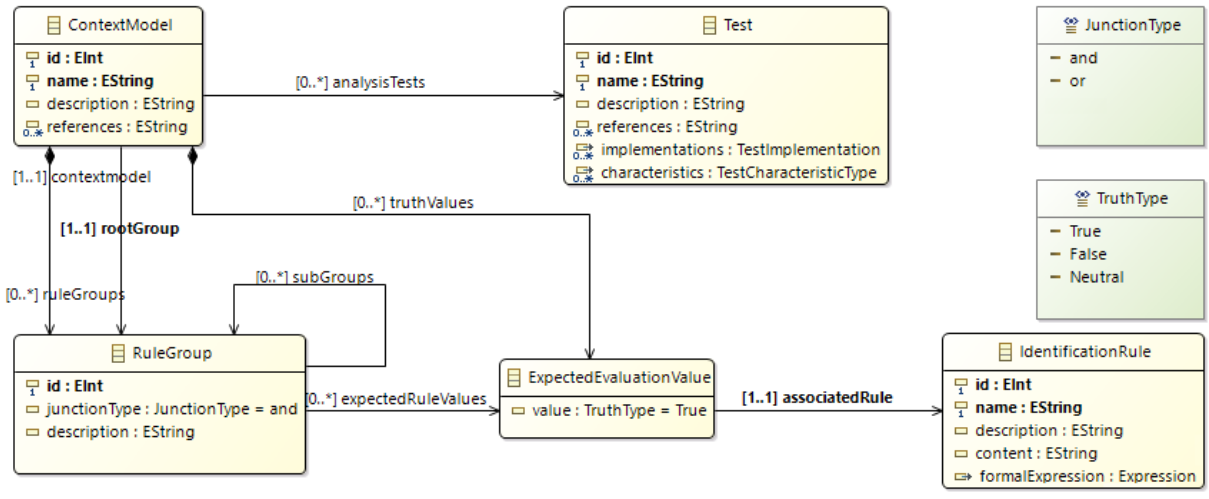

FIGURE 6.12 – Les concepts principaux du méta-modèle du référentiel d'analyse

Nous avons aussi développé une syntaxe concrète textuelle pour le référentiel d'analyse à l'aide de Xtext. En raison de la complexité, toutes les règles d'identification dans l'exemple ci-dessous sont reliées par la relation *AND*. La figure 6.13 présente un exemple de référentiel

d'analyse. Il contient trois contextes avec différents types de tests (dimensionnement et validation) issus de trois articles appartenant à trois différentes décennies (analyse de temps de réponse 1986, affectation de priorités 1991, analyse de sensibilité 2008) ainsi qu'un ensemble de règles d'identification.

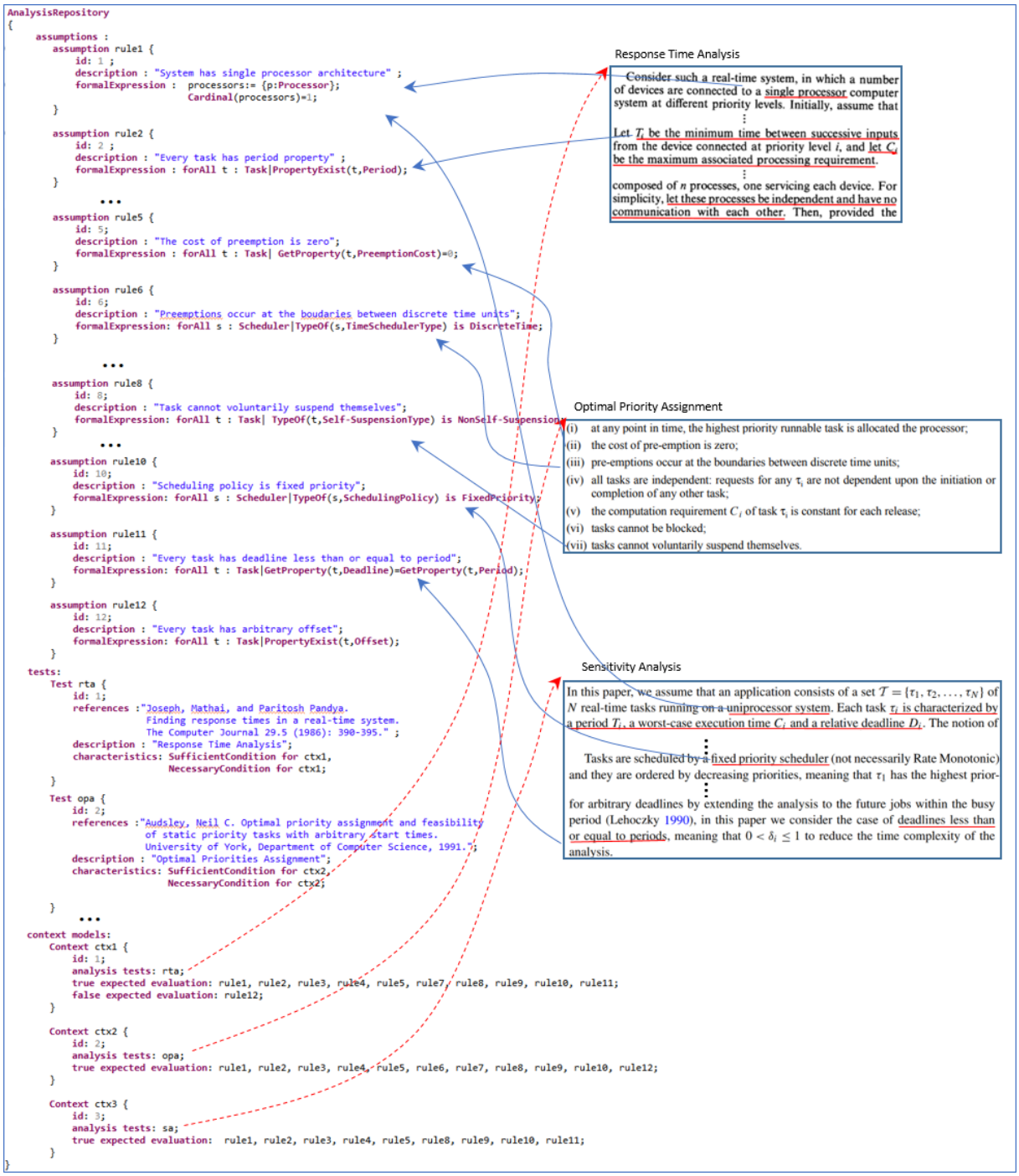

FIGURE 6.13 – Exemple d'un référentiel d'analyse

6.5.3 Fiabilité du référentiel d'analyse

Quand le nombre de règles existantes d'un référentiel d'analyse devient grand, il devient difficile de gérer la valeur attendue de nouvelles règles par rapport à différents contextes. Pour rendre alors le référentiel d'analyse cohérent, fiable et aider les utilisateurs à étendre un référentiel en ajoutant de nouvelles règles, nous avons identifiés trois possibles relations entre les règles (voir la figure 6.14) que nous expliquons ci-après :

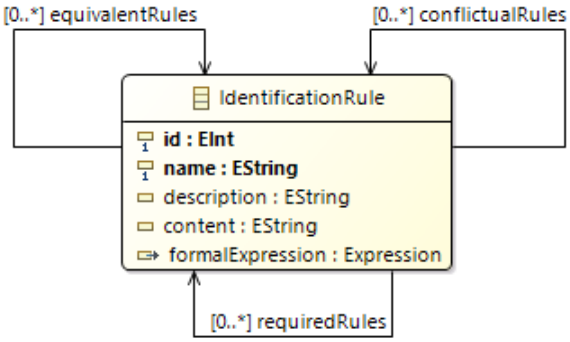

FIGURE 6.14 – Méta-modèle du référentiel d'analyse : focus sur la classe règle d'identification et ses relations réflexives

— Une règle (r_1) peut être équivalente à une autre règle (r_2). On note $r_1\, Eq\, r_2$ (voir la *equivalentRules* de la figure 6.14). Par exemple, les hypothèses "Pour toutes les tâches, l'échéance relative est inférieure ou égale à la période" et "Pour toutes les tâches, la période est supérieure ou égale à l'échéance relative" sont équivalentes. Deux règles équivalentes doivent avoir le même valeur attendue par rapport un contexte donné.

— Une règle (r_1) peut être contradictoire avec une autre règle (r_2). On note $r_1\, Con\, r_2$ (voir la *conflictualRules* de la figure 6.14). Par exemple, les hypothèses "Pour toutes les tâches, l'échéance relative est inférieure ou égale à la période" et "Existe au moins une tâche dont l'échéance relative est supérieure à la période" sont contradictoires. Les valeurs attendues, de deux règles contradictoires qui cohabitent un contexte donné, doivent être opposées.

— Une règle (r_1) peut requérir une autre règle (r_2). On note $r_1\, Req\, r_2$ (voir *requiredRules* de la figure 6.14). Par exemple, l'hypothèse "Pour toutes les tâches, l'échéance relative est équivalente à la période" requiert les hypothèses "Toutes les tâches ont la propriété l'échéance relative" et "Toutes les tâches ont la propriété la période".

Les manipulations d'administration du référentiel d'analyse seront effectuées par un analyste. Nous listons ci-dessous les propriétés à respecter :

— Les relations d'équivalence et de contradiction sont réflexives : $r_1\, Eq\, r_2 \implies r_2\, Eq\, r_1$ et $r_1\, Con\, r_2 \implies r_2\, Con\, r_1$.

— Transitivité : si une règle (r_1) requiert une règle (r_2) et que la règle (r_2) requiert une autre règle (r_3) alors la règle r_1 requiert aussi la règle r_3 : $r_1\, Req\, r_2$ et $r_2\, Req\, r_3 \implies r_1\, Req\, r_3$.

— Si la règle (r_1) est contradictoire avec la règle (r_2) alors la règle r_1 est contradictoire avec toutes les règles équivalentes à la règle r_2 : $r_1\, Con\, r_2$ et $r_2\, Eq\, r_3 \implies r_1\, Con\, r_3$.

— Pour un contexte, une règle d'identification doit avoir la même valeur attendue que ses règles équivalentes.

— Pour un contexte, si la valeur attendue d'une règle d'identification est l'inverse de la valeur attendue de ses règles contradictoires.

— Si la valeur attendue d'une règle est vraie, la valeur attendue de ses règles requises doit être aussi vraie.

Les propriétés décrites ci-avant ont été implémentées sous forme d'invariants afin d'enrichir le métamodèle du référentiel d'analyse et renforcer sa sémantique statique. La figure 6.15 représente cette implémentation en OCL.

```
invariant inv1 : IdentificationRule.allInstances()
                    ->forAll(r1,r2:IdentificationRule|r1.equivalentRules->includes(r2)
                    implies r2.equivalentRules->includes(r1));
invariant inv2 : IdentificationRule.allInstances()
                    ->forAll(r1,r2:IdentificationRule|r1.conflictualRules->includes(r2)
                    implies r2.conflictualRules->includes(r1));
invariant inv3 : IdentificationRule.allInstances()
                    ->forAll(r1,r2:IdentificationRule|r1.conflictualRules->includes(r2)
                    implies r1.conflictualRules->includesAll(r2.equivalentRules));

invariant inv4 : IdentificationRule.allInstances()
            ->forAll(r1,r2 | (r1.equivalentRules->includes(r2) implies
                self.truthValues->select(tv| tv.value=TruthType::True)
            ->collect(ex: ExpectedEvaluationValue| ex.associatedRule)->includesAll(r1,r2)
            or  self.truthValues->select(tv|tv.value=TruthType::False)
            ->collect(ex:ExpectedEvaluationValue|ex.associatedRule)-> includesAll(r1,r2)
            or self.truthValue->select(tv|tv.value=TruthValue::Neutral)
            ->collect(ex:ExpectedEvaluationValue|ex.associatedRule)->includesAll(r1,r2)));
```

FIGURE 6.15 – Implémentation des propriétés en OCL

6.6 Transformation et adaptation

A ce stade, tout référentiel d'analyse peut être vu comme une collection de travaux existants ou un corpus des analyses temporelles des systèmes temps réel. Afin de pouvoir appliquer un référentiel d'analyse aux cas pratiques modélisés par des DSLs du domaine, une adaptation et transformation des règles d'identifications du référentiel d'analyse s'imposent selon le langage cible (MARTE, AADL, MoSaRT, etc.). Nous avons mis en place ce processus de transformation pour supporter le langage AADL ainsi que Time4Sys. Cette action donne lieu aux *Performance Analysis Repository for AADL Designs - PARAD* (pour AADL) et *Time4sys-Compliant Analysis Repository - TyscoAR* (pour Time4Sys). Certaines règles de correspondance sont listées ci-dessous dans les tableaux suivants : le tableau 6.1 et le tableau 6.2.

6.6.1 Exemple d'utilisation de PARAD

Une fois le contenu d'un référentiel d'analyse est adapté pour supporter un langage de modélisation comme AADL, les modèles peuvent être soumis à ce référentiel (instance de PARAD) afin d'aider/guider les concepteurs à identifier le test d'analyse le plus adéquat. Nous rappelons ici que la fiabilité et la précision de ce processus dépend de la richesse du référentiel d'analyse en termes de règles d'identification ainsi que leur exactitude (cette dernière peut être consolidée grâce aux relations entre les règles : équivalence, contradiction, etc.). Nous présentons par la suite deux scénarios d'usage de PARAD.

6.6.1.1 Le premier scénario d'usage du référentiel d'analyse PARAD

La figure 6.16 représente le premier scénario d'usage de PARAD.

En effet, le premier scénario d'usage de PARAD se compose de trois étapes. La première étape consiste à créer/enrichir des référentiels d'analyses (instances de PARAD), elle est dédiée aux analystes et chercheurs (processus (1) dans la figure 6.16). Un référentiel d'analyses peut être créé par un ou plusieurs analystes. Il représente donc les connaissances et l'expertise que les analystes veulent partager avec d'autres collaborateurs. La deuxième étape est dédiée aux concepteurs, ils conçoivent leurs systèmes en AADL (processus (2) dans la figure 6.16). Après avoir conçu les

TABLEAU 6.1 – Mapping entre IRL et AADL

Identification Rule Language	**AADL**
System	les éléments de type *SystemInstance*
Memory	*ComponentInstance* category=”memory”
Processor	*ComponentInstance* category=”processor”
Bus	*ComponentInstance* category=”bus”
Priority	propriété *Priority*
Offset	propriété *Dispatch_Offset*
SchedulingPolicy	propriété *Scheduling_Policy*
Deadline	propriété *Deadline*
Jitter	propriété *Dispatch_Jitter*
Period	propriété *Period*
Periodicity	propriété *Dispatch_Protocol*
ExecutionTime	propriété *Compute_Execution_Time*
EarliestDeadlineFirst	propriété *EDF*
LeastLaxityFirst	propriété *LLF*
FixedPriority	propriété *POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL*

TABLEAU 6.2 – Mapping entre IRL et Time4Sys

Identification Rule Language	**Time4Sys**
Processor	*hrm : :HardwareProcessor*
Memory	*hrm : :HardwareMemory*
Scheduler	*grm : :Scheduler*
MutualExclusionResource	*srm : :SoftwareMutualExclusionResource*
Offset	attribut *phase* des éléments de type *gqam : :PeriodicPattern*
SchedulingPolicy	attribut *policy* des éléments de type *grm : :SchedulingPolicy*
Deadline	paramètre qui s'appele *deadline* dans l'attribut *schedParams* des éléments de type *grm : :SchedulableResource*
Jitter	attribut *jitter* des éléments de type *gqam : :ArrivalPattern*
Period	attribut *period* des éléments de type *gqam : :PeriodicPattern*
Periodicity	genre de l'élément *gqam : :ArrivalPattern*, peut être *PeriodicPattern*, *SporadicPattern* ou *AperiodicPattern*
ExecutionTime	attribut *worstCET* des éléments de type *gqam : :Step*
EarliestDeadlineFirst	*grm : :SchedPolicyKind : :EarliestDeadlineFirst*
LeastLaxityFirst	*grm : :SchedPolicyKind : :LeastLaxityFirst*
FixedPriority	*grm : :SchedPolicyKind : :FixedPriority*

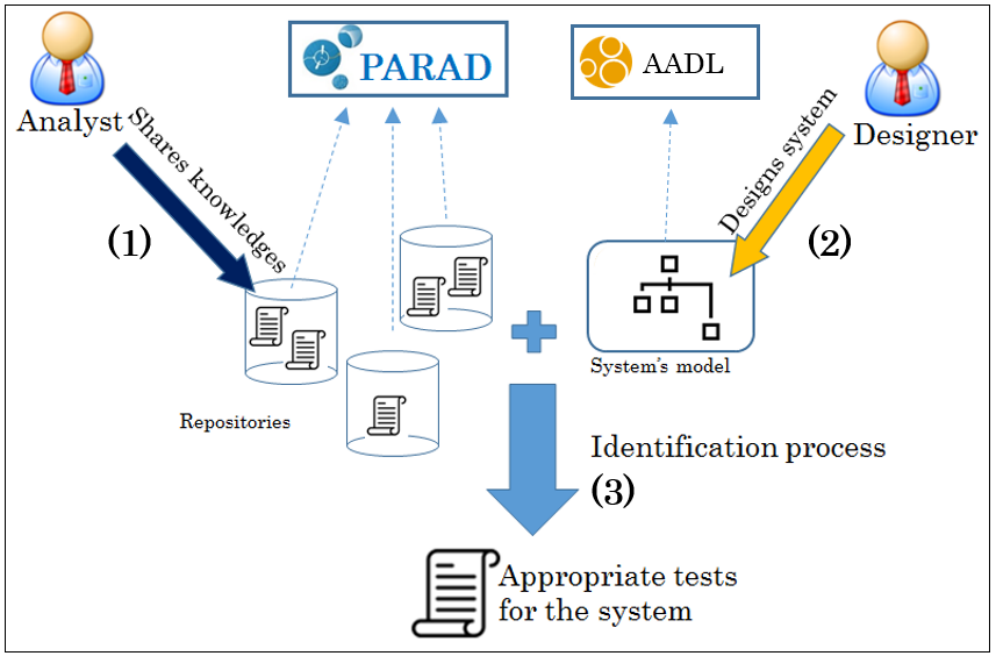

FIGURE 6.16 – Utilisation de PARAD

systèmes en AADL, les concepteurs lancent le processus d'identification (processus (3) dans la figure 6.16) qui fouille dans les référentiels d'analyses (fournis par des analystes et des experts) afin de trouver les tests appropriés pour leurs systèmes.

Nous détaillons le processus d'identification par les algorithmes suivants. L'algorithme central du processus d'identification est le pseudo algorithme 2, il appelle le résultat d'évaluation des règles depuis le pseudo algorithme 1 et appelle la fonction *groupCheck* depuis le pseudo algorithme 4 qui à son tour, appelle la fonction *isMatched* depuis le pseudo algorithme 3.

Algorithm 1: Évaluation des règles d'identification

```
INPUT
    Ar : Analysis Repository;
    M : A valid analytical model of a real-time system;
OUTPUT
    Evaluation results;
Loading and evaluating all identification rules R existing in Ar;
Let R_M^true =R_M^false =R_M^undef ={};
for r_i ∈ R do
    if M ⊨ r_i then
        R_M^true ← R_M^true ∪ {r_i};
    else
        if M ⊨ ¬r_i then
            R_M^false ← R_M^false ∪ {r_i};
        else
            R_M^undef ← R_M^undef ∪ {r_i};
        end
    end
end
```

Algorithm 2: Chercher les contextes appropriés

```
1  INPUT
2    Ar : Analysis Repository;
3    Rs : list of evaluated results got from the previous algorithm 1;
4  OUTPUT
5    X_M : The appropriate real-time contexts;
6  Let C : all contexts from the analysis repository;
7  for ctx ∈ C do
8  |   Let rGr : root group of the context ctx;
9  |   if groupCheck(rGr,Rs) then
10 |   |   X_M ← X_M ∪ ctx
11 |   else
12 |   |   do nothing
13 |   end
14 end
```

Algorithm 3: Fonction *isMatched*

```
1  INPUT
2    e : expected value;
3    rs : associated evaluated result from the algorithm 1;
4  OUTPUT
5    Boolean : the evaluated result matches its expected value or not;
6  if e = TRUE and rs = TRUE then
7  |   return true;
8  else
9  |   if e = FALSE and rs = FALSE then
10 |   |   return true;
11 |   else
12 |   |   if e = NEUTRAL then
13 |   |   |   return true;
14 |   |   else
15 |   |   |   return false;
16 |   |   end
17 |   |   return false;
18 |   end
19 |   return false;
20 end
```

Algorithm 4: Fonction *groupCheck*

```
INPUT
  rGr : group of identification rules;
  Rs : list of evaluated results got from the previous algorithm 1;
OUTPUT
  Boolean : The group is valid or not;
Let junctionType : junction type of the group;
Let subGroups : list of direct sub-group of the group ;
Let expectedValues : expected values list of the context;
if junctionType == AND then
    // All direct identification rules must respect their expected value
    for each e_i ∈ expectedValues do
        Let rs_i : evaluated result of rule r_i in Rs;
        if not isMatched(e_i, rs_i) then
            return false
        else
            do nothing
        end
    end
    // All sub-group must be valid
    for each g_j ∈ subGroups do
        if not groupCheck(g_j, Rs) then
            return false
        else
            do nothing
        end
    end
    return true;
else
    if junctionType == OR then
        // Exist at least a rule respecting its expected value or a valid group
        count ← 0
        for each e_i ∈ expectedValues do
            Let rs_i : evaluated result of rule r_i in Rs;
            if isMatched(e_i, rs_i) then
                count++
            else
                do nothing
            end
        end
        for each g_j ∈ subGroups do
            if groupCheck(g_j, Rs) then
                count++
            else
                do nothing
            end
        end
        if count > 0 then
            return true
        else
            return false
        end
    else
        do nothing
    end
end
```

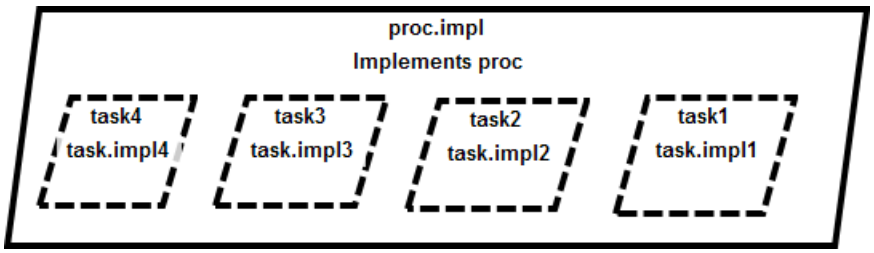

FIGURE 6.17 – Système de tâches en AADL

Nous illustrons ce scénario via un simple exemple. Soit S un système temps réel conçu en AADL et se composant de quatre tâches périodiques indépendantes : $S = \tau_i(O_i, C_i, D_i, T_i) = \tau_1(2, 3, 15, 20), \tau_2(0, 4, 8, 23), \tau_3(5, 5, 13, 23), \tau_4(7, 9, 23, 23)$. On suppose l'existence d'un référentiel d'analyse qui se compose de trois contextes. Chaque contexte contient 18 règles d'identification, le contenu des trois contextes est illustré par la figure 6.19.

Suite à l'appel du processus d'identification, le résultat d'évaluation des règles d'identification et les contextes trouvés sont illustrés par la figure 6.18.

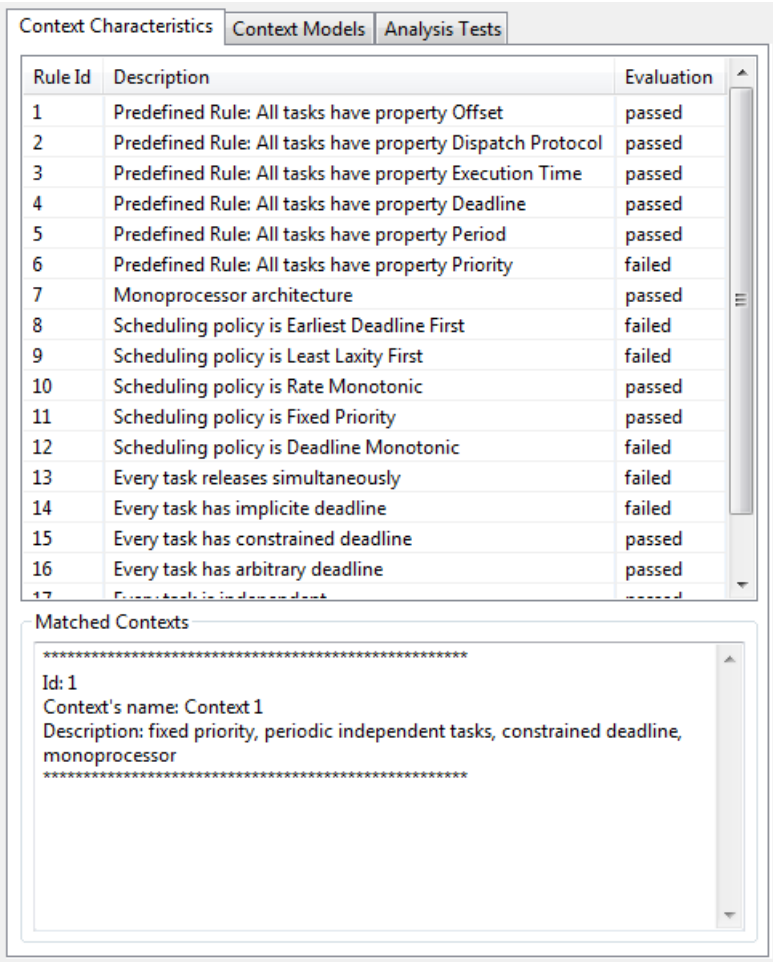

FIGURE 6.18 – Le résultat d'évaluation des règles

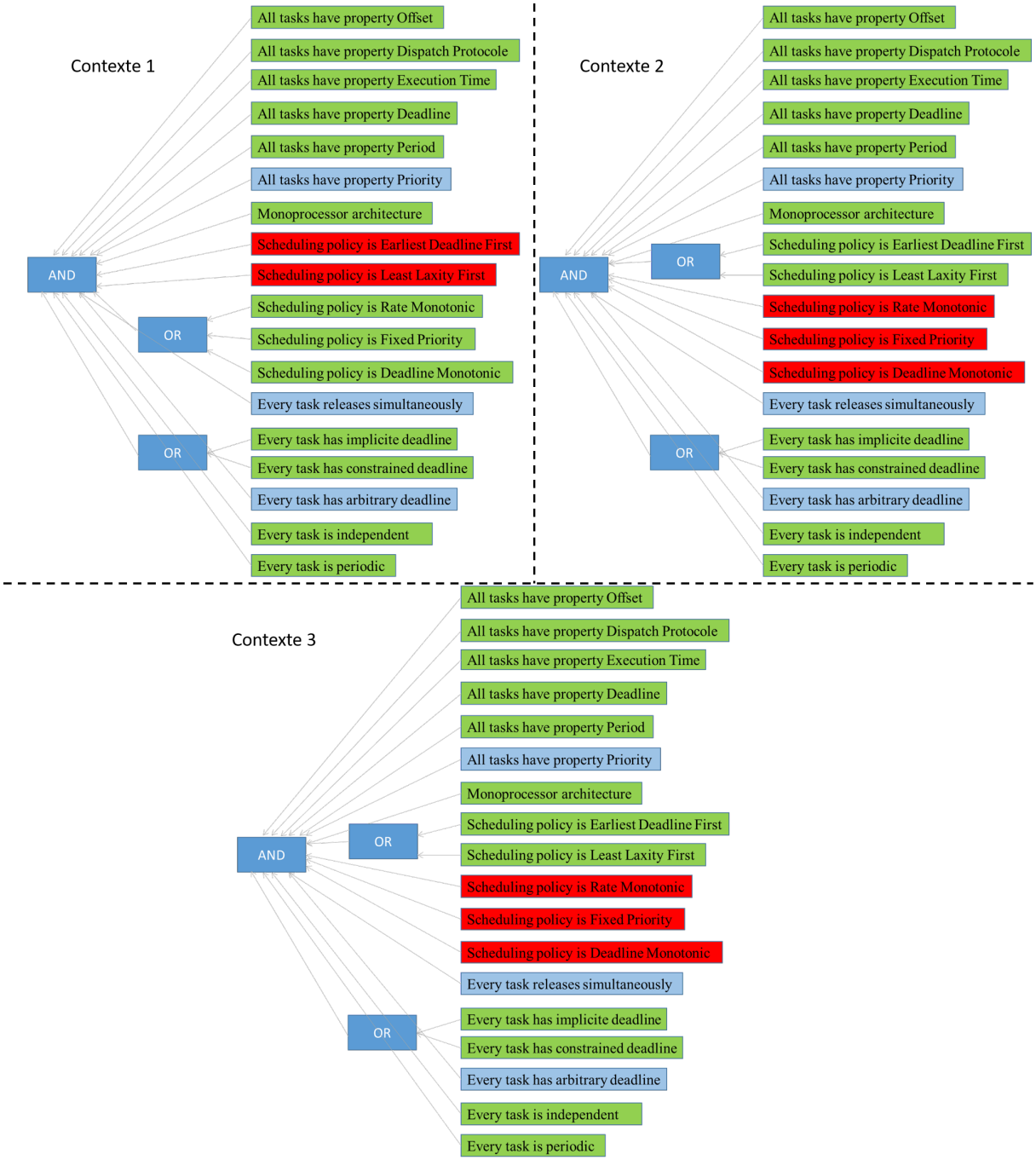

FIGURE 6.19 – Le contenu des contextes

6.6.1.2 Le deuxième scénario d'usage du référentiel d'analyse PARAD

Le deuxième scénario consiste à fixer en amont le contexte souhaité avant de commencer la conception. La conception doit alors respecter toujours les règles qui définissent ce contexte. En effet, il est aussi possible d'appliquer les tests d'analyse qui correspondent au contexte à la volée au moment de la conception. Ce scénario peut être vu comme l'utilisation des design patterns permettant ainsi l'architecte de mettre sa conception dans une sorte de carcan.

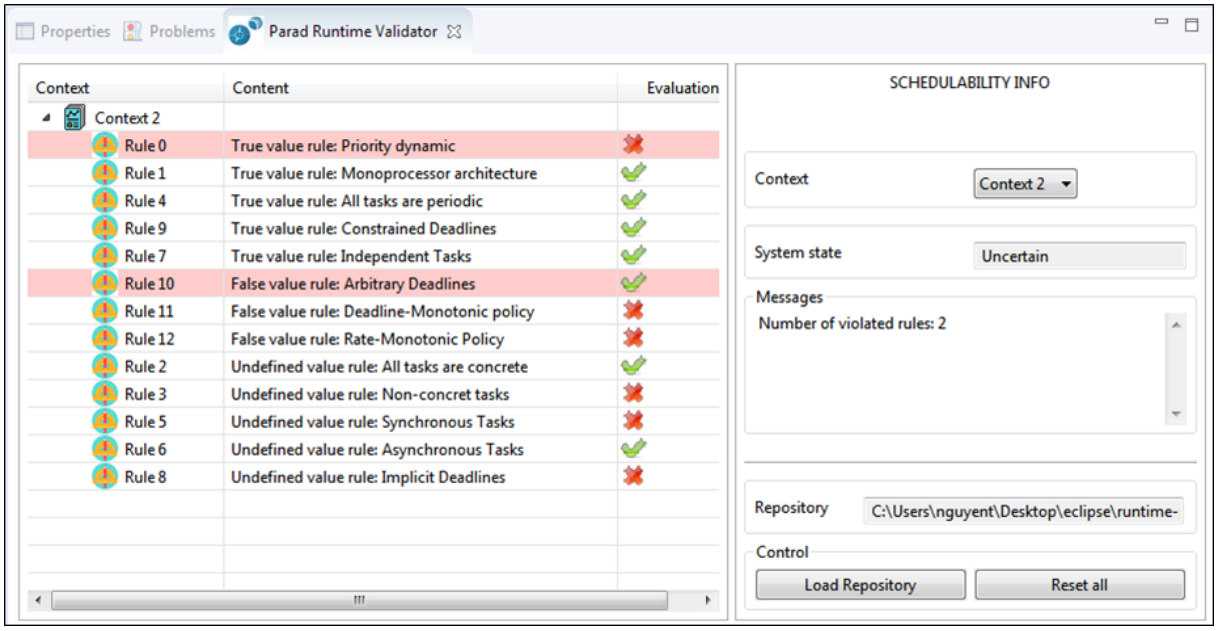

FIGURE 6.20 – Le résultat du deuxième scénario d'usage du référentiel d'analyse

La figure 6.20 présente l'outil correspondant à ce deuxième scénario d'usage du référentiel d'analyse. La partie gauche est l'évaluation des règles d'identification (la troisième colonne) sur le système et la comparaison avec les valeurs attendues (la deuxième colonne) du contexte choisi. Les règles qui ne respectent pas leur valeur attendue sont en rouge (par exemple, la règle 0 et 10 dans la figure 6.20). Le concepteur peut corriger la conception selon les règles violées pour que le système soit toujours conforme au contexte choisi.

6.6.2 Prototypage rapide des tests

Au lieu d'utiliser les tests implémentés dans les outils existants où il faut transformer le modèle en formalisme d'entrée des outils, nous avons pensé à une autre approche apportant plutôt les tests d'analyse à l'environnement de modélisation de AADL. Pour ce faire, nous proposons un framework basé sur les modèles permettant aux analystes et chercheurs du domaine temps réel de prototyper et d'intégrer rapidement leurs tests d'analyse afin d'être partagés et utilisés facilement par les concepteurs.

L'architecture du framework est présentée par la figure 6.21. Elle se compose de trois parties : (i) représentation des formules mathématiques, (ii) mapping entre les artefacts de AADL et les entrées/sorties des formules mathématique et (iii) la génération automatique du code.

Dans la partie de représentation des formules mathématiques (l'action (1) de la figure 6.21), nous nous basons sur MathML (*Mathematical Markup Language* [Wik]), comme c'est un standard pour décrire les notions mathématiques et capturer à la fois leurs structures et leurs contenus. Dans cette partie, l'analyste exprime son test d'analyse d'une manière conforme au méta-modèle MathML. La figure 6.22 représente un exemple d'encodage d'une formule mathématique en utilisant MathML.

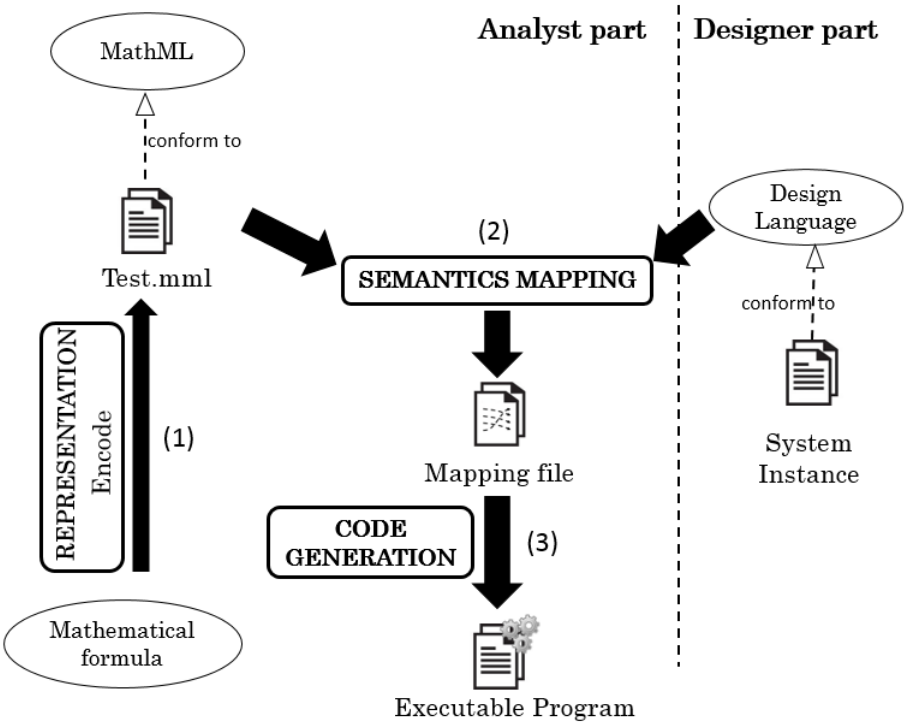

FIGURE 6.21 – Architecture du framework

Dans la deuxième partie (l'action (2) de la figure 6.21), nous associons à chaque composant de la formule mathématique (obtenue dans la première partie) une sémantique en le liant à un élément du langage de modélisation (AADL par exemple). Chaque liaison (ou mapping) est une instance du méta-modèle *Mapping*. Ce dernier est illustré par la figure 6.23. Il se compose des éléments de type *MathComponent*, des éléments de type *Property* et des éléments de type *ComponentType*. Chaque élément *MathComponent* est associé à un élément *Property* et chaque élément *Property* appartient à un élément *ComponentType*.

Une fois le test exprimé sous forme de fichier MathML et que les relations de mapping avec un langage de conception spécifique sont définies, nous pouvons passer à la dernière partie : la génération de code (l'action (3) de la figure 6.21). La génération de code dépend du langage de programmation cible. Dans cette illustration, nous avons choisi le langage Scilab comme langage cible de transformation vu sa syntaxe simple et avons choisi le langage AADL comme langage de conception.

Pour montrer l'utilité de notre framework, nous allons prendre comme exemple la formule mathématique suivante qui représente le test d'analyse du pire temps de réponse [JP86a].

$$R_{i.m} = \begin{cases} C_i & m = 0 \\ C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_{i.m-1}}{T_j} \right\rceil C_j & otherwise \end{cases} \tag{6.1}$$

$$hp(i) = \{k | 1 \leq k \leq n \wedge Pr_k > Pr_i\} \tag{6.2}$$

Le pire temps de réponse est calculé pour chaque tâche i selon l'équation 6.1 où C, T, Pr, hp représentent respectivement le pire temps d'exécution, la période, la priorité et l'ensemble de tâches les plus prioritaires que la tâche i. La figure 6.24 montre les paramètres des formules mathématiques et les concepts associés.

La figure 6.25 représente le résultat final généré automatiquement. Les mappings des concepts servent alors à extraire les données d'entrée depuis le modèle AADL pour faire le calcul et affecter le résultat au données de sortie. Autrement dit, toute modification de la formule (via le

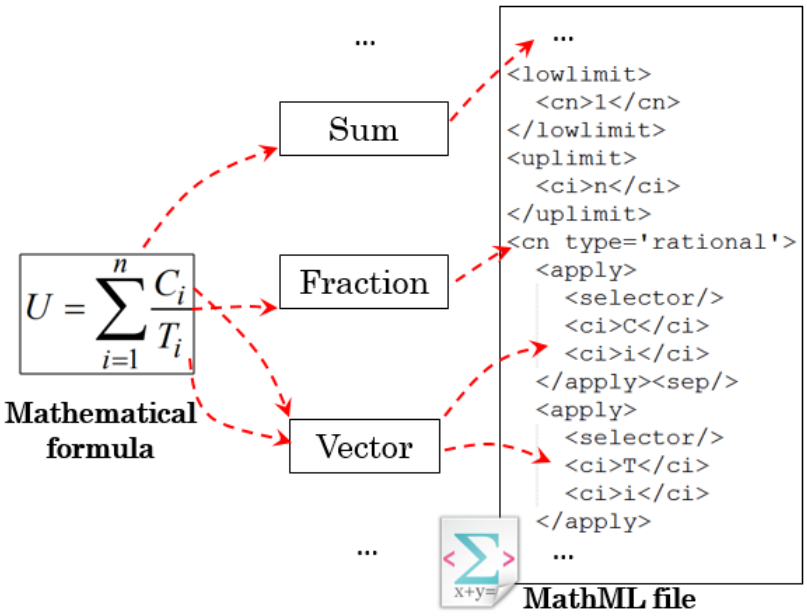

FIGURE 6.22 – Représentation de la formule mathématique en MathML

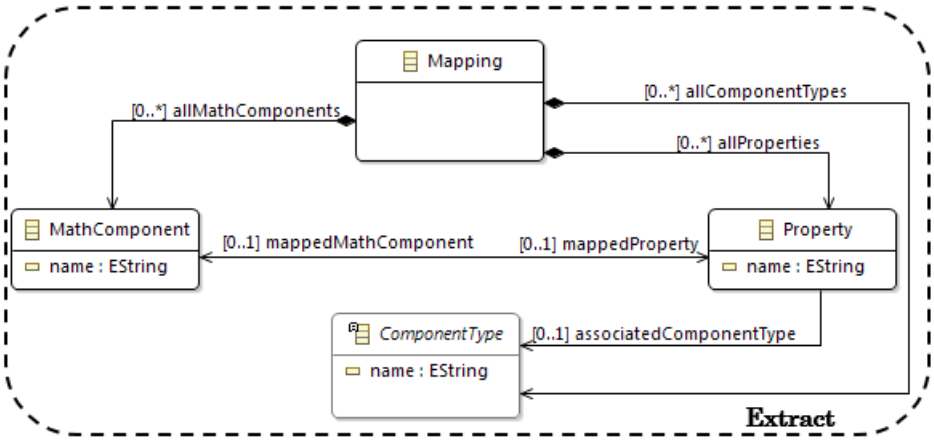

FIGURE 6.23 – Extrait du méta-modèle, exprimé en Ecore [Ecl], des relations de mapping sémantique

fichier MathML) ou de mapping avec les artefacts d'AADL déclenche à nouveau le processus de génération de Scilab.

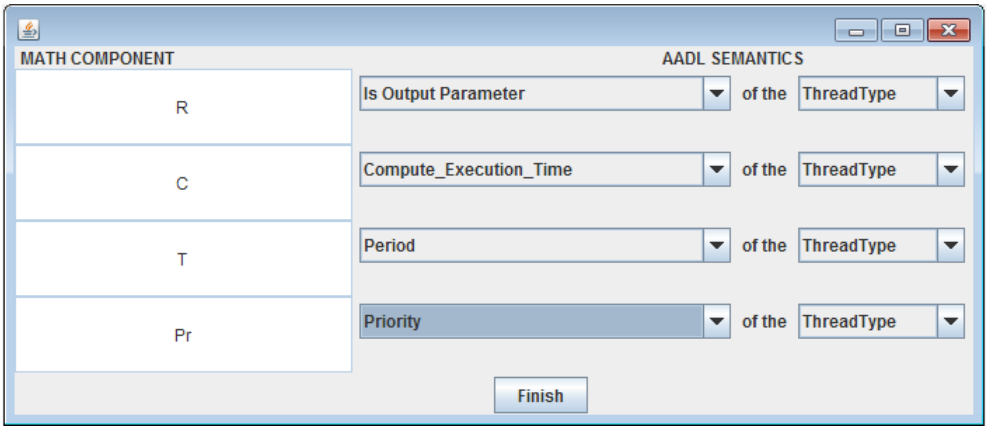

FIGURE 6.24 – Processus de mapping

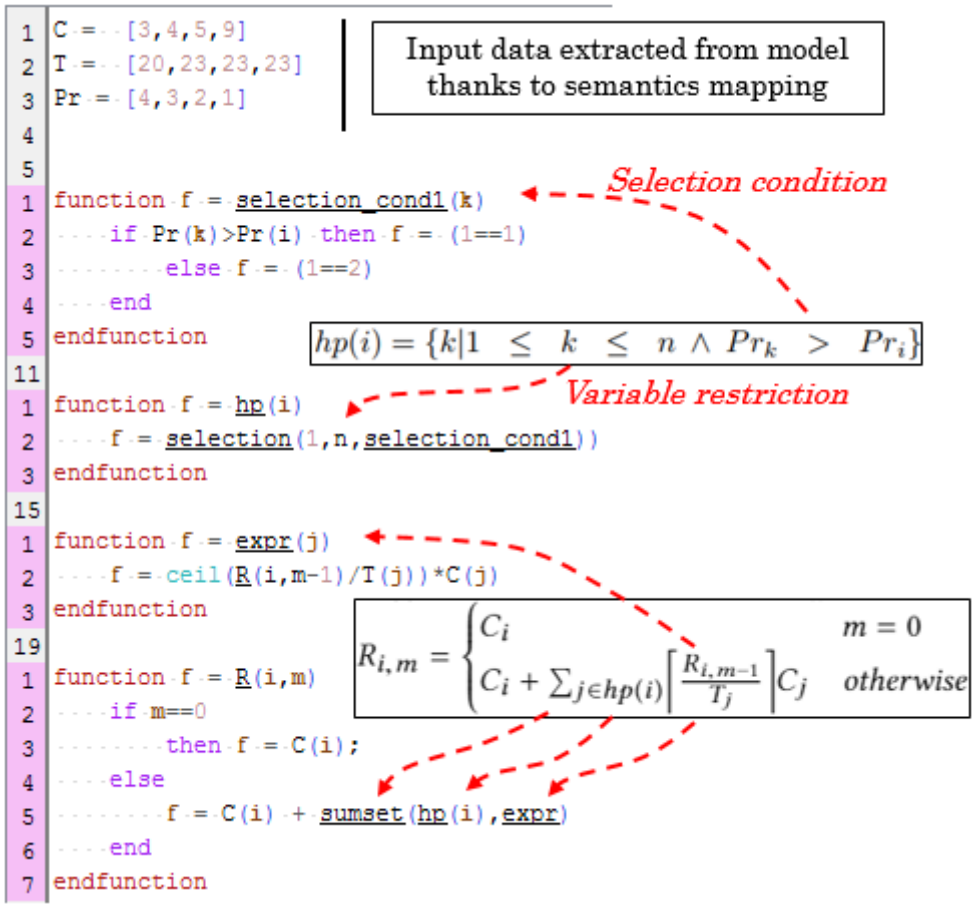

FIGURE 6.25 – Analyse de temps de réponse en Scilab

6.7 Conclusion

Dans ce chapitre, nous avons comparé les différentes approches d'aide à l'analyse temporelle. Nous avons montré l'intérêt et le point faible de chaque approche en privilégiant par la suite le référentiel d'analyse de MoSaRT. Ce dernier se base essentiellement sur règles d'identification qui dépendent du langage de contrainte OCL et reposent directement sur les concepts de MoSaRT design language qui ne sont pas faciles à maîtriser. Pour surmonter ces difficultés, nous avons proposé un DSL (*Domain Specific Language*) nommé IRL (*Identification Rule Language*) indépendant d'OCL et de tout langage de description d'achitecture. La nouveauté du langage

consiste en la syntaxe de prédicat et l'ensemble de concepts extraits directement des analyses. Il est dédié aux analystes, qui peuvent l'utiliser pour écrire leurs règles d'identification indépendamment du langage sous-jacent de description, tout en restant facile à lire et comprendre. Grâce à ce nouveau langage, le référentiel d'analyse devient générique. Nous pensons que l'adoption d'une telle démarche par la communauté temps réel pourrait être bénéfique afin d'avoir une base de données qui ressemble à DBLP (https ://dblp.uni-trier.de/) mais avec des requêtes plus spécifiques au domaine des systèmes temps réel. Pour l'utilisation du référentiel d'analyse avec des langages de modélisation spécifiques, nous n'avons qu'à transformer les règles d'identification depuis IRL vers les langage cibles. A ce jour, il y a deux transformations disponibles : IRL vers AADL et IRL vers Time4Sys.

Troisième partie

Conclusions

Chapitre 7

Conclusion et Perspectives

Sommaire

Ce dernier chapitre résume les contributions de cette thèse. Divers problèmes ont été étudiés à travers cette thèse et ont donné lieu à des contributions liées à la fois à l'analyse d'ordonnançabilité des systèmes et aux approches dirigées par les modèles permettant une utilisation et intégration fluide des tests dans le processus de conception.

7.1 Conclusion

Dans cette thèse, nous avons abordé la modélisation et la vérification temporelle des systèmes temps réel critiques. Les chapitres 2, 3 ont été dédiés à la présentation de l'état de l'art de la thèse. Le chapitre 2 a introduit les notions de base des systèmes embarqués temps réel. En effet, nous avons présenté la structure générale y compris l'architecture logicielle et l'architecture matérielle, ainsi que le système d'exploitation. Ensuite, nous avons présenté les caractéristiques des tests d'ordonnançabilité. Et puis, nous avons présenté la notion de l'ingénierie dirigée par les modèles et son utilisation dans le cycle de développement des systèmes embarqués temps réel à travers des langages de modélisation. Le chapitre 3 a abordé l'aspect de l'analyse d'ordonnancement des systèmes temps réel. Beaucoup de politiques d'ordonnancement et tests associés ont été proposés en fonction de l'architecture matérielle et logicielle. Le cas monoprocesseur a été bien analysé dans la littérature. Les politiques EDF, OPA atteignent l'optimalité, notamment EDF permet une utilisation processeur complète. Quant au cas multiprocesseur, les travaux sur l'ordonnancement partitionné ont réduit le problème multiprocesseur à un ensemble de problèmes monoprocesseurs. Néanmoins, l'ordonnancement partitionné ne permet pas d'atteindre l'optimalité et n'accepte qu'une utilisation processeur relativement faible, contrairement à l'ordonnancement global où les tâches sont autorisées de migrer pendant leur exécution. L'ordonnancement global permet d'atteindre l'optimalité mais il a un point faible : le nombre élevé de préemptions et de migrations. L'ordonnancement semi-partitionné est considéré comme une bonne combinaison entre l'ordonnancement partitionné et l'ordonnancement global en restreignant la migration des tâches. L'ordonnancement semi-partitionné présente une utilisation processeur plus élevée par rapport à l'ordonnancement partitionné et moins de migrations et de préemptions par rapport à l'ordonnancement global. Dans le cas distribué, les systèmes se composent des noeuds inter-connectés à travers des réseaux. L'architecture de chaque noeud est mono/multi-processeur. L'analyse d'ordonnancement des systèmes distribués doit prendre en compte l'architecture des réseaux ainsi que le protocole de gestion des messages sur les réseaux. Nous avons présenté deux réseaux bien utilisés dans la pratique : CAN et AFDX.

Cette thèse présente trois contributions principales :

— La première (voir chapitre 4) contribution porte sur une contrainte de précédence appelée *Semaphore Precedence Constraint - SPC*. Le SPC repose sur le paradigme m-n producteurs/consommateurs. Le comportement des tâches sous contrainte dépend d'un compteur sémaphore initialisé à $h > 0$. Nous avons premièrement renforcé la sémantique des SPC en permettant la valeur négative de h signifiant que certain nombre d'instances de la tâche source doit se produire jusqu'à ce que $h > 0$. Un système de tâches contraintes par les SPC peut être exprimé en utilisant le graphe SPC où chaque noeud représente une tâche et chaque arc représente un SPC. Le graphe SPC peur être déplié en graphe de précédence des instances où chaque noeud représente une instance et chaque arc représente une précédence entre les instances. Initialement, un graphe SPC ne pouvait pas avoir de cycles, nous avons constaté que dans le cas des politiques d'ordonnancement fixes aux travaux, il est possible d'avoir des cycles dans le graphe SPC tant que le graphe déplié ne contient pas de cycles. Et puis, nous avons implémenté SPC ainsi que ses tests d'ordonnançabilité en AADL pour supporter la communication multi-périodique.

— La deuxième contribution (voir le chapitre 5) porte sur un test exact d'ordonnançabilité pour un système de tâches dépendantes sur architecture monoprocesseur ou multiprocesseur. En effet, sur architecture monoprocesseur, nous avons proposé une modélisation à l'aide de réseau de Petri temporel des systèmes de tâches partageant des ressources critiques dans deux cas : sans protocole de gestion et sous le protocole PCP immédiat. Pour l'architecture multiprocesseur, nous n'avons proposé qu'une modélisation des systèmes de tâches partageant les ressources critiques sans protocole de gestion en raison de l'absence de protocole implémenté dans les RTOS malgré quelques protocoles proposés dans la lit-

térature. Nous avons appliqué le principe de modélisation pour transformer les systèmes Time4Sys en Roméo pour estimer les temps de réponse des tâches.

— La troisième contribution (voir le chapitre 6) porte sur le référentiel d'analyse dont la pierre angulaire est la notion des règles d'identification qui représentent les hypothèses des tests d'analyse. Les règles d'identification du référentiel d'analyse, telles que proposées avant, dépendaient du langage de contrainte OCL et de l'ensemble des artefacts des langages de modélisation. Premièrement, nous avons proposé un langage (*Identification Rule Language* - IRL) orienté analyse pour exprimer les règles d'identification indépendamment d'OCL et de tous les langages de conception des systèmes. IRL bénéficie de la structure des prédicats et permet aux analystes de trouver les concepts proches de leur domaine de maîtrise. Deuxièmement, nous avons associé le langage IRL au référentiel d'analyse pour que ce dernier devienne générique. Pour pouvoir utiliser le référentiel d'analyse dans les cas pratiques, il est nécessaire de transformer les règles d'identification de IRL vers OCL avec les concepts du langage cible. Nous avons montré la faisabilité avec le langage Time4Sys et le langage AADL. En utilisant ce dernier comme exemple, nous avons aussi poussé la réflexion plus loin et proposé un outil du prototypage rapide des tests d'analyse. Autrement dit, grâce au prototypage rapide des tests le concepteur n'aurait plus besoin de transformer systématiquement son modèle vers des formalismes des outils externes, évitant ainsi le gap sémantique qui pourrait y avoir entre le modèle et les outils.

7.2 Perspectives

— Actuellement, nous détectons les *deadlocks* dans le graphe déplié en utilisant l'algorithme de Kahn [Kah62] dont la complexité est $O(N+A)$, N représente le nombre de noeuds et A représente le nombre d'arcs dans le graphe. Cependant, la largeur du graphe déplié correpond quelques fois à l'hyper-période. La complexité de l'algorithme est donc exponentielle. SPC est un cas particulier de SDF (*Synchronous Data Flow* [LM87]). On a développé pour SDF un critère algébrique pour assurer l'absence de dépendances cycliques des données (i.e. *deadlocks*) donc la première perspective est d'appliquer ce critère pour le graphe de SPC.

— Traitement Automatique des Langues (TAL) est un domaine de recherche multidisciplinaire incluant la linguistique, l'informatique et l'intelligence artificielle qui vise au traitement de la langue naturelle. Nous pouvons utiliser le TAL pour extraire les informations depuis les articles scientifique afin d'alimenter le référentiel d'analyse.

— La transformation de règles IRL vers OCL peut être améliorée. Le problème survient du fait que IRL est un langage descriptif alors que OCL est un langage prescriptif. Cette difficulté pourrait être résolue en utilisant une ontologie de référence dont chaque concept est associé aux différents artefacts des langages standards du domaine des systèmes temps réel.

— Le principe du référentiel d'analyse peut être exploité plus en étant appliqué à toutes les étapes de conception avant le déploiement. Un travail dans ce sens a été initié récemment au laboratoire. Une thèse sur ce sujet est en cours.

Bibliographie

[AAD19] AADL. Architecture analysis and design language. `http://www.aadl.info/aadl/currentsite/`, 2019. (Cité en page 33), (Cité en page 105)

[AB90] Neil Audsley and Alan Burns. Real-time system scheduling. 1990. (Cité en page 16)

[AB08] Björn Andersson and Konstantinos Bletsas. Sporadic multiprocessor scheduling with few preemptions. In *2008 Euromicro Conference on Real-Time Systems*, pages 243–252. IEEE, 2008. (Cité en page 48)

[ABB08] Björn Andersson, Konstantinos Bletsas, and Sanjoy Baruah. Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In *2008 Real-Time Systems Symposium*, pages 385–394. IEEE, 2008. (Cité en page 48)

[ABD05] James H Anderson, Vasile Bud, and UmaMaheswari C Devi. An edf-based scheduling algorithm for multiprocessor soft real-time systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 199–208. IEEE, 2005. (Cité en page 47)

[ABJ01] Björn Andersson, Sanjoy Baruah, and Jan Jonsson. Static-priority scheduling on multiprocessors. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*, pages 193–202. IEEE, 2001. (Cité en page 45)

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and computation*, 104(1) :2–34, 1993. (Cité en page 53)

[AFM+02] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times b—a tool for modelling and implementation of embedded systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 460–464. Springer, 2002. (Cité en page 82)

[AJ01] Björn Andersson and Jan Jonsson. Preemptive multiprocessor scheduling anomalies. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, pages 8–pp. IEEE, 2001. (Cité en page 45)

[Alu99] Rajeev Alur. Timed automata. In *International Conference on Computer Aided Verification*, pages 8–22. Springer, 1999. (Cité en page 25), (Cité en page 89)

[AS99] J Anderson and Anand Srinivasan. A new look at pfair priorities. *Submitted to Real-time Systems Journal*, 1999. (Cité en page 46)

[AT06] Björn Andersson and Eduardo Tovar. Multiprocessor scheduling with few preemptions. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*, pages 322–334. IEEE, 2006. (Cité en page 47)

[Aud91] Neil C Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. Citeseer, 1991. (Cité en page 41), (Cité en page 64), (Cité en page 65), (Cité en page 74)

[Baa02] Franz Baader. Basic description logics. In *In [11*. Citeseer, 2002. (Cité en page 107)

[Bak91] Theodore P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1) :67–99, 1991. (Cité en page 44)

[Bar98] Sanjoy K Baruah. Feasibility analysis of recurring branching tasks. In *Proceeding. 10th EUROMICRO Workshop on Real-Time Systems (Cat. No. 98EX168)*, pages 138–145. IEEE, 1998. (Cité en page 22)

[Bar03] Sanjoy K Baruah. Dynamic-and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1) :93–128, 2003. (Cité en page 22)

[Bar10] Sanjoy Baruah. The non-cyclic recurring real-time task model. In *2010 31st IEEE Real-Time Systems Symposium*, pages 173–182. IEEE, 2010. (Cité en page 22)

[BB06] Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 159–168. IEEE, 2006. (Cité en page 21)

[BBB03] Enrico Bini, Giorgio C Buttazzo, and Giuseppe M Buttazzo. Rate monotonic analysis : the hyperbolic bound. *IEEE Transactions on Computers*, 52(7) :933–942, 2003. (Cité en page 23), (Cité en page 40)

[BC07] Theodore P Baker and Michele Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *International Conference On Principles Of DIstributed Systems*, pages 62–75. Springer, 2007. (Cité en page 82), (Cité en page 89)

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

[BCL05] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *International Conference on Principles of Distributed Systems*, pages 306–321. Springer, 2005. (Cité en page 45)

[BCPV96] Sanjoy K Baruah, Neil K Cohen, C Greg Plaxton, and Donald A Varvel. Proportionate progress : A notion of fairness in resource allocation. *Algorithmica*, 15(6) :600–625, 1996. (Cité en page 46)

[BD91] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE transactions on software engineering*, (3) :259–273, 1991. (Cité en page 26)

[BG03] Sanjoy K Baruah and Joël Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE transactions on computers*, 52(7) :966–970, 2003. (Cité en page 45)

[BH06] Hanifa Boucheneb and Rachid Hadjidj. Using inclusion abstraction to construct atomic state class graphs for time petri nets. *International Journal of Embedded Systems*, 2(1-2) :128–139, 2006. (Cité en page 26)

[BHN15] Guillaume Brau, Jérôme Hugues, and Nicolas Navet. A contract-based approach to support goal-driven analysis. In *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*, pages 236–243. IEEE, 2015. (Cité en page xii), (Cité en page 100)

[BKT+06] Krishnakumar Balasubramanian, Arvind S. Krishna, Emre Turkay, Jaiganesh Balasubramanian, Jeff Parsons, Aniruddha S. Gokhale, and Douglas C. Schmidt. Applying model-driven development to distributed real-time and embedded avionics systems. *IJES*, 2(3/4) :142–155, 2006. (Cité en page 5)

[Bla76] Jacek Blazewicz. Scheduling dependent tasks with different arrival times to meet deadlines. In *Proceedings of the International Workshop organized by the Commision of the European Communities on Modelling and Performance Evaluation of Computer Systems*, pages 57–65. North-Holland Publishing Co., 1976. (Cité en page 61)

[BLBA07] Aaron Block, Hennadiy Leontyev, Bjorn B Brandenburg, and James H Anderson. A flexible real-time locking protocol for multiprocessors. In *13th IEEE international conference on embedded and real-time computing systems and applications (RTCSA 2007)*, pages 47–56. IEEE, 2007. (Cité en page 90)

[BLR05] Gerd Behrmann, Kim G Larsen, and Jacob I Rasmussen. Optimal scheduling using priced timed automata. *ACM SIGMETRICS Performance Evaluation Review*, 32(4) :34–40, 2005. (Cité en page 82)

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time petri nets. In *Proceedings IFIP*. Citeseer, 1983. (Cité en page 26)

[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*, pages 168–177. ACM, 2017. (Cité en page 100), (Cité en page 103)

[Boe88] Barry W Boehm. A spiral model of software development and enhancement. *Computer*, (5) :61–72, 1988. (Cité en page 5)

[BRH90] Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-time systems*, 2(4) :301–324, 1990. (Cité en page 24)

[Bur99] Alan Burns. The ravenscar profile. *ACM SIGAda Ada Letters*, 19(4) :49–52, 1999. (Cité en page 84)

[But97] GC Buttazzo. Predictable scheduling algorithms and applications, hard real-time computing systems, 1997. (Cité en page 15)

[BV03] Bernard Berthomieu and François Vernadat. State class constructions for branching analysis of time petri nets. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 442–457. Springer, 2003. (Cité en page 26)

[BW01] Alan Burns and Andrew J Wellings. *Real-time systems and programming languages : Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001. (Cité en page 15)

[C+91] Rene L Cruz et al. A calculus for network delay, part i : Network elements in isolation. *IEEE Transactions on information theory*, 37(1) :114–131, 1991. (Cité en page 51)

[CEP03] Luis Alejandro Cortés, Petru Eles, and Zebo Peng. Modeling and formal verification of embedded systems based on a petri net representation. *Journal of Systems Architecture*, 49(12-15) :571–598, 2003. (Cité en page 25)

[CFH+04] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James H Anderson, and Sanjoy K Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms., 2004. (Cité en page 44)

[CGQ05] Guy Cohen, Stéphane Gaubert, and Jean-Pierre Quadrat. L'algèbre des sandwichs. *Pour la science*, 328 :56–63, 2005. (Cité en page 51)

[CHD14] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. Simso : A simulation tool to evaluate real-time multiprocessor scheduling algorithms. 2014. (Cité en page 53)

[CKS02] Liliana Cucu, Rémy Kocik, and Yves Sorel. Real-time scheduling for systems with precedence, periodicity and latency constraints. In *Proceedings of 10th Real-Time Systems Conference, RTS'02*, 2002. (Cité en page 63)

[Cru91] Rene L Cruz. A calculus for network delay. ii. network analysis. *IEEE Transactions on information theory*, 37(1) :132–141, 1991. (Cité en page 51)

[CSB90] Houssine Chetto, Maryline Silly, and T Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3) :181–194, 1990. (Cité en page 61), (Cité en page 62), (Cité en page 64)

[Dav90] Alan M Davis. *Software requirements : analysis and specification.* Prentice Hall Press, 1990. (Cité en page 5)

[Dav14] Robert I Davis. A review of fixed priority and edf scheduling for hard real-time uniprocessor systems. *ACM SIGBED Review*, 11(1) :8–19, 2014. (Cité en page 15)

[DB11a] Robert I Davis and Alan Burns. Fpzl schedulability analysis. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 245–256. IEEE, 2011. (Cité en page 46)

[DB11b] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4) :35, 2011. (Cité en page 20)

[DB11c] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4) :35, 2011. (Cité en page 103)

[Der74] Michael Dertouzos. Control robotics : The procedural control of physical processeds. In *Proc. IFIP congress*, pages 807–813, 1974. (Cité en page 20), (Cité en page 41), (Cité en page 74)

[EA09] Arvind Easwaran and Bjorn Andersson. Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In *2009 30th IEEE Real-Time Systems Symposium*, pages 377–386. IEEE, 2009. (Cité en page 91)

[EB10] Moritz Eysholdt and Heiko Behrens. Xtext : implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 307–309. ACM, 2010. (Cité en page 109)

[Ecl] Eclipsepedia. Ecore. https ://wiki.eclipse.org/Ecore. [Online ; accessed 25/03/2019]. (Cité en page xii), (Cité en page 32), (Cité en page 107), (Cité en page 124)

[emf] Eclipse modeling framework. `https://www.eclipse.org/modeling/emf/`. [Online ; accessed 20/02/2017]. (Cité en page 32)

[FBB06] Nathan Fisher, Sanjoy Baruah, and Theodore P Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 10–pp. IEEE, 2006. (Cité en page 45)

[FBG+10] Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, and Claire Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 301–310. IEEE, 2010. (Cité en page 64), (Cité en page 65), (Cité en page 66), (Cité en page 67), (Cité en page 68)

[FGPR11] Mien Forget, Emmanuel Grolleau, Claire Pagetti, and Pascal Richard. Dynamic priority scheduling of periodic tasks with extended precedences. In *ETFA2011*, pages 1–8. IEEE, 2011. (Cité en page 64), (Cité en page 65), (Cité en page 68)

[G+02] Object Management Group et al. Uml profile for schedulability, performance, and time specification. *http. ://www. omg. org/cgibin/doc*, 2002. (Cité en page 32), (Cité en page 34)

[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, 2014. (Cité en page xii), (Cité en page 100)

[GCG00a] Emmanuel Grolleau and Annie Choquet-Geniet. Cyclicité des ordonnancements de systèmes de tâches périodiques différées. *RTS*, pages 216–231, 2000. (Cité en page 24)

[GCG00b] Emmanuel Grolleau and Annie Choquet-Geniet. Scheduling real-time systems by means of petri nets. *IFAC Proceedings Volumes*, 33(7) :89–94, 2000. (Cité en page 82)

[GFB03] Joël Goossens, Shelby Funk, and Sanjoy Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-time systems*, 25(2-3) :187–205, 2003. (Cité en page 45), (Cité en page 46)

[GGD+07] Nan Guan, Zonghua Gu, Qingxu Deng, Shuaihong Gao, and Ge Yu. Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. In *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*, pages 263–272. Springer, 2007. (Cité en page 89)

[GGL13] Gilles Geeraerts, Joël Goossens, and Markus Lindström. Multiprocessor schedulability of arbitrary-deadline sporadic tasks : complexity and antichain algorithm. *Real-time systems*, 49(2) :171–218, 2013. (Cité en page 89)

[GLDN01] Paolo Gai, Giuseppe Lipari, and Marco Di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*, pages 73–83. IEEE, 2001. (Cité en page 90)

[GPH12] J Javier Gutiérrez, J Carlos Palencia, and M González Harbour. Response time analysis in afdx networks with sub-virtual links and prioritized switches. *XV Jornadas de Tiempo Real*, pages 1–18, 2012. (Cité en page 51)

[Gra69] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2) :416–429, 1969. (Cité en page 61)

[Gri04] Jérôme Grieu. *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques*. PhD thesis, 2004. (Cité en page 51)

[GRR06] Guillaume Gardey, Olivier H Roux, and Olivier F Roux. State space computation and analysis of time petri nets. *Theory and Practice of Logic Programming*, 6(3) :301–320, 2006. (Cité en page 26)

[GRRR13] Emmanuel GROLLEAU, Michaël RICHARD, Pascal RICHARD, and Frédéric RIDOUARD. Ordonnancement temps réel-ordonnancement réparti. 2013. (Cité en page 51)

[GSP+14] Vincent Gaudel, Frank Singhoff, Alain Plantec, Pierre Dissaux, and Jérôme Legrand. Composition of design patterns : from the modeling of rtos synchronization tools to schedulability analysis. *ACM SIGBED Review*, 11(1) :44–49, 2014. (Cité en page 99)

[Hav68] James W. Havender. Avoiding deadlock in multitasking systems. *IBM systems journal*, 7(2) :74–84, 1968. (Cité en page 43)

[Hes06] Wolfgang Hesse. More matters on (meta-)modelling : remarks on thomas kühne's "matters". *Software and System Modeling*, 5(4) :387–394, 2006. (Cité en page 109)

[HGGM01] M González Harbour, JJ Gutiérrez García, JC Palencia Gutiérrez, and JM Drake Moyano. Mast : Modeling and analysis suite for real time applications. In *Proceedings 13th Euromicro Conference on Real-Time Systems*, pages 125–134. IEEE, 2001. (Cité en page 52)

[Hor74] WA Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1) :177–185, 1974. (Cité en page 41), (Cité en page 74)

[IRC] IRCCyN. Romeo - a tool for time petri nets analysis. `http://romeo.rts-software.org/`. (Cité en page 53)

[JAB+06] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. Atl : a qvt-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 719–720. ACM, 2006. (Cité en page 33)

[Jen81] Kurt Jensen. Coloured petri nets and the invariant-method. *Theoretical computer science*, 14(3) :317–336, 1981. (Cité en page 82)

[Joh74] David S Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3) :272–314, 1974. (Cité en page 45)

[JP86a] Mathai Joseph and Paritosh Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5) :390–395, 1986. (Cité en page 24), (Cité en page 49), (Cité en page 123)

[JP86b] Mathai Joseph and Paritosh Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5) :390–395, 1986. (Cité en page 40), (Cité en page 74)

[JS93a] Kevin Jeffay and Donald Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *1993 Proceedings Real-Time Systems Symposium*, pages 212–221. IEEE, 1993. (Cité en page 24), (Cité en page 69)

[JS93b] Kevin Jeffay and Donald Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *Real-Time Systems Symposium, 1993., Proceedings.*, pages 212–221. IEEE, 1993. (Cité en page 74)

[K+82] Claude Kaiser et al. Exclusion mutuelle et ordonnancement par priorité. 1982. (Cité en page 44)

[Kah62] Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11) :558–562, 1962. (Cité en page 69), (Cité en page 132)

[Kem14] Georges Arnaud Kemayo. *Evaluation et validation des systèmes distribués avioniques*. PhD thesis, Chasseneuil-du-Poitou, Ecole nationale supérieure de mécanique et d'Aérotechnique, 2014. (Cité en page xi), (Cité en page 50), (Cité en page 51)

[Kor03] Richard E Korf. An improved algorithm for optimal bin packing. In *IJCAI*, volume 3, pages 1252–1258, 2003. (Cité en page 45)

[KY08] Shinpei Kato and Nobuyuki Yamasaki. Semi-partitioning technique for multiprocessor real-time scheduling. *migration*, 1 :P2, 2008. (Cité en page 48)

[Lee94] Suk Kyoon Lee. On-line multiprocessor scheduling algorithms for real-time tasks. In *Proceedings of TENCON'94-1994 IEEE Region 10's 9th Annual International Conference on :'Frontiers of Computer Technology'*, pages 607–611. IEEE, 1994. (Cité en page 46)

[Leh90] John P Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *[1990] Proceedings 11th Real-Time Systems Symposium*, pages 201–209. IEEE, 1990. (Cité en page 24), (Cité en page 40)

[LL73a] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1) :46–61, 1973. (Cité en page 21), (Cité en page 22), (Cité en page 23), (Cité en page 55), (Cité en page 104)

[LL73b] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1) :46–61, 1973. (Cité en page 39), (Cité en page 40), (Cité en page 42)

[LM80] Joseph Y-T Leung and ML Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters*, 11(3) :115–118, 1980. (Cité en page 24), (Cité en page 25)

[LM87] Edward A Lee and David G Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9) :1235–1245, 1987. (Cité en page 132)

[LOG+17] Anh-Toan Bui Long, Yassine Ouhammou, Emmanuel Grolleau, Loïc Fejoz, and Laurent Rioux. Bridging the gap between practical cases and temporal performance analysis : a models repository-based approach. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, pages 178–187. ACM, 2017. (Cité en page xii), (Cité en page 101)

[LPY97] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1-2) :134–152, 1997. (Cité en page 82)

[LR06] Didier Lime and Olivier H Roux. Model checking of time petri nets using the state class timed automaton. *Discrete Event Dynamic Systems*, 16(2) :179–205, 2006. (Cité en page 27)

[LRS96] G Laurent, N Rivierre, and M Spuri. Preemptive and nonpreemptive real-time uniprocessor scheduling. Technical report, Tech. Rep, 1996. (Cité en page 49)

[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 21), (Cité en page 41), (Cité en page 74)

[MC97] Aloysius K Mok and Deji Chen. A multiframe model for real-time tasks. *IEEE transactions on Software Engineering*, 23(10) :635–645, 1997. (Cité en page 22)

[McN59] Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1) :1–12, 1959. (Cité en page 47)

[MF76] Philip Merlin and David Farber. Recoverability of communication protocols-implications of a theoretical study. *IEEE transactions on Communications*, 24(9) :1036–1043, 1976. (Cité en page 26)

[mJGJ96] EG Co man Jr, MR Garey, and DS Johnson. Approximation algorithms for bin packing : A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996. (Cité en page 45)

[MJL+06] Jonathan Musset, Étienne Juliot, Stéphane Lacrampe, William Piers, Cédric Brun, Laurent Goubet, Yvan Lussaud, and Freddy Allilaire. Acceleo user guide. *See also http ://acceleo. org/doc/obeo/en/acceleo-2.6-user-guide. pdf*, 2 :157, 2006. (Cité en page 33)

[MMG04] Steven Martin, Pascale Minet, and Laurent George. *Non-premptive Fixed Priority schedulingwith FIFO arbitration : uniprocessor and distributed cases.* PhD thesis, INRIA, 2004. (Cité en page 51)

[MNLM10] Noel Tchidjo Moyo, Eric Nicollet, Frederic Lafaye, and Christophe Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *2010 22nd Euromicro Conference on Real-Time Systems*, pages 271–278. IEEE, 2010. (Cité en page 23)

[Mok83] Aloysius Ka-Lau Mok. *Fundamental design problems of distributed systems for the hard-real-time environment.* PhD thesis, Massachusetts Institute of Technology, 1983. (Cité en page 20), (Cité en page 22), (Cité en page 42)

[MR84] John McDermid and Knut Ripken. *Life cycle support in the ADA environment.* CUP Archive, 1984. (Cité en page 5)

[MTPG11] Chokri Mraidha, Sara Tucci-Piergiovanni, and Sebastien Gerard. Optimum : a marte-based methodology for schedulability analysis at early design stages. *ACM SIGSOFT Software Engineering Notes*, 36(1) :1–8, 2011. (Cité en page 101)

[NAS19] NASA. International space station. `https://www.nasa.gov/mission_pages/station/main/index.html`, 2019. (Cité en page 70)

[Nav96] Nicolas Navet. Le réseau can (controller area network). 1996. (Cité en page 49)

[OMGa] OMG. Modeling and analysis of real-time and embedded systems. `http://www.omg.org/omgmarte/`. [Online ; accessed 20/02/2017]. (Cité en page 34), (Cité en page 105)

[OMGb] OMG. Object constraint language proposed by object menagement group (omg). `http://www.omg.org/spec/OCL/`. [Online ; accessed 20/02/2017]. (Cité en page 33)

[OMGc] OMG. Object management group. `https://www.omg.org/`. (Cité en page 31)

[OMGd] OMG. Query/view/transformation qvt. `https://www.omg.org/spec/QVT/`. (Cité en page 33)

[OMGe] OMG. Uml version 2.5. omg specification, 2015. (Cité en page 31)

[OS95] Yingfeng Oh and Sang H Son. Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Systems*, 9(3) :207–239, 1995. (Cité en page 45)

[Ouh13] Yassine Ouhammou. *Model-based framework for using advanced scheduling theory in real-time systems design*. PhD thesis, 2013. (Cité en page xii), (Cité en page 34), (Cité en page 98), (Cité en page 103), (Cité en page 104), (Cité en page 105)

[PCTM02] John Poole, Dan Chang, Douglas Tolbert, and David Mellor. *Common warehouse metamodel*, volume 20. John Wiley & Sons, 2002. (Cité en page 32)

[PH98] José Carlos Palencia and M González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*, pages 26–37. IEEE, 1998. (Cité en page 22), (Cité en page 52), (Cité en page 55)

[PHAB12] Klaus Pohl, Harald Hönninger, Reinhold Achatz, and Manfred Broy, editors. *Model-Based Engineering of Embedded Systems, The SPES 2020 Methodology*. Springer, 2012. (Cité en page 5)

[PHK+05] Minkyu Park, Sangchul Han, Heeheon Kim, Seongje Cho, and Yookun Cho. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. *IEICE transactions on information and systems*, 88(3) :658–661, 2005. (Cité en page 46)

[PL05] Rodolfo Pellizzoni and Giuseppe Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems*, 30(1-2) :105–128, 2005. (Cité en page 42)

[PRH+16] Baptiste Parquier, Laurent Rioux, Rafik Henia, Romain Soulat, Olivier H Roux, Didier Lime, and Étienne André. Applying parametric model-checking techniques for reusing real-time critical systems. In *International Workshop on Formal Techniques for Safety-Critical Systems*, pages 129–144. Springer, 2016. (Cité en page xii), (Cité en page 82), (Cité en page 83), (Cité en page 84), (Cité en page 85)

[pyt] python. python. `https://www.python.org`. (Cité en page 53)

[Raj12] Ragunathan Rajkumar. *Synchronization in real-time systems : a priority inheritance approach*, volume 151. Springer Science & Business Media, 2012. (Cité en page 90)

[RCR01] Pascal Richard, Francis Cottet, and Michaël Richard. On-line scheduling of real-time distributed computers with complex communication constraints. In *Proceedings Seventh IEEE International Conference on Engineering of Complex Computer Systems*, pages 26–34. IEEE, 2001. (Cité en page 62)

[RGRR12] Ahmed Rahni, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Feasibility analysis of real-time transactions. *Real-Time Systems*, 48(3) :320–358, 2012. (Cité en page 22), (Cité en page 55)

[RRGC02] M Richard, P Richard, E Grolleau, and F Cottet. Contraintes de précédences et ordonnancement mono-processeur. *Real-time and embedded systems (RTS'02)*, 2002. (Cité en page 61)

[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 103)

[SB02] Anand Srinivasan and Sanjoy Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information processing letters*, 84(2) :93–98, 2002. (Cité en page 45)

[Sch06] Douglas C. Schmidt. Guest editor's introduction : Model-driven engineering. *IEEE Computer*, 39(2) :25–31, 2006. (Cité en page 29)

[SEGY11a] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The digraph real-time task model. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 71–80. IEEE, 2011. (Cité en page 23)

[SEGY11b] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. On the tractability of digraph-based task models. In *2011 23rd Euromicro Conference on Real-Time Systems*, pages 162–171. IEEE, 2011. (Cité en page 23)

[SH98] Mikael Sjodin and Hans Hansson. Improved response-time analysis calculations. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*, pages 399–408. IEEE, 1998. (Cité en page 40)

[Sha04] Paul Shaw. A constraint for bin packing. In *International conference on principles and practice of constraint programming*, pages 648–662. Springer, 2004. (Cité en page 45)

[SL16] Youcheng Sun and Giuseppe Lipari. A pre-order relation for exact schedulability test of sporadic tasks on multiprocessor global fixed-priority scheduling. *Real-Time Systems*, 52(3) :323–355, 2016. (Cité en page 90)

[SLNM04] Frank Singhoff, Jérôme Legrand, Laurent Nana, and Lionel Marcé. Cheddar : a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM, 2004. (Cité en page 52)

[SRL90] Lui Sha, Ragunathan Rajkumar, and John P Lehoczky. Priority inheritance protocols : An approach to real-time synchronization. *IEEE Transactions on computers*, 39(9) :1175–1185, 1990. (Cité en page 43), (Cité en page 44), (Cité en page 90)

[Sta88] John A. Stankovic. Misconceptions about real-time computing : A serious problem for next-generation systems. *Computer*, 21(10) :10–19, 1988. (Cité en page 15)

[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 xi), (Cité en page 23), (Cité en page 103)

[TC94] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2-3) :117–134, 1994. (Cité en page 49)

[Tin94] Ken Tindell. *Adding time-offsets to schedulability analysis*. Citeseer, 1994. (Cité en page 49), (Cité en page 52)

[VK00] Steve Vestal and Jonathan Krueger. Technical and historical overview of metah. *Honeywell Technology Center*, page 42, 2000. (Cité en page 33)

[Wik] Wikipedia. Mathml. `https://en.wikipedia.org/wiki/MathML`. [Online ; accessed 15/05/2018]. (Cité en page 122)

[WO01] Jos Warmer and Klasse Objecten. The future of uml. *OMG Information Day, Amsterdam*, 2001. (Cité en page xi), (Cité en page 31)

[YR98] Tomohiro Yoneda and Hikaru Ryuba. Ctl model checking of time petri nets using geometric regions. *IEICE Transactions on Information and Systems*, 81(3) :297–306, 1998. (Cité en page 26)

Résumé

Les systèmes embarqués temps réel sont de plus en plus omniprésents dans la vie quotidienne. Le cycle de développement des systèmes embarqués temps réel critiques peut prendre des mois voire des années. Par conséquent, la modélisation de ces systèmes doit être analysée à un stade précoce du cycle de développement afin de vérifier si toutes les exigences sont satisfaites, y compris les exigences relatives à la performance temporelle (par exemple, les latences, les délais de bout en bout, etc.). Cette thèse, qui a été financée dans le cadre d'un projet FUI, propose trois contributions majeures. La première contribution porte sur le système de tâches monoprocesseur avec des relations de communication multi-périodique déterministe. Un pattern basé sur les relations de précédence avec sémaphore (SPC) a été étendu afin de supporter les cycles dans le cas d'ordonnancement à priorités dynamiques. Un graphe de dépliage a été également proposé afin de présenter la cyclicité des systèmes à base de SPC et garantir le non- blocage. Le pattern SPC étendu ainsi que le test d'ordonnançabilité correspondant ont été implémentés pour le langage standard AADL. La deuxième contribution de cette thèse consiste en la proposition d'un calcul exact de temps de réponse de bout en bout, en utilisant le formalisme de réseau de Petri temporel, pour les systèmes multiprocesseurs identiques. Il prend en compte la dépendance entre les tâches : la précédence et l'exclusion mutuelle sans protocole de gestion. La troisième contribution porte sur la capitalisation des efforts du processus d'analyse. En effet, de nombreux tests d'analyse ont été proposés, notamment par des chercheurs académiques, basés sur la théorie d'ordonnancement et dédiés aux différentes architectures logicielles et matérielles. Cependant, l'une des principales difficultés rencontrées par les concepteurs est de choisir le test d'analyse le plus approprié permettant de valider et/ou de dimensionner correctement leurs conceptions. Cette difficulté se concrétise, dans le milieu industriel, par le peu de tests d'analyse utilisés malgré la multitude de tests proposés. Cette thèse vise donc à faciliter le processus d'analyse, réduire le fossé sémantique entre le modèle métier et les entrées des tests d'analyse ainsi qu'accélérer le transfert technologique et l'adoption des tests académiques. La thèse propose un référentiel d'analyse jouant le rôle d'un dictionnaire de tests, leurs contextes pour une utilisation correcte, les outils les implémentant, ainsi qu'un mécanisme pour le choix des tests selon le modèle métier d'entrée.

Mots-clés : AADL (informatique) ; Analyse temporelle ; Ingénierie dirigée par les modèles ; Ordonnancement (informatique) ; Système, Analyse de ; Système, Conception de ; Systèmes embarqués (informatique) ; Temps réel (informatique)

Abstract

Real-time embedded systems are increasingly omnipresent in everyday life. The development cycle of critical systems can take months or even years. Therefore, modeling of these systems should be analyzed at an early stage in the development cycle to verify that all requirements are met, including temporal requirements (e.g., latencies, end-to-end delays). This thesis, which was funded as part of FUI project, offers three major contributions. The first contribution relates to mono-processor task system with deterministic multi-periodic communication relationships. A pattern based on Semaphore Precedence Constraint (SPC) has been extended to support cycles in the case of dynamic priority scheduling. An unfolding graph has also been proposed in order to present the cyclicity of SPC-based systems and guarantee deadlock free. The extended SPC pattern and the corresponding scheduling analysis tests have been implemented for the standard AADL language. The second contribution of this thesis consists in proposing an exact calculation of end-to-end response time using the time Petri net formalism for identical multiprocessor systems. It takes into account the dependence between the tasks : precedence and mutual exclusion. The third contribution concerns the capitalization of the efforts of the analysis process. Indeed, many analytical tests have been proposed, notably by academic researchers, based on scheduling theory and dedicated to the different software and hardware architectures. However, one of the main difficulties encountered by designers is to choose the most appropriate analysis test to validate and/or correctly dimension their designs. In industrial environment, there are few analytical tests used despite the multitude of the tests offered. This thesis therefore aims to facilitate the analysis process, reduce the semantic gap between the business model and the entries in the analysis tests as well as accelerate technology transfer and the adoption of academic tests. The thesis proposes an analysis repository playing the role of a dictionary of tests, their contexts for correct use, the tools implementing them, as well as a mechanism for choosing tests according to the input business model.

Keywords : Architecture Analysis ans Design Language ; Time-domain analysis ; Computer scheduling ; System analysis ; System design ; Embedded computer systems ; Real-time data processing.

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