Architecture logicielle et méthodologie de conception embarquée sous contraintes temps réel pour la radio logicielle

Noël Tchidjo Moyo

N° d'ordre 4314 ANNEE 2011

UNIVERSITÉ DE RENNES 1

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de
DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Electronique
Ecole doctorale Matisse

présentée par

Noël Bertrand TCHIDJO MOYO

préparée à l'unité de recherche : SUPELEC/IETR UMR 6164
Nom développé de l'unité : Institut d'Electronique et de Télécommunication de Rennes
Composante universitaire : S.P.M

Architecture logicielle et méthodologie de conception embarquée sous contraintes temps réel pour la radio logicielle

Thèse soutenue à Rennes le 20 avril 2011
devant le jury composé de :

Maryline CHETTO
Professeur, IUT, IRCCYN, Nantes / *rapporteur*

Joël CHAMPEAU
Maître de conférence, ENSTA, Bretagne / *rapporteur*

Isabelle PUAUT
Professeur, Université de Rennes 1, IRISA / *examinateur*

Christophe MOY
Professeur, SUPELEC / *directeur de thèse*

Frédéric LAFAYE
Responsable industriel, THALES / *examinateur*

Jean-Philippe DELAHAYE
Ingénieur de recherche, DGA / *examinateur*

Remerciements

Cette thèse est le fruit de 3 années de travail effectuées au sein du service SPM (Signal Processing Multimedia) de THALES Communications (Colombes) et du Laboratoire SCEE/IETR de SUPELEC (Campus de Rennes). Je remercie beaucoup Monsieur Gilles BOURDE chef du service SPM et Monsieur Jacques PALICOT responsable du laboratoire SCEE pour m'avoir accueilli pour cette thèse et pour leurs conseils tout au long de mes travaux.

Je tiens à remercier particulièrement mon directeur de thèse Christophe MOY et mes encadrants industriels Frédéric LAFAYE et Eric NICOLLET pour leur disponibilité, leur encadrement et leurs conseils qui ont été précieux dans les résultats obtenus pendant ce travail.

J'adresse un grand remerciement aux rapporteurs : Maryline CHETTO, Joël CHAMPEAU ; et aux membres du jury : Isabelle PUAUT, Christophe MOY, Frédéric LAFAYE, Jean-Philippe DELAHAYE.

Je remercie tous ceux avec qui j'ai eu des discussions scientifiques très fructueuses pendant ma thèse, en particulier Marie-Anne LEFEBVRE, Nabil SADOU, Vincent SEIGNOLE, Frédéric BRUEL, Pierre-André LAURENT.

Je tiens à exprimer ma reconnaissance et ma profonde amitié à Frédéric LAFAYE, pour son accompagnement, sa formation, sa gentillesse, bref je lui dois beaucoup.

Mes remerciements vont aussi à l'ensemble de mes collègues chez THALES Communications pour les échanges à la pause café, à la cantine, et pour les bons moments passés ensemble au challenge et à la section Basket.

Enfin je remercie ma famille, ma maman chérie, mes proches et mes amis pour leur présence, leur soutien et leurs encouragements.

Une pensée pour terminer ces remerciements à mon père parti très tôt, je sais que tu aurais été très fier de ton fils.

Résumé

Cette étude répond au problème d'ordonnancement temps réel de composants logiciels s'exécutant sur un processeur de traitement du signal dans un contexte de radio logicielle. Elle vise ainsi à compléter l'offre en termes d'outillage de conception radio logicielle. Dans la pratique actuelle, l'ordonnancement temps réel des applications de traitement du signal flexibles s'exécutant sur un processeur donné, est effectué de manière manuelle, en utilisant des méthodes empiriques, et en prenant des marges non négligeables. Etant donnée l'augmentation pressentie du nombre de composants logiciels de la couche physique s'exécutant simultanément sur un même processeur dans les futures radios logicielles, ces méthodes seront sujettes à erreur, feront perdre beaucoup de temps et ne trouveront pas nécessairement de solutions d'ordonnancement valides même lorsqu'il en existera une.

Pour cela, cette thèse définit un nouveau modèle de tâche représentant plus précisément le comportement des tâches dans certains contextes de radio logicielle : le modèle GMF (Generalized Multi-Frame) non cyclique. Pour ce modèle, nous présentons une formulation du calcul du temps de réponse des tâches, ainsi qu'un nouveau test de faisabilité suffisant pour des tâches s'exécutant sur un processeur avec la politique d'ordonnancement « Earliest Deadline First » (EDF). Nous fournissons aussi pour ce modèle de tâche un algorithme efficace, permettant la détermination exacte de la faisabilité temps réel.

Nous présentons dans cette thèse un nouveau flot de conception IDM (Ingénierie Dirigée par les Modèles), permettant de spécifier les paramètres rendant possibles une analyse d'ordonnançabilité temps réel des composants logiciels s'exécutant sur un processeur dans une radio logicielle. Cette thèse propose des méthodes pour calculer les contraintes temporelles dans une radio logicielle. Elle présente les éléments du standard MARTE à utiliser pour renseigner les contraintes dans le modèle ainsi que les règles de transformations de modèles qui permettent d'obtenir un modèle exploitable par un outil d'analyse d'ordonnançabilité temps réel.

Cette thèse présente une approche, implantée sous forme d'un outil de simulation, effectuant l'analyse d'ordonnancement temps réel des tâches de traitement du signal flexibles s'exécutant sur un processeur suivant une politique d'ordonnancement hybride. Cet outil est intégré au flot IDM proposé.

Abstract

This study addresses the problem of real-time scheduling of software components executing in a digital signal processor in a software radio context. It aims at providing new tooling for software radio design. Real-time scheduling analysis of flexible signal processing applications executing in a processor is currently done manually, using ad hoc methods, and taking significant margins. Given the foreseen increase of software components of the physical layer executing simultaneously on a processor in future software radios, these methods for scheduling analysis will be error-prone, time consuming and will often fail to find a feasible schedule even when one exists.

For that purpose, this thesis defines a new task model which represents more precisely the behaviour of the tasks in certain software radio context: the non-cylic GMF (Generalized Multi-Frame) model. For this model, we present a formula to compute response time of tasks, as well as a new sufficient feasibility test for tasks executing in a processor according to the "Earliest Deadline First" scheduling policy. We also provide for this task model an efficient algorithm, for exact feasibility determination.
We present in this thesis a new MDE (Model Driven Engineering) design methodology, to specify the parameters which make possible a real-time scheduling analysis of software components executing in a processor. This thesis proposes methods to compute real-time constraints in a software radio. It presents the elements of the MARTE standard to be used, to note the constraints in the model as well as model transformation rules to obtain a suitable model for real-time scheduling analysis.
This thesis presents an approach, implemented as a simulation tool, to realize real-time scheduling analysis of tasks implementing flexible signal processing algorithms in a processor and scheduled according to a hybrid scheduling policy. This tool is integrated into the proposed MDE design methodology.

Table des matières

Table des figures

Liste des tableaux

Introduction

Télécommunications d'hier à aujourd'hui

Il est communément reconnu que l'homme existe socialement à travers la communication, c'est pourquoi il a toujours eu besoin de communiquer avec ses congénères. L'élément essentiel sur lequel s'est d'abord appuyé l'homme pour échanger est la voix et les gestes, mais pour se comprendre il faut utiliser le même langage, utiliser le même protocole.

Au cours des âges, les moyens de communications des hommes ont évolué en même temps que l'évolution de leurs mœurs et de leurs besoins. Communiquant d'abord au moyen de la voix, la portée de celle-ci a contraint les hommes à réfléchir à d'autres moyens permettant de communiquer en s'affranchissant de la distance. Les alternatives sont basées sur la mise en forme de l'information suivant un protocole et sa transmission via un support (un outil autre que la parole). Ce fut le cas des peuples d'Afrique qui dialoguaient avec des tam-tams, des indiens d'Amérique par le biais des nuages de fumées ou des sémaphores des frères Chappe à la fin du 18ème siècle en France. Ces moyens ne permettant pas d'avoir de vraies discussions, la découverte de l'électricité a permis de mettre en place d'autres protocoles et d'expérimenter des moyens de communications sur de plus longues distances. Ainsi la naissance du télégraphe en 1838 par Wheatstone associé au morse (1844) de Samuel Morse ont permis d'envoyer un message sur une distance de 60km via le réseau télégraphique. Ensuite la compréhension de l'onde radio a permis d'envoyer des signaux sans support câblé. Par la suite, se sont succédés l'apparition du téléphone (1876), du télégraphe sans fil (1899), de la radiophonie (1900), ainsi que les protocoles utilisés par ces outils. Ces outils reposent sur des composants et des systèmes électroniques, qui eux aussi ont connu au fil du temps une croissance rapide de leurs capacités. La radio n'a cessé de se développer au 20ème siècle, pour passer d'applications militaires à une multitude d'applications civiles dans lesquelles baigne notre vie quotidienne (GPS, TV, Radio, commandes d'ouvertures à distance, réseaux sans fil, etc.). Aujourd'hui, différents standards de communications radio sont disponibles pour le grand public : GSM, GPRS, EDGE, UMTS et bientôt LTE.

Face à cette multitude de standards radios, la mobilité accrue des personnes, le besoin croissant d'appareils ayant la capacité de s'adapter automatiquement à l'environnement dans lequel ils se trouvent, il est nécessaire de proposer des équipements de transmission radio flexibles et adaptatifs. Une solution sur laquelle les chercheurs travaillent depuis une vingtaine d'années est la **radio logicielle.**

La Radio logicielle

Le terme « Software Radio » (Radio logicielle) a été inventé en 1991 par le professeur Joseph Mitola, qui a publié un premier article sur le sujet en 1992 [90]. Bien que le concept ait été proposé en 1991, la radio logicielle a ses origines dans les années 70 lorsque l'armée américaine s'est intéressée à l'interopérabilité entre divers systèmes

fonctionnant dans des bandes de communications différentes (HF, VHF, UHF…), avec des chaînes de modulations différentes (AM et FM) et des formats de données différents. Une radio logicielle est une radio dont les fonctions sont majoritairement implantées sous forme logicielle. Etant donné les contraintes technologiques actuelles, il y a toujours du matériel dédiés aux fréquences radio, mais l'idée est d'avoir, tant que c'est faisable, les traitements effectués sous forme logicielle le plus proche possible de l'antenne. Les « entorses » au concept de la radio logicielle (idéale, toute en logiciel) ont fait apparaître un autre terme, la radio logicielle restreinte ou « SDR » (Software Defined Radio), mais les deux appellations sont souvent considérées équivalentes. La figure 1 présente un schéma simplifié d'une chaîne radio.

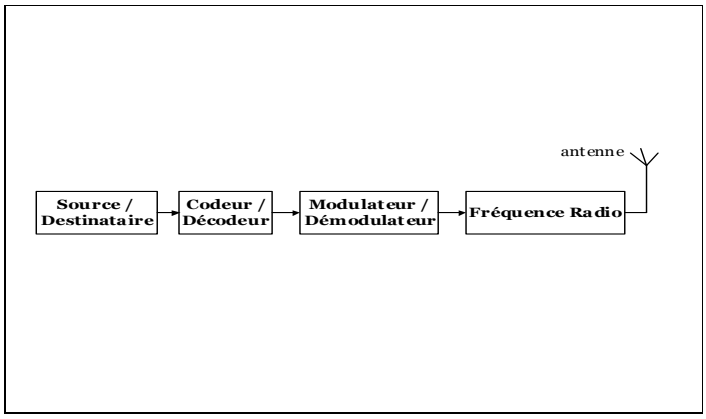

Fig. 1– Schéma simplifié d'une chaîne radio

Une radio logicielle basique pourrait consister en un processeur de traitement du signal numérique, un convertisseur numérique/analogique, une carte RF/IF (conversion des fréquences intermédiaires en fréquence radio), et d'une antenne. Ainsi, une quantité significative du traitement du signal est faite de manière logicielle sur le processeur au lieu d'être implanté sur du matériel spécifique. Une telle conception permet à la radio de pouvoir supporter une large variété de protocoles radios en se basant uniquement sur des mises à jour logicielles [90]. Une radio logicielle est donc flexible.

La formalisation de la radio logicielle s'effectue au sein d'un Forum qui rassemble les acteurs majeurs du domaine. Il s'agit du SDR Forum (aujourd'hui Wireless Innovation Forum), où les différents acteurs présentent les avancées dans le domaine. D'ailleurs, les travaux de cette thèse y ont été présentés. Pour faciliter le développement de la radio logicielle les programmes américains de défense : JTRS (Joint Tactical Radio Systems) et le JPEO (Joint Program Executive Office) ont standardisé une architecture pour les équipements radio. C'est le SCA (Software Communications Architecture) [91]. Le SCA spécifie les moyens de déploiement, de gestion, d'interconnexion et d'intercommunication des composants logiciels d'une radio. Le SCA est une structure architecturale qui a été créée pour maximiser la portabilité, l'interopérabilité, et la reconfigurabilité du logiciel tout en gardant la flexibilité pour s'adapter aux exigences et restrictions spécifiques à un domaine. Les contraintes sur le développement du logiciel imposé par la structure concernent les interfaces et la structure du logiciel mais pas la mise en œuvre des fonctionnalités qui sont exécutées.

Cependant, les applications de traitement du signal ont une notion duale d'exactitude : exactitude logique et exactitude temporelle. Il n'est pas suffisant de juste produire les données correctes (par exemple les données décodées sans erreur de transmission). Les données correctes doivent être produites dans un intervalle de temps borné, connu à l'avance (par exemple les données décodées sans erreur de transmission au bout de 2ms). Ces contraintes temporelles sont généralement imposées par le standard considéré. Ainsi, les évolutions de la conception radio entraînent des modifications dans le processus de conception, de vérification et de validation de l'ordonnançabilité temps réel d'une radio logicielle. La vérification de l'ordonnancement temps réel n'est pas adressée dans le SCA.

Contexte des travaux de thèse

Dans la pratique actuelle, l'ordonnancement temps réel d'une radio logicielle est fait de manière manuelle, utilisant des méthodes empiriques, basées sur l'expérience des ingénieurs. Cependant, considérons une radio logicielle composée d'une part au niveau des applications, de 2 schémas de modulation OFDM (Orthogonal Frequency Division Multiplexing), d'un schéma de modulation CPM (Continuous Phase Modulation), et d'autre part au niveau de sa plate-forme matérielle d'un processeur générique, d'un processeur de traitement du signal numérique, de trois composants FPGA, de convertisseurs, d'amplificateurs et de trois antennes. Evidemment si nous avions une approche matérielle, les algorithmes de traitement du signal émanant de ces 3 schémas de modulation s'exécuteraient en parallèle sur des circuits intégrés séparés et dédiés. Dans cette approche, il n'y aurait pas de problème d'ordonnancement temps réel car il suffirait de trouver un ordre statique hors ligne entre les composants. Dans cette étude nous choisissons une approche logicielle, compte tenu de tous les avantages qu'elle apporte. Par conséquent si nous ramenons le problème de conception au cas du processeur de traitement du signal numérique, étant donné la grande combinatoire de fonctions de traitement du signal pouvant s'exécuter simultanément sur ce processeur, les méthodes manuelles et empiriques pour l'ordonnancement temps réel prendront beaucoup de temps, seront sujettes à erreur, et ne réussiront pas à déterminer une solution faisable même lorsqu'il en existera une. En effet contrairement à une approche matérielle, les fonctions ne s'exécutent pas en parallèle sur le processeur, mais séquentiellement. Il faut par conséquent répartir l'exécution des tâches au cours du temps sur le processeur, ce qui crée des problèmes d'ordonnancement temps réel. C'est pourquoi THALES Communications S.A (Colombes) et SUPELEC (Campus de Rennes) ont proposé un sujet de recherche dans ce domaine.

L'objectif est de définir et mettre en place une méthodologie de conception orientée MDA pour la radio logicielle et intégrant des moyens de tests d'ordonnançabilité temps réel.

Ces travaux s'appuient sur un projet d'étude amont financé par THALES Communications. Le projet consiste en une implantation de la version multi-utilisateur du schéma de modulation OFDM avec évasion de fréquence (saut de fréquence) sur une carte électronique composé d'un PowerQuicc, d'un DSP et d'un FPGA. Ainsi plusieurs utilisateurs partagent le même équipement et l'accès multiple est atteint en assignant un sous-ensemble de sous-porteuses à chaque utilisateur. Cet équipement transmet des

signaux radio en sautant rapidement parmi plusieurs fréquences porteuses et en utilisant une séquence pseudo aléatoire connue par l'émetteur et le récepteur. De plus étant donné, l'émission et la réception en continu sur l'antenne, certains traitements doivent être anticipés et exécutés simultanément par rapport à d'autres. Par exemple la fonction qui programme les fréquences porteuses saute en fréquence pour la réception en cours d'une trame, pendant que l'on code la trame à émettre à la suite de cette réception. Dans cette radio logicielle, plusieurs fonctions devant être exécutées simultanément vont être implantées sous forme logicielle. Cet équipement radio est programmable car toute la partie adressant le contrôle de la radio (.i.e. le séquencement et la programmation des fréquences porteuses et de l'amplificateur de puissance) est effectuée sous forme logicielle. Ainsi, on pourra changer de bande de fréquence ainsi que la liste des fréquences sur lesquelles on saute juste en faisant un changement des paramètres du logiciel. Nous allons montrer que cette radio logicielle pose donc des nouveaux problèmes d'ordonnancement temps réel dans le cas notamment d'une implantation mono-processeur. Les travaux de cette thèse s'intéressent à l'ordonnancement temps réel des fonctions de traitement du signal qui ont été déployées sur le DSP. Dans ce document, lorsque nous parlons de radio logicielle cela intègre aussi les radios logicielles compatibles SCA. Le SCA propose une structure permettant le déploiement de composants logiciels correspondants à un standard radio.

Problématiques abordées

Le cadre de ce projet repose sur un cas d'utilisation réel auquel doit répondre un équipement, à savoir que l'équipement radio supporte des trames de tailles différentes, car elles correspondent à des services différents (par exemple : trame audio – petite taille, trame vidéo – grande taille). Ainsi, les tâches logicielles implantant les algorithmes de traitement du signal de l'équipement radio ont des contraintes temporelles qui varient en fonction du type de trame en entrée. En effet les trames audio ont une échéance temporelle plus courte que celle des trames vidéo. De plus les trames vidéo nécessitent plus de temps de traitement que les trames audio. Un autre point important est la modélisation de la tâche logicielle qui programme les fréquences porteuses, car en fonction de la taille de la trame en entrée, le nombre d'activations de la tâche varie aussi. En outre, le concepteur doit pouvoir disposer de méthodes lui permettant de calculer les contraintes temporelles, d'outillage, et d'une méthodologie de conception afin de déterminer la faisabilité temps réel des composants logiciels d'une radio. En résumé, le concepteur de radio logicielle se pose la question suivante :

Etant donné la spécification d'une radio logicielle, composée de tâches implantant des algorithmes de traitement du signal flexibles, i.e. avec des contraintes temporelles variables, comment peut-on déterminer en phase de conception, si l'ensemble des tâches s'exécutant simultanément sur un processeur vont être ordonnancées de manière à ce que chaque tâche respecte ses échéances temporelles ?

Contributions

Ce document fournit les contributions suivantes :

(1) Un nouveau modèle de tâche (le modèle de tâche multi-trames généralisées non cycliques) et une nouvelle formule pour le calcul du temps de réponse des tâches s'exécutant sur un processeur en fonction de la politique d'ordonnancement EDF (Earliest Deadline First). Ce modèle caractérise mieux le comportement des tâches dans une radio logicielle. A partir de cette formule, nous présentons un nouveau test de faisabilité suffisant.

(2) Un algorithme efficace, pour la détermination exacte de la faisabilité temps réel d'un ensemble de tâches multi-trames généralisés non cycliques ordonnancées avec EDF.

(3) Une méthodologie de conception permettant l'analyse d'ordonnancement temps réel dans une radio logicielle. Nous présentons un ensemble de règles à suivre pendant les phases de modélisation PIM (Platform Independent Model), PDM (Platform Description Model) et PSM (Platform Specific Model), en considérant en particulier le profil MARTE [7] (une spécialisation d'UML pour les systèmes temps réel embarqués). Une expérimentation est aussi présentée afin de démontrer les avantages de nos méthodes.

(4) Une approche pour la vérification de l'ordonnancement temps réel des tâches implantant des algorithmes de traitement du signal flexibles et s'exécutant sur un processeur en fonction d'une politique d'ordonnancement hybride. Cette approche est implantée sous forme d'un outil de simulation et nous démontrons que nous obtenons une bonne précision en comparant les résultats prédits par l'outil avec ceux obtenus dans un cas réel d'implantation sur cible matérielle.

Plan du mémoire

Ce document s'organise de la manière suivante.
Le premier chapitre présente la structure d'une radio logicielle ainsi que les concepts de base en méthodologie de conception et en ordonnancement temps réel monoprocesseur.

Le deuxième chapitre présente notre nouveau modèle de tâche, une formule pour le calcul du temps de réponse, un test de faisabilité suffisant, puis un algorithme efficace pour l'analyse exacte de l'ordonnancement temps réel de notre nouveau modèle de tâche en considérant un ordonnanceur EDF.

Le troisième chapitre montre la méthodologie de conception, et l'outillage que nous avons mis en place pour déterminer la faisabilité temps réel des composants logiciels s'exécutant simultanément sur un processeur dans un équipement radio.

Le quatrième chapitre présente une expérimentation de nos méthodes (s'appuyant sur un projet en cours de réalisation) qui démontre les avantages de notre méthodologie. Nous montrons aussi dans ce chapitre que notre outil de simulation a une bonne précision en comparant les résultats prédits par l'outil à ceux obtenus sur cible.
Enfin la conclusion retrace nos travaux effectués, présente quelques améliorations qui pourraient être apportées et ouvre la voie sur une poursuite des travaux.

Chapitre 1

Etat de l'art

1.1 Introduction

Les successions d'innovations rapides de ces 25 dernières années dans le domaine de l'électronique embarquée, sont notamment le résultat de l'amélioration permanente de méthodologies de conception en adéquation avec les technologies des composants physiques qui supportent ces applications. Une méthodologie de conception, dans le domaine de l'électronique embarquée, est l'ensemble des méthodes et outils, utilisés pour mener à bien la conception de systèmes électroniques. Ces systèmes électroniques ont pour la plupart la propriété d'être « temps réel » car ils doivent répondre en des intervalles de temps bien précis, connus à l'avance. Ce premier chapitre décrit d'abord l'architecture d'une radio logicielle telle qu'elle est envisagée dans cette étude, ensuite il présente les différentes méthodes et langages dont le concepteur dispose pour décrire et concevoir un système. Il s'attarde notamment sur le langage UML et sur la prise en compte de l'aspect temps réel en phase de modélisation. Ce chapitre propose aussi un état de l'art de l'ordonnancement temps réel sur une architecture monoprocesseur. A la fin de ce chapitre nous présentons les problématiques d'ordonnancement temps réel rencontrées durant la conception d'une radio logicielle. Il s'agit d'une part des problématiques liées à la prise en compte des contraintes temporelles durant la phase de modélisation et d'autre part de problématiques liées à la validation de l'ordonnancement temps réel à partir de ces contraintes.

1.2 La radio logicielle

Les activités humaines utilisent de nombreux signaux radio émanant des téléphones mobiles, de périphériques de communications « bluetooth », de liaisons WIFI pour l'accès à internet, de signaux destinés à des récepteurs AM/FM, des téléviseurs HD, des récepteurs GPS, de portes automatiques de garage (cette liste est loin d'être exhaustive). Par analogie avec le concept de l'ordinateur pouvant supporter différents types d'applications, est apparu le concept de radio logicielle (software radio) pour exécuter plusieurs protocoles de communications sur un même équipement radio. Le terme « Software Radio », dont l'objectif premier est d'implanter une majorité des fonctions d'un équipement radio de manière logicielle, a été proposé par Joseph Mitola [90]. Ainsi

l'équipement radio est plus flexible par rapport à une implantation où une majorité des composants sont réalisés sous forme matérielle.
Une radio logicielle est donc une radio flexible où les fonctionnalités sont contrôlées avec le logiciel qui est capable d'exécuter plusieurs formes d'ondes sur le même équipement radio en fonction des besoins opérationnels. Une forme d'onde « waveform - [91]» correspond à la transformation entre une entrée utilisateur et une sortie en fréquence radio et vice versa. Une forme d'onde est définie grâce aux protocoles radio, qui sont conformes aux standards tel que l'UMTS, le GSM etc. ou qui sont propriétaires et définis par les fabricants d'équipements radio. Par principe (approche logicielle de type PC), une radio logicielle favorise la réutilisation, et la flexibilité car les évolutions de l'équipement peuvent être facilement effectuées avec des mises à jour logicielles au lieu de remplacer le matériel.
Les trois principaux objectifs de la radio logicielle sont donc, **la portabilité** [93] des formes d'ondes à travers les différentes plateformes radio, **l'interopérabilité** entre les composants des formes d'ondes, et **la reconfigurabilité** [94] de l'équipement afin de répondre à plusieurs besoins opérationnels.
L'ajout dans une radio logicielle d'une certaine intelligence, produit une radio cognitive ou radio intelligente. Cette notion a été proposée par Joseph Mitola lors d'un séminaire en 1998 puis publié en 1999 dans [95]. Une radio intelligente est une radio qui s'adapte et optimise ses paramètres radio en fonction de l'environnement pour, par exemple, utiliser efficacement le spectre de fréquence et les ressources disponibles. Au-delà, tout ce qui permet une meilleure gestion des besoins des utilisateurs, par une adaptation en temps-réel des traitements relatifs au lien radio, est concerné par la radio intelligente. Quelques méthodologies de conception ont été proposées dans le domaine de la radio intelligente [96] [19] [98].

1.2.1 L'architecture logicielle SCA

L'architecture logicielle SCA a pour but de faciliter le développement de radio logicielle en maximisant **la portabilité, la réutilisation,** et **l'interopérabilité** du logiciel grâce à l'utilisation des produits et protocoles commerciaux. L'architecture a été développée en utilisant une approche orientée objet avec les pratiques courantes venant de l'approche par composants et les « design patterns ». Le SCA représente donc une radio logicielle sous la forme d'un ensemble de composants logiciels qui ont besoin de former un chemin de communication et de s'exécuter sur un ensemble d'éléments matériels. Il propose une structure commune afin de gérer les composants logiciels et matériels, ainsi qu'un jeu d'interfaces pour isoler l'application logicielle du matériel. Il s'agit du « core framework ».
Comme décrit dans la Figure 1.1, le « core framework » fait partie d'un environnement d'exécution (OE : Operating Environment). Grâce aux capacités de déploiement dynamique fournies par le « core framework », l'OE peut reconfigurer un équipement radio et permettre l'exécution de plusieurs formes d'ondes en parallèle. En plus des services réalisés par le « core framework », l'OE fournit le mécanisme de connectivité entre les composants (CORBA – Common Object Request Broker Architecture), le service

de gestion du temps, un système d'exploitation temps réel et une interface compatible POSIX pour l'utilisation de ces services. La plateforme de la Figure 1.1 est composée de plusieurs unités de calculs. En effet bien que la radio logicielle idéale préconise que toutes les fonctions de traitement du signal soient réalisées de manière entièrement logicielle, certains traitements demandent une très grande puissance de calcul et ont besoin d'être mis en œuvre sur des unités de calculs comme des FPGA.

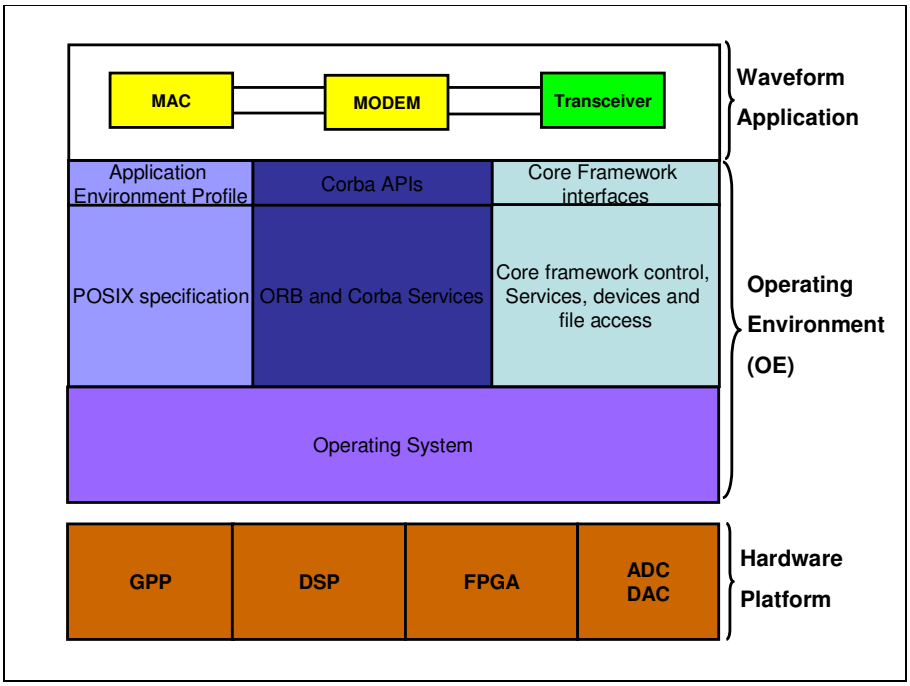

Figure 1.1 – Architecture logicielle d'un équipement radio d'après le SCA

Le SCA souffre du fait que l'architecture qu'il propose, engendre une certaine latence dans le système et n'est pas adapté pour certains traitements fortement contraints en temps (notamment les traitements de la partie modem de l'équipement radio [102]). De plus cette architecture occupe une empreinte mémoire non négligeable. Une nouvelle version du SCA (SCA NEXT) en cours de réalisation a pour objectif de combler ces lacunes [103].

1.2.2 Cas d'étude de la thèse

Le cas d'étude auquel se rapporte cette thèse consiste à l'implantation sur un équipement radio de la version multi-utilisateur du schéma de modulation OFDM. Contrairement à la technologie actuelle existante, la portée et la puissance de l'équipement radio considéré sont plus importantes. En effet l'équipement radio a une puissance de 10 Wt pour offrir un débit de 2 Mégabits/s sur une distance de 5 km, alors la technologie WIFI 802.11b (utilisant elle aussi une modulation OFDM [100]) que nous utilisons quotidiennement offre un débit de 2 Mégabits/s sur 400 m avec une puissance de 10 mW. Cette portée plus importante est nécessaire dans un contexte militaire car les modems sont embarqués dans des véhicules avec lesquels les militaires se déplacent. Ce qui n'est pas le cas du

WIFI où les bornes sont fixes. Pour atteindre ce débit à cette distance il faut implanter des algorithmes de traitement du signal plus robustes et par conséquent plus exigeants en termes de calcul. On va par exemple utiliser un turbo code pour le codage et le décodage canal. Une implantation logicielle de cet algorithme (par rapport à son équivalent matériel) aura un impact fort sur les contraintes temps réel. De plus l'équipement radio gère des trames de tailles différentes car elles correspondent à des services différents (vidéo – grande trame, audio – petite taille). Ceci implique des contraintes temporelles variables. Ce n'est pas le cas dans le WIFI où la pile IP regroupe généralement les données en paquets de même taille avant de les envoyer.

Les trames de longueurs différentes sont aussi rencontrées dans d'autres systèmes à base d'OFDM, comme le standard DVB-T pour la diffusion de la TNT. En effet ce standard utilise le dispositif de compression vidéo MPEG [61] qui manipule trois types de trames connus sous le nom de trame I, trame P, trame B. Cependant ces trames de tailles différentes sont regroupées en trame d'une longueur de 288 octets en passant par une couche transport MPEG2-TS [101]. Ainsi au niveau du modulateur OFDM, les trames ont la même taille. Par conséquent contrairement à l'équipement radio considéré, le modulateur OFDM du DVB-T traite donc des trames de même taille. En outre dans l'équipement radio considéré ici on saute en fréquence à l'intérieur de la trame car la trame est découpée en paliers émis et reçus à des fréquences porteuses spécifiques. Dans la technologie WIFI, il y'a également un saut de fréquence, cependant la longueur du saut est de 400 ms et les fréquences de saut sont fixes alors que dans notre équipement la longueur du saut est inférieure ou égale à 1 ms et les fréquences de saut peuvent changer pendant la communication. Ainsi pendant 1 seconde d'émission de données sur le WIFI on sautera sur 2 fréquences fixes connues par l'émetteur et le récepteur tandis que dans notre équipement on sautera sur au moins 10 fréquences qui peuvent changer pendant la communication. Notre équipement radio doit donc être plus réactif. La fonction de contrôle qui programme les fréquences porteuses et l'amplificateur de puissance va également être implantée sous forme logicielle et s'ajoute ainsi aux fonctions d'émission et de réception que l'on veut exécuter sur le même processeur, ce qui nécessite donc une analyse temps réel approfondie. Cette capacité de configuration dynamique de notre équipement radio permet non seulement d'être plus robuste face au multi-trajet mais aussi d'adapter l'équipement à tout type de terrain (rural, urbain, montagneux, etc.).

L'un des problèmes rencontré pendant la conception d'une telle radio logicielle est donc de garantir l'exécution simultanée, dans un intervalle de temps borné, des composants logiciels émanant de différentes chaînes (émission, réception, contrôle), sous des contraintes temporelles variables et non prévisibles (ordre d'arrivée des trames arbitraire) sur un même processeur. Pour faire face à ce problème le concepteur doit disposer d'une bonne méthodologie de conception.

1.3 Méthodologies de conception

Une méthodologie de conception correspond à un ensemble de méthodes que doit suivre le concepteur afin de réaliser son logiciel. Il existe deux grandes catégories de méthodologies de conception : les méthodologies de conception systématiques et les méthodologies de conception formelles. Les méthodologies de conception formelles utilisent majoritairement des formules et notations mathématiques afin de représenter le logiciel et vérifier sa consistance. Les méthodologies systématiques utilisent moins de notations mathématiques, et consistent en un ensemble de méthodes procédurales qui décrivent comment la structure du logiciel doit être représentée. Depuis les années 70, il y a eu une augmentation des méthodologies de conception. Différentes méthodologies ont été développées pour résoudre différents types de problèmes. En décrivant ces problèmes, il est souvent nécessaire de les regrouper en problèmes ayant des caractéristiques similaires. Un processus de conception typique consiste à la définition des **exigences**, la **conception préliminaire**, la **conception détaillée**, le **codage**, les **tests unitaires**, les **tests d'intégration** et les **tests de validation**. Durant la conception les étapes préliminaires au développement sont formalisées dans le but de rendre le développement fidèle aux besoins du client. La phase de conception définie donc de manière précise le fonctionnement du système. Très souvent, le concepteur utilise un langage de modélisation pour cela. Par conséquent la conception d'un logiciel est généralement basée sur un jeu de modèles. Les modèles sont basés sur des langages et chaque langage possède une syntaxe et une sémantique.
Nous allons à présent présenter les concepts généraux de conception.

1.3.1 Concepts généraux de conception

Les concepts généraux rencontrés dans la conception de tout système logiciel sont : *l'abstraction, l'encapsulation, la modularité et la hiérarchisation*. Ils fournissent de bonnes pratiques de conception. Ces concepts ont été présentés par Booch [2] dans le contexte de l'approche orientée objet, mais sont assez génériques pour être appliqués dans toute approche de conception dans le domaine de l'ingénierie.

- **L'abstraction**

L'abstraction correspond à la séparation entre les éléments pertinents d'un système à un certain niveau de la conception, et ceux moins pertinents à ce même niveau. L'objectif est d'augmenter la compréhension du système et diminuer sa complexité [2] [3]. Une abstraction est formée en réduisant les informations contenues dans le système. Typiquement, seules les informations importantes à un niveau précis de conception sont retenues. Ainsi, le concepteur peut se concentrer sur quelques informations à chaque étape de la conception. Un système peut avoir plusieurs niveaux d'abstractions où sont exposés une certaine quantité d'informations à chaque niveau. La simplification que fournit une bonne abstraction facilite la réutilisation en produisant un concept qui peut être facilement appréhendé et ainsi utilisé dans d'autres systèmes. Cependant plus un niveau d'abstraction est élevé, plus l'écart entre la spécification et sa mise en œuvre est

grand. Il est donc important de définir le bon jeu d'abstractions depuis la spécification jusqu'à la mise en œuvre.
Une abstraction identifie et regroupe les informations communes d'une entité. Un bon exemple d'abstraction est le modèle OSI pour les systèmes de communications. Il est composé de 7 niveaux d'abstraction et chaque niveau aborde différents besoins du système de communications.

- **L'encapsulation**

Booch [2] définit l'encapsulation comme étant le processus consistant à compartimenter les éléments d'une abstraction qui constituent sa structure et son comportement ; l'encapsulation sert à séparer l'interface contractuelle d'une abstraction de son implantation. Cette définition est considérée dans beaucoup de documents comme étant la référence pour la définition d'une encapsulation. L'encapsulation consiste donc à cacher la vue interne d'une abstraction afin de réduire sa complexité, augmenter sa robustesse et permettre au concepteur de limiter l'interdépendance entre les composants (logiciels dans le cas considéré). Ceci permet de protéger l'intégrité du composant. Les utilisateurs du composant ne peuvent pas modifier les données propres au composant vers une situation invalide ou inconsistante.

- **La modularité**

La modularité consiste à décomposer un système en un ensemble de composants appelés modules. Chaque module permet d'accomplir une fonctionnalité. Les modules permettent la séparation des préoccupations et augmentent la maintenabilité du système en imposant des bornes logiques entre les modules. Les modules sont intégrés dans le système grâce à des interfaces et chaque interface exprime les éléments que fournissent le module et les éléments qu'il nécessite. Un système modulaire est de loin plus réutilisable et plus facile à assembler qu'un système non modulaire. Ainsi, les éléments d'une équipe n'ont pas forcément besoin de connaître le système dans son intégralité pour travailler sur différentes sous-parties du système. Ils peuvent se concentrer juste sur la tâche qui leur a été assignée.
Cependant un compromis doit être trouvé entre le niveau de granularité d'un module et le nombre de modules. En effet lorsqu'il y'a trop de modules, il y'a trop d'interfaces, ce qui est bien plus difficile à gérer qu'un système avec peu de modules.

- **La hiérarchisation**

La hiérarchisation consiste à établir un ordre à travers les différents niveaux d'abstractions [2]. Le système est décomposé en niveaux d'abstraction hiérarchiques. A partir d'un niveau supérieur, on peut décliner un/plusieurs niveau(x) inférieur(s). Les informations et comportements spécifiés au niveau supérieur sont hérités et utilisés au niveau inférieur. La notion d'héritage utilisée en programmation orientée objet (notamment en langage C++) est un bon exemple de hiérarchisation. En effet les classes filles sont dérivées des classes mères par héritage. Elles possèdent les attributs et méthodes définies dans les classes mères.

1.3.2 Les langages

Les langages de programmation et de modélisation ont été développés dans le but d'élever le niveau d'abstraction auquel la conception des systèmes est effectuée. Ils permettent de mieux gérer la complexité croissante des systèmes embarqués et d'accroître la productivité. En effet avec par exemple les langages de programmation, nous pouvons réduire le temps de développements et les coûts puisque moins de lignes de code sont nécessaires pour réaliser la même fonctionnalité (par exemple le passage de l'assembleur aux langages plus évolués comme le langage C). Par ailleurs les langages de modélisations facilitent la génération automatique de squelette de code et l'exploration de différentes solutions, ce qui permet d'augmenter considérablement la productivité.

- **Langages de modélisation**

Le langage UML

Le langage UML (Unified Modelling Language) est un langage de modélisation standardisé et maintenu par l'OMG (Object Management Group) depuis 1997. Ce langage, supporté de manière graphique, est défini par un métamodèle UML, lui-même défini à partir du MOF (Meta Object Facility) [4]. Ce dernier décrit les informations utilisées par les métamodèles. Ces mêmes métamodèles décrivent la sémantique (définition et sens d'utilisation) des éléments employés dans les modèles et leurs relations. UML permet aux concepteurs de modéliser un système complexe suivant plusieurs aspects tel que leurs caractéristiques structurelles et comportementales.

UML spécifie 15 diagrammes divisés en deux principales catégories : modélisation structurelle et modélisation comportementale. Nous n'allons pas décrire tous ces types de diagrammes ici, néanmoins nous présentons une description de quelques uns [2] :

Représentation statique ou structurelle

- Le diagramme de package décrit les dépendances entre les packages qui composent un modèle. Les diagrammes de package peuvent être utilisés pour illustrer les différentes fonctionnalités d'un système ou pour représenter les différentes couches d'un système.
- Le diagramme de composant décrit comment le système est subdivisé en composants et montre les dépendances entre les composants. Les composants sont reliés entre eux en utilisant un connecteur d'assemblage afin de connecter une interface requise par un composant avec une interface fournie par un autre composant.
- Le diagramme de déploiement décrit l'architecture matérielle utilisée dans le système, l'environnement d'exécution ainsi que les éléments déployés sur le matériel.
- Le diagramme de classe décrit la structure du système en montrant les classes du système, leurs attributs, leurs opérations et les relations entre les classes. Le diagramme de classe est utilisé aussi bien pour la conception générale de

l'application que pour la conception détaillée en transformant les modèles en code sources.

- Le diagramme de structure composite montre la structure interne d'une classe et les collaborations que cette structure rend possible. Une structure composite est un ensemble d'éléments qui collaborent à l'exécution pour atteindre un but précis. Chaque élément a un rôle défini dans la collaboration.
- Le diagramme d'objets décrit une vue partielle ou complète d'une structure du système modélisé à un instant précis.

Représentation comportementale

- Le diagramme d'activité fournit une description étape par étape des flux d'activités d'un système. Il montre l'intégralité du flux de contrôle.
- Le diagramme de séquence montre comment les processus interagissent entre eux et dans quel ordre. Il est composé de lignes de vie, d'opérations et permet de spécifier un scénario d'exécution simple.
- Le diagramme de cas d'utilisation décrit les fonctionnalités fournies par le système en termes d'acteurs, leurs buts (représentés sur forme de cas d'utilisations) et toutes les dépendances entre les cas d'utilisations.
- Le diagramme de machine à états exprime le comportement du système sous forme de machines à états finis. Le passage d'un état à un autre est déclenché par des évènements.
- Le diagramme temporel montre les variations d'une donnée au cours du temps. Il s'agit d'une forme spéciale des diagrammes de séquence où les axes sont inversés de manière à ce que le temps s'écoule de gauche vers la droite. Les lignes de vies sont présentées dans des compartiments séparés, rangés de manière verticale.
- Le diagramme de communication décrit les interactions entre les objets en termes de séquences de messages. Il représente la combinaison des informations prises dans des diagrammes de classes, de séquences et de cas d'utilisations en décrivant la structure statique et le comportement dynamique d'un système.

Chacun de ces diagrammes manipule des éléments particuliers de la sémantique UML, ce sont des *« Class »*, des instances de classe *« Object »*, des *« components »* etc. Les différents domaines d'activités qui utilisent UML, disposent des mêmes éléments UML (issus du métamodèle) pour modéliser leurs applications. Or les applications sont très variées, et les objets manipulés différents. UML offre donc un mécanisme de spécialisation des éléments liés aux classes d'applications, qui leur confère une sémantique adaptée à leur domaine. Cette spécialisation s'effectue par le biais de **profils UML** qui apportent des extensions au langage.

Adapter UML : Les profils

Le profil définit la sémantique particulière qui spécialise UML pour un domaine d'activité. Il est composé d'un certain nombre de stéréotypes qui permettent d'attribuer

des caractéristiques particulières (paramètres) aux divers éléments du métamodèle. Plusieurs profils ont été développés et standardisés pour intégrer les aspects temps réel des systèmes électroniques et permettre des vérifications et validations temporelles. C'est le cas des profils (1) *« UML Profile for Schedulability, Performance and Time »* (SPT) [5] et (2) *« UML Profile for Modelling QoS and Fault Tolerance Characteristics and Mechanisms »* (QoS) [6].

Le premier (1) apporte la sémantique des contraintes temps réel utiles aux analyses d'ordonnancement temps réel avec certaines caractéristiques de qualité de services introduites en tant qu'attribut. Le second (2) profil décrit les caractéristiques permettant d'apporter des informations non fonctionnelles à un système en terme de qualité de service (QoS) (latence, sûreté, probabilité d'erreur, etc.), les restrictions qui leurs sont imposées (bornes), et ceci en fonction de plusieurs modes d'exécution. Concernant les profils UML pour la radio logicielle, le profil « UML Profile for SWRadio » [9] a été proposé afin de permettre la modélisation de radio logicielle. Il étend le langage UML 1.4 non seulement pour modéliser, mais aussi pour permettre la génération automatique de fichiers de description (fichier XML) et de code (squelette). Une autre tentative de profil pour la radio logicielle fut le profil A3S (Adéquation Architecture Application Système) [10]. Il permettait d'effectuer des vérifications de cohérence sur des modèles de radio logicielle et était basé sur UML 1.4 or nous utilisons aujourd'hui UML 2. En outre ces profils ne sont pas suffisants pour exprimer l'intégralité des caractéristiques d'un système complet. De plus, les mécanismes d'annotation et d'utilisation des éléments ne sont pas simples et ils ne facilitent pas la construction et la compréhension des modèles. Au-delà du domaine de la radio logicielle un nouveau profil a été développé afin d'offrir davantage de possibilités de développement, de flexibilité et faciliter la modélisation. C'est le profil **MARTE** *« UML profile for Modelling and Analysis of Real-time and Embedded Systems »* [7]**.**

Le profil UML MARTE

Le profil UML MARTE remplace les profils SPT et QoS. Il est compatible avec d'autres standards de l'OMG tel que SysML, SAE AADL (Society for Automotive Engineer Architecture Analysis and Design Language), ARINC 653 (Aeronautical Radio Incoporated). Le profil UML MARTE est composé de trois parties principales : la première partie définit les concepts de base utilisés dans les systèmes temps réel embarqués avec les notions de propriétés non fonctionnelles (*NFP : Non-Functional Properties*), de temps (*Time*), de ressource (*GRM : Generic Resource Modelling*), et d'allocation (*Alloc*). La seconde partie est le modèle de conception temps réel embarqué (*Real-time Embedded Design model*). Elle permet la description de composants logiciels (*SRM : Software Resource Modelling)* et plateformes matérielles (*HRM : Hardware Resource Modelling*). La dernière partie est le modèle d'analyse temps réel embarqué, qui supporte l'analyse quantitative. La Figure 1.2 représente l'architecture du profil MARTE.

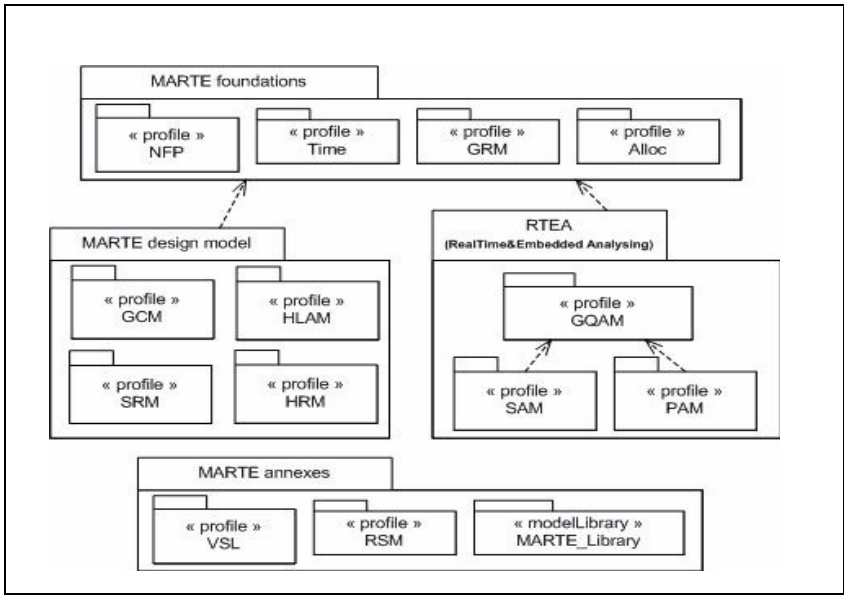

Figure 1.2 – Architecture du profil UML MARTE

- **Langages de programmation**

Un langage de programmation est un langage artificiel conçu pour exprimer des calculs que peuvent effectuer une machine. Un langage de programmation peut être utilisé pour contrôler le comportement d'une carte électronique, ou pour exprimer un algorithme de traitement du signal par exemple.

C/C++

Le langage C est un langage de programmation impératif (procédural) développé par Dennis Ritchie entre 1969 et 1973 au Bell Labs pour utiliser le système d'exploitation UNIX. C'est l'un des langages les plus populaires de tous les temps car il existe beaucoup d'architectures matérielles pour lesquelles un compilateur C existe. Il permet d'effectuer des accès mémoire de bas niveau et nécessite un support d'exécution minimal. Cet un langage très proche des langages machines. Il a beaucoup influencé le langage C++, qui a commencé comme étant une extension du langage C. Le langage C++ a été développé par Bjarne Stroustrup au Bell Labs. Il ajoute au langage C les notions de classes, d'héritage, de polymorphisme, de fonctions virtuelles, de surcharge d'operateurs, de « templates » et de gestion des exceptions. C'est un langage de niveau intermédiaire car il combine les caractéristiques des langages de haut niveau et de bas niveau.

Java

Le langage Java est un langage de programmation développé à l'origine par James Gosling à Sun Microsystems (qui fait parti aujourd'hui d'Oracle). Le langage Java dérive la plupart de sa syntaxe du langage C et C++, mais possède un modèle d'objet simple avec peu de possibilités de bas niveau. Les codes sources java son compilés en « bytecode » et peuvent être exécutés sur n'importe quelle architecture matérielle possédant une « Java Virtual Machine » (JVM).

- **Langage HDL (Hardware Description Language)**

Un langage HDL est un langage de description formelle et de conception des circuits électroniques. Il est composé d'expressions textuelles correspondant à la structure spatiale et temporelle d'un système électronique ainsi que son comportement. Un langage HDL possède une syntaxe et une sémantique explicite pour exprimer la concurrence. Contrairement au langage de programmation logiciel, il possède une notion explicite du temps, qui est un attribut principal du matériel. Les langages HDL sont utilisés pour écrire des spécifications exécutables de certaines parties matérielles. Pour cela, le code HDL est synthétisé en un circuit matériel grâce aux outils de synthèse. L'outil de synthèse (l'équivalent du compilateur en langage de programmation) transforme le code HDL en une liste physique de portes (netlist). On appelle « netlist » les langages dont la seule caractéristique est de décrire les connexions entre les différents blocs d'un circuit. Une étape essentielle de la conception HDL est la capacité de simuler les programmes HDL. En effet la simulation permet de vérifier si la conception effectuée en langage HDL réalise bien la fonctionnalité voulue. Elle permet aussi une exploration de l'architecture car le concepteur peut expérimenter plusieurs choix de conception en écrivant plusieurs variantes de la conception de base et en comparant le comportement en simulation.

Les deux langages HDL les plus répandus et utilisées en industries sont : VHDL (*VHSIC hardware description language ; VHSIC : very-high-speed integrated circuit*) et Verilog.

1.3.3 Les modèles de calculs

Le modèle de calcul (*model of computation*) explique comment le comportement de l'ensemble du système est le résultat du comportement de chacun de ses composants. Le modèle de calcul décrit la sémantique de communication entre les composants. Il peut être utilisé pour mesurer la complexité d'un algorithme en temps d'exécution et en mémoire. En considérant certains modèles de calculs, il est possible de calculer les ressources nécessaires pour exécuter un algorithme ou alors discuter sur ces limites. Les modèles de calculs décrivent l'évolution d'un calcul et sont souvent exprimés comme la description d'une forme d'automate. Plusieurs modèles de calculs ont été identifiés dans la littérature.

Synchronous Data Flow (SDF)

Le flot de données est un paradigme naturel pour décrire les applications de traitement du signal. Pour les applications de traitement du signal, les programmes en flots de données sont des graphes directs où chaque nœud représente une fonction et chaque arc un chemin du signal. Le flot de données synchrone est un cas spécial de flot de données dans lequel le nombre d'échantillons de données produit/consommé par chaque nœud est connu a priori [11]. Les nœuds peuvent donc être ordonnancés de manière statique (à la compilation) dans un seul ou plusieurs processeurs en parallèle. Par conséquent les dépassements observés à l'exécution des flots de donnés sont éliminés. Les taux d'échantillons multiples rencontrés dans un système sont aussi facilement gérer. Le principe d'un flot de données est que chaque nœud ne peut effectuer son calcul que si ses

données en entrées sont disponibles sur son arc d'entrée. Un nœud qui n'a pas d'arc d'entrées peut effectuer son calcul à n'importe quel moment.

Communicating Sequential Processes (CSP)

C'est un langage formel pour décrire les modèles d'interaction dans les systèmes concurrents. Il est bien adapté à la modélisation et l'analyse des systèmes qui comprennent des échanges de messages complexes [12]. Le CSP a été décrit en 1978 par C. A. Hoare mais a depuis considérablement évolué. Il a été appliqué dans l'industrie comme un outil pour la spécification et la vérification des aspects concurrents sur différents systèmes tels que les systèmes sécurisés d'e-commerce.

Kahn Process Networks (KPN)

C'est un modèle de calcul distribué où un groupe de processus séquentiels déterministes communiquent grâce à des canaux FIFO non bornés. Le modèle a été initialement développé pour la modélisation de systèmes distribués [13], mais a prouvé sa convenance pour la modélisation des applications de traitement du signal.

Réseaux de pétri (petri nets)

Un réseau de pétri est un graphe orienté direct biparti, dans lequel les nœuds représentent les transitions (c'est à dire les événements qui peuvent survenir, signifié par des barres) et des lieux (c'est à dire les conditions signifiées par des cercles). Les arcs orientés indiquent quels lieux sont des pré ou post conditions aux transitions. Les réseaux de pétri ont une définition mathématique exacte de leur sémantique d'exécution, avec une théorie bien développée pour l'analyse des processus.

Automates à états finis (Finite State Machine)

Il s'agit d'un modèle de comportement composé d'un nombre fini d'états, les transitions entre ces états, et les actions. Il est similaire à un graphe de flot dans lequel on peut inspecter la manière avec laquelle la logique s'exécute lorsque certaines conditions sont remplies. Le fonctionnement d'un automate à états finis commence à partir de l'un de ses états (appelé état de départ), passe par des transitions en fonction des entrées aux différents états et se termine sur l'un des états disponible. Cependant seul un certain nombre d'états (états finaux) marquent un bon déroulement de l'opération.

Des outils ont été développés pour modéliser et simuler les modèles de calculs, et étudier leur hétérogénéité. En guise d'exemple, on peut citer : **Ptolemy** [14].

1.3.4 Les modèles de communications

- **Le modèle OSI**

Le modèle OSI (OSI – Open System Interconnection) est un moyen de subdiviser un système en sous parties (appelées couches) dans le domaine des communications. Une

couche est une collection de fonctions similaires sur le plan conceptuel qui fournit des services à la couche au–dessus et reçoit des services de la couche en dessous.

Le modèle OSI est composé de 7 couches.

La couche physique

La couche physique définit les spécifications électriques et physiques pour le transfert d'un flux de *bits* sur un média de communication. Elle est responsable de l'établissement et de la terminaison de la communication sur le média de communication. Elle effectue la modulation, puis la conversion entre les données numériques dans l'équipement de l'utilisateur et les signaux transmis à travers le média de communication.

La couche liaison de données

Elle est responsable des communications entre les nœuds adjacents d'un réseau. Elle est en outre responsable de la surveillance, la correction du flux de trames ainsi que des erreurs qui se glissent dans la transmission de trames. Dans certains réseaux, tels que les réseaux locaux (LAN : Local Area network) la couche liaison de données se compose de deux sous-couches : la sous-couche de contrôle de liaison logique (LLC – Logical Link Control), et la sous-couche de contrôle d'accès au média (MAC – Medium Access Control). Comme exemple de couche liaison de données, on peut citer l'Ethernet pour les réseaux LAN, ou le PPP (Point-to-Point Protocol).

La couche réseau

La couche réseau fournit les moyens de transférer des données de tailles variables d'une machine hôte source sur un réseau vers une machine destinataire sur un autre réseau, tout en maintenant la qualité de service demandée par la couche transport. Elle effectue les fonctions de routage, et pourrait aussi effectuer la fragmentation, le réassemblage ou fournir les erreurs de livraisons. C'est à ce niveau que travaillent les routeurs.

La couche transport

La couche transport fournit les services de communications de bout en bout pour les applications entre deux systèmes sur le réseau. Elle est chargée de transmettre le message complet d'un processus sur la machine hôte à un autre processus sur la machine destinataire. Elle divise le message en paquets numérotés qui, après la transmission, sont rassemblés dans le bon ordre. La couche transport fournit des services tel que la communication en mode connecté (TCP/IP), la fiabilité, le contrôle de flot, le multiplexage.

La couche session

La couche session fournit le mécanisme d'ouverture, de clôture et la gestion d'une session entre les processus de l'application finale, à savoir un dialogue semi permanent. Les sessions de communications sont composées de requêtes et de réponses qui se produisent entre les applications.

La couche présentation

La couche présentation est responsable de la livraison et du formatage de l'information à la couche application pour un traitement ultérieur ou pour l'affichage. Il soulage la couche application de la préoccupation concernant les différences de syntaxe dans la représentation des données du système de l'utilisateur final.

La couche application

La couche application du modèle OSI est la couche la plus proche de l'utilisateur final. Ce qui signifie que la couche application et l'utilisateur interagissent directement avec l'application logicielle. Les fonctions réalisées par la couche application comprennent : l'identification des protagonistes de la communication, la détermination des ressources disponibles, la synchronisation de la communication. Comme exemple de couche application, nous pouvons citer un lecteur audio/vidéo tel que VLC.

1.3.5 L'ingénierie dirigée par les modèles

L'ingénierie dirigée par les modèles (MDE – Model Driven Engineering) est une méthodologie de développement logiciel/matériel qui se concentre sur la création de modèles, ou des abstractions, plus proche de certains concepts d'un domaine particulier plutôt que des concepts informatiques ou algorithmiques. Elle a vocation à accroître la productivité en maximisant la compatibilité entre les systèmes, ce qui simplifie le processus de conception, et de promouvoir la communication entre les ingénieurs et les équipes travaillant sur le système.

Les modèles sont développés grâce à une communication étendue entre les encadrants, les concepteurs, et les membres de l'équipe de développement. La meilleure initiative MDE connue est l'architecture dirigée par les modèles (MDA – Model Driven Architecture), initiative de l'OMG.

1.3.6 L'approche MDA

L'approche MDA est une démarche de développement proposée par l'OMG, permettant de séparer deux visions extrêmes du même système : ses spécifications fonctionnelles d'une part, sa mise en œuvre physique d'autre part, incluant plusieurs aspects de la vie du logiciel, à savoir ses tests, ses exigences qualité, la définition des itérations successives, etc.

Pour ce faire, l'approche MDA propose de structurer les besoins selon une architecture indépendante du contexte technique, avant de se livrer à une transformation de cette modélisation fonctionnelle en modélisation technique, tout en testant chaque modèle produit. La mise en œuvre du processus MDA peut se découper en 4 étapes :

1. La réalisation d'un modèle indépendant de toute plate forme appelée PIM (Platform Independent Model),
2. L'enrichissement de ce modèle par étapes successives,
3. En fonction des modèles de description de plate forme (PDM – Platform Description Model), un choix d'une plate forme de mise en œuvre est effectué

puis, s'en suit la génération du modèle spécifique correspondant appelé PSM (Platform Specific Model),

4. Le raffinement de celui-ci jusqu'à l'obtention d'une implantation exécutable.

La Figure 1.3 décrit ces différentes étapes.

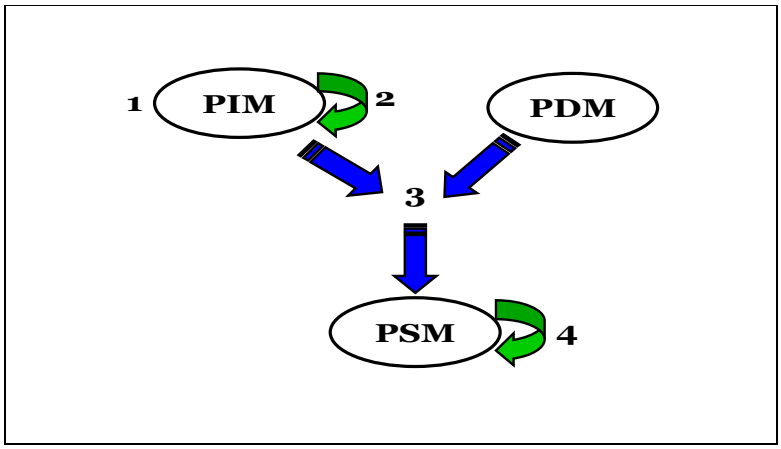

Figure 1.3 – Les étapes d'une démarche MDA

- **Modèle indépendant de la plate-forme (PIM)**

La première étape du processus MDA consiste à réaliser un modèle indépendant de toute plate forme. Il existe plusieurs niveaux de PIM mais tous sont indépendants de n'importe quelle plate-forme. Le PIM de base représente uniquement les capacités fonctionnelles métiers et le comportement du système, sans « dégradations » par des considérations technologiques. La clarté de ce modèle doit permettre à des experts du domaine de le comprendre bien mieux qu'un modèle de mise en œuvre. Ils peuvent ainsi vérifier plus facilement que le PIM est complet et correct.

Un autre bénéfice de l'indépendance technique de ce modèle est qu'il garde tout son intérêt au cours du temps et doit être modifié uniquement si les connaissances ou besoins métiers changent.

Une fois le PIM suffisamment détaillé, il est projeté vers un modèle spécifique.

- **Modèle spécifique à la plate-forme (PSM)**

Pour obtenir un modèle spécifique, il faut choisir la ou les plates-formes d'exécution (plusieurs plates-formes peuvent être utilisées pour mettre en œuvre un même modèle). Les caractéristiques d'exécution et les informations de configuration qui ont été définies de façon générique sont converties pour tenir compte des spécificités de la plate-forme.

Comme pour les PIM, il existe plusieurs niveaux de PSM. Le premier, est obtenu directement à partir du modèle PIM, les autres sont obtenus par transformations successives jusqu'à l'obtention du système exécutable.

- **Les bases techniques**

Au cœur du MDA, se trouvent plusieurs standards importants de l'OMG : UML, le Meta Object Facility (MOF), le XML Metadata Interchange (XMI) et le Common

Warehouse Metamodel (CWM). Ces standards définissent l'infrastructure du MDA. Ils sont complétés par des règles de transformation de modèles.

Meta Object Facility (MOF)

Le langage MOF fournit le standard de métamodélisation et d'échange de constructions utilisées par MDA. Les autres modèles standards de l'OMG, comme UML et CWM sont définis par des constructions MOF, ce qui permet de les relier entre eux. C'est également le mécanisme par lequel les modèles sont sérialisés en XMI. Le MOF est un exemple de métamétamodèle. Il définit les éléments essentiels, la syntaxe et la structure des métamodèles utilisés pour construire des modèles. La spécification MOF fournit les points suivants :

- Un modèle abstrait d'objets MOF génériques et leurs associations.
- Un ensemble de règles pour exprimer un métamodèle MOF à l'aide d'interfaces IDL.
- Un ensemble de règles sur le cycle de vie, la composition et la fermeture sémantique des éléments d'un métamodèle MOF.
- Une hiérarchie d'interfaces réflexives permettant de découvrir et manipuler des modèles basés sur des métamodèles MOF dont on ne connaît pas les interfaces.

Un intérêt du MOF est qu'il permet de faire interopérer des métamodèles différents. Une application MOF peut manipuler un modèle à l'aide d'opérations génériques sans connaissance du domaine.

XML Metadata Interchange (XMI)

Le langage XMI permet de décrire une instance du MOF sous forme textuelle, grâce au langage eXtensible Markup Language (XML) du W3C (World Wide Web consortium). XMI définit comment utiliser les balises XML pour représenter un modèle MOF en XML. Les métamodèles MOF sont décrits par des DTDs (Document Type Definition). Le XMI résout beaucoup de problèmes rencontrés lorsque l'on veut représenter des objets et les associations sous forme textuelle. De plus, puisque XMI est basé sur XML, les métamodèles (tags) et les instances (elements) sont regroupés dans le même document, ce qui permet à une application de comprendre les instances grâce à leurs métadonnées. Le XMI est le format d'échange standard entre les différents outils MDA.

Common Warehouse Metamodel (CWM)

Le CWM est le standard de l'OMG pour les techniques liées aux entrepôts de données. Il couvre le cycle de vie complet de modélisation, construction et gestion des entrepôts de données. Le CWM définit un métamodèle qui représente les métadonnées aussi bien métiers que techniques qui sont le plus souvent trouvées dans les entrepôts de données. Il est utilisé à la base des échanges de métadonnées entre systèmes hétérogènes.

Le CWM comprend actuellement un certain nombre de métamodèles concernant les entrepôts de données (représentation des données, analyse, gestion). Les métamodèles de données permettent de modéliser des ressources comme les bases de données relationnelles, les bases de données orientées objet. Une couche d'analyse du CWM définit des métamodèles pour les transformations de données, la visualisation, la

nomenclature et le datamining. Une couche de gestion est constituée de métamodèles représentant les processus standards, la journalisation et la planification des activités.

Finalement, le métamodèle de base définit les services et éléments communs, comme les types de données, les projections vers les types de systèmes, les clés et index, les expressions et le déploiement de programmes à base de composants. Il est possible à l'aide d'outils CWM de générer une instance d'entrepôt de données à partir de son modèle.

Transformations de modèles

Les techniques de transformation de modèles sont aussi au cœur du MDA. La transformation de modèle consiste à convertir un modèle source M1 conforme à un métamodèle MM1 en un modèle M2 conforme à un métamodèle MM2 en utilisant des règles de « mapping ». Une transformation de modèle convertit un modèle d'un niveau d'abstraction donnée vers un autre modèle d'un autre niveau d'abstraction tout en gardant les deux modèles synchronisés. Les règles de transformation sont réutilisables, elles peuvent être appliquées à l'ensemble des modèles sources conformes au métamodèle pour lequel les règles de « mapping » ont été définies.

De plus les modèles peuvent être accompagnés de contraintes en OCL (Object Constraint Language) afin de vérifier leur validité tout au long du processus. La transformation automatique de modèles réduit le temps de développement tout en augmentant la qualité du logiciel.

L'OMG a défini un standard pour la transformation de modèle QVT (Query/View/Transformation) [15]. La spécification QVT est organisée en une partie déclarative et une partie impérative. Certaines réalisations de QVT sont développées dans le sous projet « Model to Model Transformation (M2M) » du projet « Eclipse Modelling Project ». ATL (Atlas Transformation Language) est un langage de transformation de modèle et un outil développé par l'INRIA et la société OBEO. ATL est un composant du sous projet M2M et est disponible sous eclipse. Il existe d'autres outils de transformation de langage ayant les mêmes capacités que ceux définies dans QVT tel que Kermeta workbench [89] et Tefkat [78].

1.3.7 La conception de type PBD (Platform-Based Design)

La conception de type PBD [16] consiste à « mapper » successivement le modèle d'une application et le modèle d'une plate-forme à travers plusieurs niveaux d'abstractions jusqu' à l'obtention du système électronique désiré. PDB est une approche « middle-out », où une instance d'une application venant de la liste des applications, et une instance de plate-forme venant de la liste des plates-formes, sont « mappés » ensemble et raffinées par étapes successives afin d'explorer une possibilité de conception. Ce type d'approche à d'abord été proposé par Gajski et Kuhn [18] (Y-chart model). Le modèle de l'application est un modèle abstrait et formel, typiquement un algorithme. Alors que le modèle de plate-forme correspond à une mise en œuvre spécifique.

Le modèle de l'application décrit des fonctionnalités à réaliser et les services requis qui doivent être fournis par le modèle de plate-forme caractérisé par les contraintes de plate-forme.

Méthodologie de conception A3S

La méthodologie A3S (utilisé lors de la conception avec le profil A3S) est basée sur la méthodologie de conception PBD [17] [97]. Cette méthodologie ne requiert que deux types de diagrammes UML : le diagramme d'activité et le diagramme de déploiement. Elle considère que les concepteurs disposent d'une bibliothèque d'IP (Intellectual Property) matérielles/logicielles et propose une spécification en 4 étapes.

- *Modélisation de l'architecture matérielle* : le concepteur peut indifféremment débuter par la modélisation de son (ses) architecture(s) matérielle(s) ou de son (ses) application(s). Une bibliothèque de composants dit « matériels » lui propose un ensemble de composants matériels qu'il peut utiliser pour créer son architecture.
- *Modélisation de l'application* : sachant que le concepteur peut réutiliser des codes logiciels de traitements déjà développés dans d'autres applications, sachant également que les applications peuvent être décomposées en tâches spécifiques, une seconde bibliothèque de composants dit « logiciels », lui propose un ensemble de composants logiciels représentant des fonctionnalités. Il peut donc modéliser sa (ses) application(s) de manière indépendante de toute cible architecturale par l'assemblage de composants logiciels « génériques » sous la forme d'un graphe de tâches.
- *Déploiement* : une fois l'application logicielle et l'architecture matérielle modélisées, le concepteur peut décider des choix d'implantations. Il choisit alors manuellement la réalisation en matériel ou en logiciel de chacune des fonctions du graphe de tâches.
- *Vérifications et Analyse* : A chacune des étapes précédentes, des vérifications sont effectuées pour valider les modèles réalisés.

Méthodologie de conception MOPCOM

La méthodologie MOPCOM [19] [99] est un raffinement de la méthodologie MDA « Y-chart ». Les techniques MDA couplées avec UML sont utilisées pour effectuer de la génération de code (VHDL, C/C++, SystemC). Il prend en entrée des exigences fonctionnelles, non fonctionnelles et en termes d'allocation. La modélisation s'effectue en 3 niveaux d'abstractions.

- *AML (Abstract Modelling Level)* : il a pour but de fournir la description du niveau souhaité de concurrence et de pipeline à travers le « mapping » de bloc fonctionnels sur des plates-formes d'exécution virtuelles.
- *EML (Execution Modelling Level)* : il a pour but de fournir la topologie d'une plate-forme générique définie en terme d'exécution, communication ou stockage de nœuds afin d'effectuer une analyse gros grain.
- *DML (Detailed Modelling Level)* : il a pour but de fournir une description détaillée de la plate-forme afin de procéder à une analyse à grain fin. Il permet la génération de code pour cible matérielle (VHDL) ou logicielle (langage C).

Pour chaque niveau, la méthodologie définit les sous-ensembles de MARTE qui sont obligatoires ainsi que les contraintes de modélisation à respecter.

1.3.8 Outils de conception

Plusieurs outils de conception d'applications existent à ce jour, cet état de l'art s'intéresse uniquement aux outils de modélisation UML, aux outils de transformation de langage et aux outils de vérification d'ordonnancement temps réel pertinents pour la conception radio logicielle.

- **Outils de modélisation UML**

Un outil de modélisation UML est une application qui supporte certaines ou toutes les notations et sémantiques du langage UML. La plupart des outils de modélisation UML permettent :

- La création et l'édition de diagrammes UML.
- La génération de code à partir de modèles et la génération de modèles à partir du code
- L'échange de modèles via les fichiers de format XMI.

Le Tableau 1.1 présente quelques outils de modélisation UML.

Nom de l'outil	Quelques détails
AgroUML	Sous licence BSD, écrit en java, disponible sur toutes les plates-formes supportant java, génère du C++, C#, Java, Php, Python, Ruby
BoUML	Intégré avec Qt3, génération de code java, C++, Php, Python, IDL. Sous licence GNU GPL.
MagicDraw	Intégré avec Eclipse, EMF, netBeans. Supporte UML 2.3, génération de code Java C++, C#, CORBA IDL, DDL, plugin MARTE disponible
Objecteering	Basé sur Eclipse, génération de code Java, C++, C#, SQL DDL, CORBA IDL et Fortran.
Papyrus	Outil open source supportant UML 2, basé sur eclipse sous licence EPL (Eclipse Public Licence)
Rational software Architect	Basé sous eclipse, supporte UML 2, génération de code de UML vers Java, C#, C++, EJB, WSDL, XSD, CORBA, SQL. De Java vers UML, du C++ vers UML, plugin MARTE disponible
Rational Rhapsody	Supporte UML 2 et SysML. Génération de code C, C++, Ada, Java.

Tableau 1.1 – Outils de modélisation UML pour la conception de radio logicielle

En particulier dans le contexte SCA, le plugin eclipse Zeligsoft CX et l'outil Spectra permettent de modéliser une application radio logicielle et générer le code des ports et des conteneurs de composants logiciels compatibles avec le standard SCA.

- **Outils de transformation de langage**

Le Tableau 1.2 présente quelques outils de transformation de langage.

Nom de l'outil	Quelques détails
ATL	Développé par OBEO et l'INRIA sous licence EPL. Utilisé pour la translation sémantique et syntaxique. En réponse au QVT « request for proposal » de L'OMG.
Kermeta workbench	Emprunte les concepts de MOF, OCL et QVT, mais aussi de BasicMTL. Développé par une équipe de L'IRISA (université de Rennes 1), Open source, sous licence EPL.
Tefkat	Basé sur F-logic et la théorie des programmes logiques stratifiés. C'est un plugin Eclipse pour l'Eclipse Modelling Framework.

Tableau 1.2 – Quelques outils de transformation de langage

1.4 Systèmes temps réel et ordonnancement temps réel

L'électronique embarquée temps réel est largement utilisée et prend une importance de plus en plus grande dans la vie quotidienne. Plusieurs domaines d'application sont concernés, notamment, l'automobile, l'avionique, la téléphonie mobile, le multimédia, l'énergie etc.

1.4.1 Introduction aux systèmes temps réel

Les systèmes temps réel concernent des applications ayant un rôle de suivi ou de contrôle de procédés dans un environnement qui évolue dynamiquement. L'application est réactive, en effet, elle doit réagir au changement d'état d'un système soumis à des contraintes temporelles précises. D'où le bon fonctionnement d'un système temps réel ne dépend pas seulement de l'exactitude des résultats du calcul mais aussi des dates auxquelles ces résultats sont produits. Certains systèmes sont classés critiques lorsque le non respect des contraintes temporelles peut provoquer des conséquences catastrophiques (perte de vie humaine, destruction de matériel…). Les systèmes de contrôle de vol, les systèmes de contrôle de centrale nucléaire en sont des exemples.

Généralement, un système temps réel est composé d'un ensemble de tâches exécutées en concurrence pour utiliser le (ou les) processeur(s) et certaines ressources partagées. Le plus souvent les exigences temporelles sont reportées sur les échéances d'exécution des tâches. Par conséquent, une validation temporelle consiste à vérifier que toutes les tâches d'un système terminent leur exécution avant leurs échéances, durant toute la durée de vie du système, et cela en examinant le pire scénario d'exécution de tâches. S'il existe un moyen de garantir la validation temporelle, alors le système est dit ordonnançable. Le test d'ordonnançabilité est dépendant du modèle de tâches qui définit l'ensemble des restrictions auxquelles doivent se conformer les tâches et de la politique d'ordonnancement utilisée pour déterminer l'ordre dans lequel doit être exécuté les tâches.

Dans les systèmes temps réel, le système électronique doit réagir en permanence aux variations du procédé et agir en conséquence sur celui-ci afin d'obtenir le comportement ou l'état souhaité, cette caractéristique définit la notion de réactivité du système vis-à-vis du procédé (environnement auquel il est connecté)[20]. Une définition des systèmes réactifs, décrivant le fonctionnement d'un système temps réel est donnée dans [21] : *Un système réactif est un système qui réagit continûment avec son environnement à un rythme imposé par cet environnement. Il reçoit, par l'intermédiaire de capteurs, des entrées provenant de l'environnement, appelées stimuli, réagit à tous ces stimuli en effectuant un certain nombre d'opérations et produit, grâce à des actionneurs, des sorties utilisables par l'environnement, appelées réactions ou commandes.*

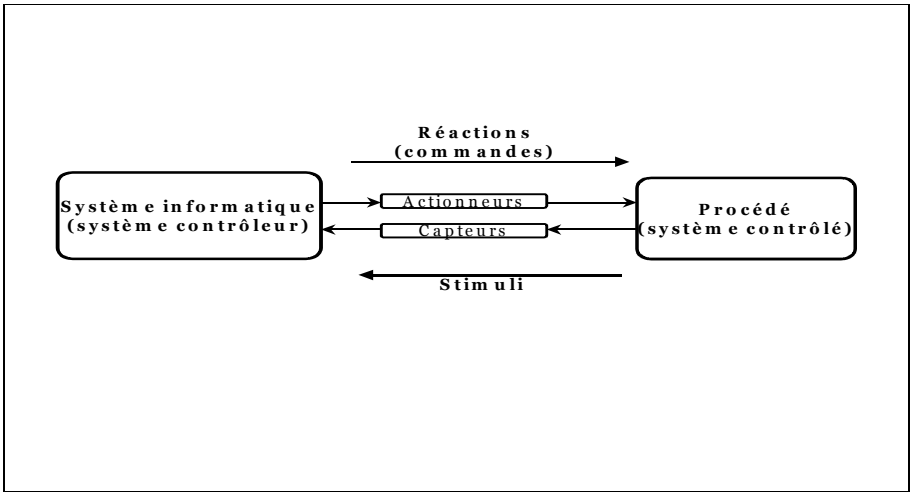

Figure 1.4 – Système temps réel

De cette définition, un système temps réel est alors composé principalement de deux éléments distincts : une ou plusieurs entités physiques constituent le procédé (le système contrôlé) et un système de contrôle (contrôleur) qui est chargé de surveiller de manière régulière l'état du procédé en récupérant les valeurs en provenance des capteurs. En fonction de l'algorithme de contrôle et des valeurs décrivant l'état actuel du procédé, le contrôleur commande les changements d'état, via des actionneurs. Dans le domaine des communications radio, le procédé est la sortie attendue sur l'antenne radio, et le système

contrôleur, un schéma de modulation sur une carte électronique. Notons qu'un système temps réel est dit **embarqué** lorsqu'il est situé à l'intérieur de l'environnement qu'il doit contrôler, comme un calculateur dans une voiture ou un avion.

Dans les systèmes temps réel, les données ont une validité définie à la fois par le domaine de valeurs admises et par la durée de validité (temporelle) de celles-ci, qui dépend naturellement de l'échéance. Les données ont donc une durée d'existence limitée [22]. Les systèmes de contrôle doivent donc respecter deux exigences : fonctionnelles et temporelles en parallèle. Selon les contraintes temporelles, on distingue deux grandes catégories des systèmes temps réel :

- **Systèmes temps réel durs** : Lorsque toutes les contraintes temporelles doivent être impérativement respectées. Le non respect des contraintes peut provoquer des conséquences catastrophiques [23]. Les systèmes de contrôle de vol, les systèmes de contrôle de station nucléaire, systèmes de contrôle de vois ferrées en sont des exemples.
- **Système temps réel souples ou mou** : au contraire des systèmes temps réel durs, le non respect des contraintes temporelles est toléré (acceptable) par le système, et sans que cela ait des conséquences catastrophiques. Par exemple les applications multimédia.

D'autre part, les systèmes temps réel sont considérablement influencés par l'architecture matérielle sur laquelle doit s'exécuter le système. Cette architecture matérielle désigne les ressources physiques nécessaires au pilotage du procédé : les processeurs, les cartes d'entrées/sorties, les mémoires, les réseaux, etc. En fonction du nombre de processeurs et de l'utilisation éventuelle d'un réseau, on distingue trois grandes catégories d'architectures [24] :

- **Architecture monoprocesseur** : Dans ce type de systèmes, toutes les tâches de l'application sont exécutées par un unique processeur partagé.
- **Architecture multiprocesseur :** Les fonctions de l'application sont réparties sur n processeurs partageant une mémoire centrale commune.
- **Architecture distribuée :** Les fonctions de l'application sont réparties sur n processeurs, mais au contraire des architectures multiprocesseurs, il n'y a pas de mémoire centrale partagée. Les processeurs sont reliés les uns aux autres par l'intermédiaire d'un réseau.

Dans la suite nous nous intéressons aux systèmes de contrôle **temps réel durs** implantés dans une **architecture monoprocesseur**.

1.4.2 Caractérisation d'une application temps réel

La méthode la plus couramment utilisée pour mener à bien le dimensionnement d'un système temps réel respecte les étapes suivantes [25]:

– Identifier le modèle de chacune des tâches,
– Identifier le modèle de contrainte temporelle considéré,
– Identifier le modèle d'ordonnancement,
– Identifier les scénarios d'activations,
– Etablir les conditions de faisabilité et vérifier sur les scénarios d'activations.

o Modèle temporel de tâche

Un système temps réel se modélise donc par un ensemble fini de tâches $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. Cet ensemble de tâches découle directement de l'objectif de l'application, c'est à dire de son cahier des charges. Soit τ_i une tâche quelconque du système, τ_i peut être :

- **Périodique** : lorsque ses activations sont périodiques et de période d'activation T_i. La période d'une tâche est la durée fixe entre deux activations des deux instances successives de τ_i. On peut ainsi calculer facilement les dates d'activation de τ_i à travers le temps.
- **Sporadique :** lorsque deux activations successives de cette tâche sont espacées d'une durée supérieure ou égale à T_i.
- **Apériodique :** Lorsque ses activations surviennent à des moments aléatoires.

o Modèle de contrainte temporelle

L'échéance temporelle ou encore **délai critique** notée D_i, correspond au temps alloué à la tâche pour terminer complètement son exécution. Chaque instance de τ_i doit terminer son exécution avant D_i unités de temps après la date de son activation. Le dépassement de cette date limite pour l'exécution produit une faute temporelle.

Définissons maintenant la durée d'exécution au pire cas d'une tâche et son temps de réponse au pire cas.

- **La durée d'exécution au pire cas**, ou WCET (Worst case Execution Time), d'une tâche τ_i est notée C_i. Cette durée d'exécution au pire cas correspond au pire temps d'exécution de la tâche τ_i seule, c'est à dire sans aucune préemption due à l'ordonnanceur ou à toute autre tâche. De nombreux travaux ont traité le problème du calcul de WCET de façon à ce qu'il ne soit pas trop surestimé. Notons que certains systèmes temps réel mou considèrent un temps d'exécution moyen ou minimal. La problématique d'estimation de la durée d'exécution fait l'objet de nombreux travaux [26][27][28][29][30].
- **Le temps de réponse au pire cas** R_i d'une tâche τ_i est la plus longue durée entre une activation de τ_i et la terminaison de l'exécution associée. La tâche τ_i pouvant être éventuellement interrompue pendant son exécution, au profit de tâches plus prioritaires, son temps de réponse au pire cas dépend des caractéristiques des autres tâches du système ainsi que de la politique d'ordonnancement. Dans tous les cas, nous aurons $R_i \geq C_i$.

Une tâche est dite à **échéance sur requête** si $D_i = T_i$. Les tâches d'un système sont à **échéances contraintes** lorsque les échéances sont inférieures ou égales aux périodes. Une tâche est bien formée si cette condition $0 \prec C_i \leq D_i \leq T_i$ est vérifiée. Si la date de la première activation d'une tâche, notée O_i, est connue, la tâche est dite **concrète**. Un jeu de tâches est dit **synchrone** si il existe un instant où toutes ses tâches sont activées simultanément, nous parlons alors d'instant critique. Autrement, il sera dit **asynchrone**.

- **Modèle d'ordonnancement**

Le problème de l'ordonnancement consiste à choisir, si tant est qu'elle existe, une politique d'attribution du processeur aux tâches telle qu'aucune faute temporelle ne sera commise durant toute la durée de vie du système. Deux approches peuvent être distinguées :

- **L'ordonnancement en ligne** : Un algorithme d'ordonnancement est dans le système. Les décisions d'ordonnancement sont alors prises pendant la durée de vie du système.
- **L'ordonnancement hors ligne** : Une séquence valide, définissant l'ordre dans lequel les tâches doivent être exécutées, est déterminée avant le déploiement du système. Une fois chargée dans une table, cette séquence sera utilisée par un séquenceur tout au long de la vie du système.

Les temps de réponse au pire cas obtenus dépendent fortement de la politique d'ordonnancement. Afin de permettre à un ordonnanceur en ligne de déterminer l'ordre dans lequel les tâches doivent s'exécuter, une solution très classique consiste à attribuer une priorité à chacune d'entre elles. Lorsqu'il est invoqué, l'ordonnanceur sélectionne alors, selon l'algorithme HPF (Highest priority First), la tâche la plus prioritaire. Nous distinguons deux types de priorités :

- **Priorité fixe** : La priorité de chaque tâche est fixée lors de la conception du système et demeure invariable tout au long de sa vie
- **Priorité dynamique :** Les priorités des tâches varient au cours de l'exécution du système. Les algorithmes d'ordonnancement dit dynamiques attribuent les priorités aux tâches en fonction des paramètres temporels dynamiques comme l'échéance la plus proche. L'algorithme doit alors recalculer les priorités à chaque instant de décision.
- **Priorité fixe/priorité dynamique :** Dans ce cas l'ordonnancement est dit mixte, ou encore hybride, et utilise les priorités fixe comme premier critère d'ordonnancement puis priorité dynamique comme second critère.

Nous distinguons deux modes d'exécution des tâches temps réel, en fonction des contraintes de ressources et de précédence :

- **Non préemptif :** L'ordonnanceur ne peut interrompre l'exécution d'une tâche (possédant le processeur) en faveur d'une autre tâche. Il doit attendre jusqu'à la terminaison de l'exécution de la tâche en cours, avant de débuter l'exécution de toute autre tâche. Si une tâche non préemptive est interrompue, son exécution doit être reprise de nouveau depuis le début.
- **Préemptif :** L'exécution d'une tâche peut être interrompue par une autre tâche plus prioritaire (la priorité d'une tâche est définie en fonction des algorithmes d'ordonnancement sous-jacents qui seront présentés plus tard dans ce chapitre). Son exécution est reprise ultérieurement.

Lorsque toutes les tâches de l'application sont préemptives, l'ordonnancement est alors dit **préemptif**. Dans ce cas dès qu'une tâche plus prioritaire que la tâche en cours d'exécution est activée, celle-ci se voit alors attribuer le processeur au détriment de la tâche moins prioritaire.

Un algorithme d'ordonnancement est dit optimal par rapport à une classe de problème d'ordonnancement s'il trouve toujours une solution à un problème de cette classe lorsqu'il en existe une. Ainsi, si l'algorithme d'ordonnancement optimal pour une classe donnée ne trouve pas de solution à un problème de cette classe alors celui-ci ne peut être résolu.

1.4.3 Concepts et propriétés

Dans la suite de cette analyse temps réel, le temps sera, le plus souvent, considéré comme **discret** : toutes les dates ainsi que les paramètres des tâches seront entiers et exprimés dans la même unité. L'article [31] montre l'intérêt de ce choix selon les faits suivants :

– La plupart des ordonnanceurs utilisent une horloge comme référence temporelle.
– Si tous les paramètres temporels sont entiers et exprimés dans la même unité, alors il existe un ordonnancement valide en temps discret si, et seulement si, il en existe un en temps continu.

Nous donnons quelques propriétés utilisées dans l'analyse d'ordonnançabilité des systèmes.

– Une séquence d'ordonnancement des tâches d'un système est valide si, et seulement si, toutes les échéances sont respectées.
– Un système de tâches S est ordonnançable si, et seulement si, il existe une séquence d'ordonnancement valide.
– Soit A un algorithme, et S un système de tâches temps réel. S est correctement ordonnancé par A si, et seulement si, la séquence produite par A est valide.
– Un test d'ordonnançabilité est un algorithme qui, pour un système S et un algorithme d'ordonnancement A donné, renvoie une réponse négative si l'algorithme A peut générer une séquence non valide. Cette thèse traite le problème d'analyse d'ordonnançabilité.

Etant donné un système de tâches S, et un algorithme d'ordonnancement A (priorités fixes ou dynamiques). Un problème de décision est posé :

L'algorithme A ordonnance t-il correctement le système S ?

La réponse est sous la forme d'un test a priori d'ordonnançabilité. Cela nécessite une connaissance complète des contraintes temporelles du système, plus l'algorithme d'attribution des priorités choisi. La validation d'un système temps réel revient à montrer que chacune de ses tâches respectera toutes ses échéances temporelles tout au long de la vie dudit système. La validation s'effectue hors-ligne, soit par **simulation** du système de tâches durant une durée suffisante, appelée **durée de simulation** [32], soit à l'aide de conditions d'ordonnançabilité analytiques basées sur les critères temporels des tâches.

1.4.4 Les techniques d'analyse

On distingue trois principales techniques d'analyse

- **L'analyse de l'utilisation du processeur (Processor Utilization Analysis) :**

Le facteur d'utilisation est le pourcentage de temps que le processeur passe à exécuter l'ensemble des tâches du système. Il se définit par $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Dans certains cas particuliers le facteur d'utilisation du processeur permet de conclure si une configuration de tâches est ordonnançable ou non.

- **L'analyse de la demande processeur (Processor Demand Analysis) :**
  Cette technique repose sur le calcul de la demande cumulée des exécutions des tâches réveillées et terminées dans un intervalle de temps (Demand Bound Function). Cette approche générale permet de construire des tests d'ordonnançabilité en limitant l'étude d'ordonnançabilité à quelques intervalles d'une période d'activité du processeur [33][34].
- **L'analyse des temps de réponse (Response Time Analysis) :** Cette technique consiste à calculer les pires temps de réponse des tâches et à les comparer avec leurs délais critiques. Une tâche est ordonnançable si, et seulement si, son pire temps de réponse est inférieur ou égal à son délai critique.

Notons que les techniques de l'utilisation du processeur et de l'analyse du temps de réponse seront utilisées dans cette thèse.
Nous présentons dans les paragraphes suivants quelques résultats d'analyse d'ordonnançabilité basés sur l'étude du facteur d'utilisation processeur.

1.4.5 Algorithmes d'ordonnancement à priorité fixe

Pour les algorithmes d'ordonnancement à priorités fixes, la priorité des tâches est calculée en phase de conception du système et reste la même pendant toute la durée de vie du système. Dans le cadre de ces algorithmes, on trouve plusieurs conditions d'ordonnançabilité basées sur l'instant critique.

Un instant critique d'une tâche correspond à une date d'activation de la tâche produisant le pire temps de réponse de la tâche parmi toutes les occurrences d'activations de la tâche.

Pour des tâches à échéances sur requêtes et indépendantes, Lui et Layland [35] ont montré que l'instant critique pour une tâche a lieu lorsqu'elle est activée en même temps que toutes les tâches plus prioritaires qu'elle.

- **Rate Monotonic (RM)**

L'algorithme de Rate Monotonic [35] a été présenté par Lui et Layland en 1973. A travers cet algorithme, chaque tâche se verra attribuer une priorité inversement proportionnelle à sa période. Ainsi la tâche la plus prioritaire correspondra à la tâche qui aura la plus petite période. En cas d'égalité de priorités pour plusieurs tâches, le choix de la tâche à exécuter sera arbitraire.

Théorème 1.1 [35]
L'algorithme RM est optimal, dans la classe des algorithmes à priorités fixes, pour des systèmes de tâches indépendantes, synchrones, et périodiques à échéances sur requête.

Théorème 1.2 [35]

Un système S composé de n tâches périodiques à échéance sur requête et indépendantes est ordonnançable par RM si :

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

Le deuxième membre de l'inégalité tend vers ln 2 (=0,69) lorsque n tend vers l'infini.

○ Deadline Monotonic (DM)

L'algorithme DM a été proposé par Leung et Whitehead [36] (appelé aussi inverse de l'échéance temporelle), elle attribue les priorités aux tâches inversement proportionnelle à leurs délais critiques. La tâche la plus prioritaire est celle possédant le délai critique le plus court. En cas d'égalité de priorités, le choix de la tâche à exécuter est fait arbitrairement.

Théorème 1.3 [36]

L'algorithme DM est optimal, dans la classe des algorithmes à priorités fixes pour les systèmes de tâches indépendantes, synchrones et à échéances inférieures aux périodes ($D_i \leq T_i$).

Lorsque les tâches sont à échéances sur requêtes ($D_i = T_i$), on se ramène au même ordre de priorité produit par l'algorithme RM.

Théorème 1.4 [36]

Un système S composé de n tâches périodiques à échéance sur requête et indépendantes est ordonnançable par DM si :

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

○ L'algorithme d'Audsley

Aucun des deux algorithmes présentés précédemment n'est optimal en général parmi les algorithmes à priorités fixes. En effet, en contexte préemptif, l'algorithme RM n'est optimal que pour les ensembles de tâches synchrones à échéances sur requête. De même, la condition $D_i \leq T_i$ doit être vérifiée pour que l'algorithme DM soit optimal. L'algorithme proposé par N. C. Audsley [37] comble cette lacune et apporte toujours une solution valide d'attribution de priorité, si tant est qu'il en existe une, sur un ensemble de tâches indépendantes et asynchrones. Le pseudo code de l'algorithme d'Audsley est donné ci-dessus.

Optimal priority Assignment Algorithm

```
for each priority level i, lowest first
        found = false
        for each unassigned task τ

                if τ is schedulable at priority I

                found = true
                assign τ to priority I
                break (continue outer loop)
                end if
        end for
        if not found return unschedulable
end for

return schedulable
```

1.4.6 Algorithmes d'ordonnancement à priorité dynamiques

Ils déterminent la priorité des tâches pendant l'exécution du système.

- **Earliest Deadline First (EDF)**

Parmi les algorithmes à priorités dynamiques, EDF est vraisemblablement le plus populaire. Cet algorithme repose sur le principe qu'à chacune de ses invocations l'ordonnanceur élit la tâche dont l'échéance est la plus proche.

Théorème 1.5 [38][39]

L'algorithme EDF est optimal pour l'ordonnancement des tâches périodiques ou sporadiques en contexte préemptif ainsi qu'en contexte non préemptif non concret. Cette optimalité n'est plus vraie en contexte non préemptif concret.

Pour les tâches synchrones à échéances sur requête, l'algorithme EDF présente l'avantage de posséder une condition nécessaire et suffisante d'ordonnançabilité simple à mettre en œuvre.

Théorème 1.6 [35]

Un système S composé de n tâches synchrones à échéances sur requêtes est ordonnançable par EDF si, et seulement si :

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

1.4.7 Ordonnancement des tâches dépendantes

L'étude de l'ordonnancement qui a été présentée dans la section précédente suppose un système monoprocesseur assez restrictif. Toutes les tâches sont indépendantes (sans aucune relation de précédence entre elles) et aucun partage de ressources n'est considéré. Nous présentons brièvement dans les sous-sections qui suivent les extensions apportées à l'étude d'ordonnancement des modèles de tâche, en prenant en compte plus de contraintes.

- **Contraintes de précédence**

La majorité des systèmes temps réel nécessitent des communications entre les tâches. Une tâche (réceptrice) en cours d'exécution, à un point de communication, est mise en attente d'une condition (message) jusqu'à la satisfaction de la condition (arrivée du message). Ainsi l'exécution de la tâche réceptrice doit être précédée par l'exécution de la tâche émettrice, ce qui introduit des contraintes de précédence entre les tâches.

- **Ressources partagées**

Il est indispensable d'intégrer le partage de ressources aux algorithmes d'ordonnancement car rares sont les applications temps réel qui n'utilisent pas de ressources critiques. Chaque ressource possède une capacité d'entrée, cette capacité représente le nombre d'accès simultané permis (accepté) par la ressource. On dit qu'une ressource est critique lorsqu'elle n'accepte qu'un seul accès à la fois. Une présentation détaillée des ressources critiques et des sémaphores est fournie dans [42]. Mok [41] a montré que le problème de l'ordonnancement en présence de ressources est NP-difficile au sens fort. De plus, deux phénomènes peuvent se produire lors de l'ordonnancement d'un système de tâches avec partage de ressource :

- **L'interblocage**

Ce phénomène se produit lorsque deux ou plusieurs tâches sont bloquées car chacune a demandé l'accès à une ressource en possession de l'autre. Supposons par exemple deux tâches τ_1 et τ_2 en partage de deux ressources en accès exclusif R1 et R2. Un interblocage survient lorsque τ_1 possède R1 et τ_2 possède R2 et que τ_1 et τ_2 ont à leur tour besoin de R2 et R1 respectivement.

- **Inversion de priorité non borné**

Ce phénomène survient lorsqu'une tâche, plus prioritaire, est bloquée par l'exécution des tâches moins prioritaires alors qu'elle ne partage pas de ressource, avec certaines des tâches moins prioritaires. Une tâche τ_1 de priorité supérieure, est bloquée en attente d'une ressource en possession d'une tâche moins prioritaire τ_2, cette dernière (τ_2) peut être préemptée par toutes tâches τ_j, de priorité intermédiaire (entre celle de τ_1 et τ_2), et qui n'est en conflit, ni avec τ_1, ni avec τ_2. Cette préemption de l'exécution de τ_2 produit un blocage non borné pour τ_1. Ce phénomène d'inversion de priorité doit être pris en considération lors de la conception afin de mener une bonne analyse d'ordonnançabilité.
Les phénomènes d'interblocage et d'inversion de priorité sont représentés à la Figure 1.5.

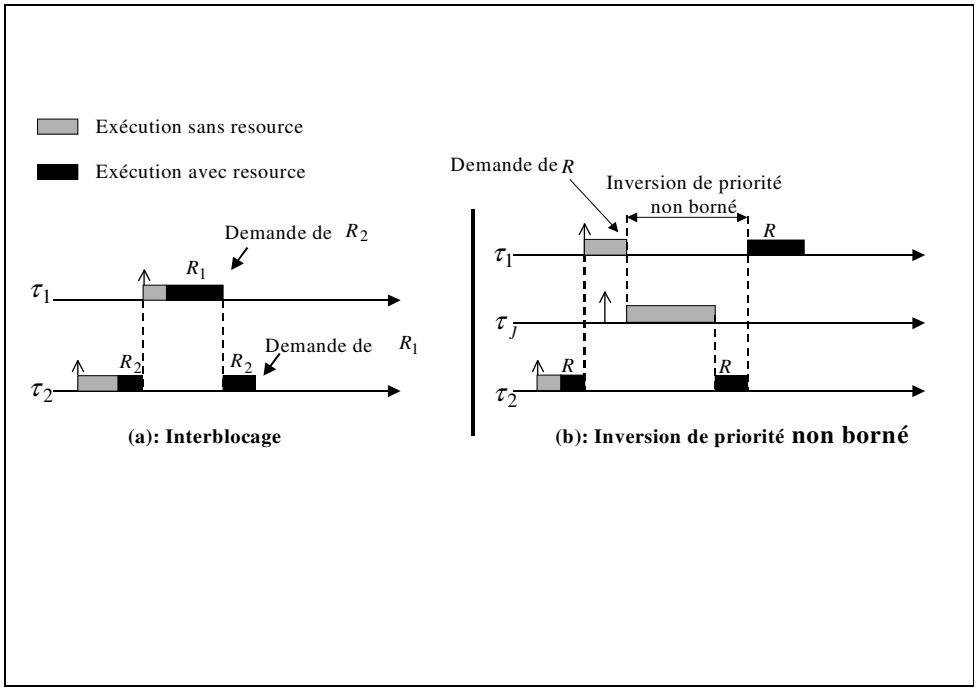

Figure 1.5 - Interblocage et inversion de priorité non bornée

Dans le but d'empêcher ou de borner l'inversion de priorité, plusieurs protocoles de gestion de ressources ont été proposés dans la littérature.

Protocole d'héritage de priorité

Le PPH (Priority Inheritance Protocol) [43] a été proposé pour palier le problème d'inversion de priorité non bornée. L'idée simple consiste à interdire toute préemption, de la tâche possédant une ressource, demandée par les tâches plus prioritaires (tâches bloquées), par toute tâche de priorité intermédiaire. Pour cela on affecte, à la tâche bloquante, la plus grande priorité parmi les priorités de toutes les tâches qu'elle bloque (on dit qu'elle hérite d'une priorité). Lorsque la tâche bloquante libère une ressource, sa priorité est recalculée en fonction des ressources qui sont toujours en sa possession. Elle reprend sa priorité initiale lorsqu'elle libère toutes les ressources. Notons que l'héritage de priorité est transitif, tel que si une tâche τ_k bloque une tâche τ_j qui elle même bloque une autre tâche τ_i, alors la tâche τ_k hérite de la priorité de τ_i par l'intermédiaire de τ_j. On parle dans ce cas de blocage transitif. Une tâche ne peut être bloquée à l'entrée d'une section critique qu'une fois par ressource existante. Ainsi le temps de blocage est la somme des plus longues sections critiques sur chaque ressource.

Théorème 1.7 [43]

En utilisant le protocole à priorités héritées, une tâche τ_i d'un système S ne peut être bloquée que pendant un temps égal à :

$$\sum_{R,ressource} \max_{j \neq i} \{\text{durée d'utilisation de R par } \tau_j\}.$$

Avec cette méthode, le temps de blocage est élevé si plusieurs tâches partagent une même ressource. Il ne peut être calculé que si les tâches possèdent des priorités fixes. Ainsi, ce protocole n'est utilisable qu'avec des algorithmes d'ordonnancement à priorités fixes. Notons enfin que l'utilisation de PPH ne résout pas le problème d'interblocage.

Protocole de priorité de plafond

Le PPP (Protocole de Priorité de Plafond) a été présenté, dans les travaux de Sha et al [43], afin de résoudre les limitations de PPH. Le principe consiste à affecter à chaque ressource une priorité plafond (seuil), qui est la plus forte des tâches qui y accèdent. Le mécanisme du protocole est le suivant :

– Lorsqu'une tâche accède à la ressource, la priorité de la tâche se voit augmentée à la priorité de plafond associée à la ressource en question. La priorité de plafond étant supérieure ou égale à la priorité maximale relevée sur l'ensemble des tâches utilisant la ressource. Après modification de sa priorité la tâche courante ne peut en aucun cas être préemptée par une tâche demandant l'accès à cette même ressource qui se trouve ainsi correctement protégée. Cependant certaines tâches qui n'emploient pas la ressource peuvent avoir une priorité strictement supérieure à sa priorité de plafond et préempter la tâche courante.
– Lorsqu'une tâche libère la ressource, la priorité de la tâche courante est ramenée à sa valeur antérieure. Une fois le changement effectué une tâche peut être éventuellement élue.

Notons que ce protocole est destiné aux algorithmes d'ordonnancement à priorités fixes. Cependant il a été étendu dans [45] pour l'utiliser avec EDF. Le protocole résultant est nommé protocole à priorité de plafond dynamique. Le PPP permet d'empêcher tout interblocage et garantit qu'une tâche ne puisse être retardée que par, au plus, une tâche moins prioritaire. Le lecteur peut se référer à l'annexe A, pour avoir des informations complémentaires.

Le modèle de tâches étudié dans le cadre de cette thèse se rapporte à la catégorie des tâches multi-trames c'est pourquoi nous nous y intéressons particulièrement dans cet état de l'art.

1.4.8 Analyse de l'ordonnançabilité des tâches multi-trames

Le modèle de tâches périodiques proposé par Lui et Layland [35] suppose la prise en compte uniquement du pire temps d'exécution de chaque tâche. Cette situation est acceptable dans la modélisation de nombreuses applications temps réel. Cependant, lorsque le temps d'exécution moyen d'une tâche est très faible devant le pire temps d'exécution, cette considération devient très pessimiste. En fait la prise en compte du pire temps d'exécution conduit à une surestimation de la charge du processeur, ce qui réduit le taux d'acceptation (validation) des systèmes. Pour calculer le taux d'utilisation réel des ressources du système, d'autres modèles de tâches ont été proposés. C'est le cas du modèle multi-trames (multiframe - MF) introduit par Mok et Chen [60]. Ce modèle prend en compte des temps d'exécution d'une tâche variables d'une instance à l'autre. Le modèle multi-trames utilise alors la liste des temps d'exécution pour établir les conditions

d'ordonnançabilité. La liste des temps d'exécution se répétant périodiquement suivant un motif d'activation.

Le type d'applications concrètes dans lesquelles on retrouve ce genre de tâches est la transmission de trames vidéo codées MPEG [61]. Dans le codage standard MPEG (Moving Picture Experts Group), il y a trois types de trames vidéo : les I-frame, les P-frames et les B-frames. Les I-frames utilisent un encodage intra-frame. Les P-frames et B-frames utilisent un encodage inter-frame. Les trames sont encodées en fonction de données précédentes, courantes ou suivantes. Toutes les transmissions de trames présentent alors un motif fixe comme celui-ci : « **I**BBPBBPBB**I**BBP.. ». Le motif de l'envoi des trames est périodique et chaque période débute par l'envoi d'une trame I.

Présentation du modèle

Une tâche multi-trames est définie par un couple (Γ_i, T_i) où Γ_i est un tableau de N_i $(N_i \geq 1)$ trames avec temps d'exécution $(C_i^0, C_i^2, ..., C_i^{N_i-1})$ et T_i est le temps minimum de séparation entre deux trames successives. Chaque trame doit terminer son exécution avant son délai critique $D_i = T_i$.

Analyse basée sur la densité pour des tâches à priorités fixes

Soit τ_i une tâche multi-trame composée de N_i trames, avec des temps d'exécution $(C_i^0, C_i^2, ..., C_i^{N_i-1})$ respectivement. On dit que τ_i est accumulativement monotonique (AM) si $\exists m \in (0 : N_i - 1)$ tel que $\forall l, j \in (0 : N_i - 1), \sum_{k=m}^{m+l} C_i^k \geq \sum_{k=j}^{j+l} C_i^k$. On appelle la trame m, la trame de pointe de τ_i.

Notons que la trame de pointe *m* est la trame ayant la plus grande durée d'exécution C_i^m dans Γ_i. Une tâche AM est une tâche pour laquelle la plus grande quantité de temps d'exécution totale de n'importe quelle séquence de *l* ($l \geq 1$) trames correspond à celle dans laquelle la première trame est la trame de pointe.

Supposons que toutes les tâches soient AM. Un instant critique produisant le pire scénario d'exécution d'une tâche survient lorsque sa trame de pointe s'active simultanément avec toutes les trames de pointe de toutes les tâches plus prioritaires. Une borne d'utilisation meilleure que celle de Lui et Layland a été fournie dans [60].

Pour une tâche τ_i, définissons

- $r_i = \dfrac{C_i^m}{C_i^{m+1}}$ si $N_i \geq 2$, *m* est la trame de pointe.
- $r_i = 1$ si $N_i = 1$

Pour un système S de *n* tâches, définissons $r = \min_{1 \leq i \leq n} r_i$

Théorème 1.8 [60]

Pour un système S de n tâches multi-trames, S est ordonnançable si le facteur d'utilisation ($U = \sum_{i=1}^{n} \dfrac{\max_{0 \leq j \leq N_i - 1} C_i^j}{T_i}$) est inférieur à $r * n * ((\dfrac{r+1}{r})^{\frac{1}{n}} - 1)$.

Analyse basée sur le temps de réponse pour les tâches à priorités fixes

Dans [62], Baruah et al ont proposé un test suffisant d'ordonnançabilité par analyse approchée du temps de réponse. L'idée consiste à utiliser une fonction qui détermine la demande d'exécution cumulative d'une tâche pour n'importe quelle séquence de k instances successives.

Pour $0 \le k \le N_i - 1$, nous avons :

$$g(\tau_i, k) = \max_{0 \le j \le N_i - 1} \left\{ \sum_{l=j}^{j+N_i-1} C_i^{l \bmod n} \right\}$$

Cette fonction fournit l'interférence maximum de k exécutions successives d'une tâche τ_i. Afin de calculer l'interférence en fonction de la longueur de l'intervalle de temps t, nous devons déterminer le nombre d'instances pouvant être activées dans t.

$$G(\tau_i, t) = g(\tau_i, \left\lceil \frac{t}{T_i} \right\rceil)$$

Appelons la trame de pointe d'une tâche τ_i la trame possédant la plus grande durée d'exécution. Une tâche τ_i respectera toujours son échéance si sa trame de pointe respecte son échéance (théorème 1 dans [62]) car les trames d'une tâche ont la même échéance temporelle et sont à échéance sur requête. La tâche τ_i est alors ordonnançable si le pire temps de réponse de la trame de pointe est inférieur à son échéance temporelle. La borne du temps de réponse est obtenue par la méthode du calcul de temps de réponse de Joseph et Pandya [48].

$$R_i^0 = g(\tau_i, 1)$$

$$R_i^k = g(\tau_i, 1) + \sum_{\tau_i \in hp(\tau_i)} G(\tau_i, R_i^{k-1})$$

Un instant critique pour une tâche analysée τ_i survient lorsque sa trame de pointe s'active simultanément avec une trame de chacune des tâches plus prioritaires [63]. Pour une analyse exacte du pire temps de réponse, on doit considérer tous les instants candidats à produire le pire temps de réponse en combinant toutes les trames des tâches plus prioritaires. La complexité de l'algorithme est exponentielle. Une réduction du nombre d'instants critiques qui doivent être examinés est possible en détectant tout instant qui ne pourra jamais produire le pire temps de réponse de la tâche analysée. Ces instants critiques coïncident avec l'activation des trames qui dominent d'autres trames [63][64]. Dans [64], les auteurs ont étendu l'analyse du pire temps de réponse en intégrant les gigues, puis en considérant le cas où l'échéance temporelle d'une tâche est supérieure à sa période.

Dans le modèle multi-trames, chaque tâche possède une période et une échéance unique pour toutes ses trames. Ces restrictions vont être levées dans le modèle multi-trame généralisé.

1.4.9 Analyse de l'ordonnançabilité des tâches multi-trames généralisées

Le modèle multi-trames généralisées (Generalized multiframe - GMF) [65] est une extension du modèle multi-trames dans les directions suivantes :

– L'échéance d'une trame peut être différente de son temps de garde ; les trames différentes peuvent avoir des échéances différentes,
– Les trames différentes peuvent avoir des temps de gardes différents.

Une tâche GMF τ_i est composée de N_i trames. Chaque trame est caractérisée par un temps d'exécution C_i^j, une échéance D_i^j et un temps de garde T_i^j entre son arrivée et l'arrivée de la trame suivante, pour j tel que $0 \le j \le N_i - 1$.

Avant d'aborder les différentes approches d'analyse des tâches GMF (avec priorité fixe et priorité dynamique), nous citons quelques propriétés qui ont été définies dans la littérature :

– une tâche satisfait la propriété L-MAD (localized monotonic Absolute Deadline) si l'échéance de chaque trame survient avant l'échéance de la trame suivante [65], c'est à dire pour tout j $(0 \le j \le N_i - 1)$ $D_i^j \le T_i^j + D_i^{(j+1) \bmod N}$,
– une tâche satisfait la propriété FS (Frame Separation) si l'échéance de chaque trame survient avant le réveil de la trame suivante [66], c'est à dire pour tout j $(0 \le j \le N_i - 1)$ $D_i^j \le T_i^j$.

Il est évident que toute tâche satisfaisant la propriété FS satisfait la propriété L-MAD.

Analyse du temps de réponse des tâches GMF à priorités fixes

Cette analyse est faite sous l'hypothèse que les tâches GMF satisfont la propriété FS.

Soit $W_{ik}(t)$ la fonction qui détermine la charge totale causée par une tâche τ_i dans un intervalle de temps t lorsqu'elle débute par la k^{eme} trame.

$$W_{ik}(t) = \sum_{h=k}^{k+j-1} C_i^{h \bmod N_i} \text{ où } j = \min\left\{ j : \sum_{h=k}^{k+j-1} T_i^{h \bmod N_i} \ge t \right\}.$$

Afin d'étudier l'ordonnançabilité d'une tâche GMF, il est nécessaire de tester l'ordonnançabilité de toutes ses trames, pour tous les instants critiques candidats. Un instant critique candidat pour une trame d'une tâche se produit lorsque la trame s'active simultanément avec une trame de toute tâche plus prioritaire. Comme pour les transactions l'analyse exacte (condition nécessaire et suffisante) du temps de réponse a une complexité exponentielle car on doit examiner toutes les combinaisons possibles entre les trames. Une analyse avec une complexité pseudo-polynomiale est possible mais elle fournit une condition suffisante d'ordonnançabilité en calculant un temps de réponse approché pour chaque trame. Cette analyse consiste à étudier l'interférence maximum qu'une tâche GMF peut imposer sur le temps de réponse d'une tâche moins prioritaire. La fonction d'interférence (MIF) [66] représente l'interférence maximum de τ_i dans l'intervalle de temps t, en considérant que chaque trame débute l'intervalle t :

$$M_i(t) = \max_{0 \le k \le N_i - 1} W_{ik}(t)$$

La borne du temps de réponse d'une trame j d'une tâche τ_i est donnée par :

$$R_{ij}^{(0)} = C_i^j$$

$$R_{ij}^{(k)} = C_i^j + \sum_{i \in hp(\tau_i)} M_i(R_{ij}^{(k-1)})$$

Cette analyse a été améliorée en remplaçant la fonction d'interférence $W_{ik}(t)$, par une nouvelle $W_{ik}^{'}(t)$ [66], qui détermine l'interférence imposée effectivement par chacune des

tâches plus prioritaires. Cette fonction est équivalente à celle d'interférence effective proposée pour les transactions :

$$W'_{ik}(t) = \sum_{h=k}^{k+j} C_i^{h \bmod N_i} + \min\left(C_i^{(h+j) \bmod N_i}, t - \sum_{h=k}^{k+j-1} T_i^{h \bmod N_i} \right)$$

Analyse de la demande processeur des tâches GMF à priorités dynamiques

Considérons que toutes les tâches GMF satisfont la propriété L-MAD et sont ordonnancées par l'algorithme à priorités dynamiques EDF. Un test pour analyser la demande processeur consiste à vérifier pour tout intervalle de temps que la charge du processeur produite est toujours inférieure ou égale à la longueur de l'intervalle. Ce test est limité par la plus longue période d'activité [31] et plus précisément par les dates d'échéances survenant dans la plus longue période d'activité.

Soit $dbf(\tau_i, t)$ (Demand-Bound Function) la fonction déterminant le temps processeur maximum requis par les instances de la tâche τ_i activées dans l'intervalle [0,t) et dont les échéances sont elles aussi dans cet intervalle. Baruah et al [65] donnent un algorithme « Build list » calculant la demande processeur durant la première période (somme des temps de garde des N_i trames) de chaque tâche GMF du système S. Cet algorithme enregistre la demande processeur sous forme de couples (date d'échéance, demande processeur). La liste des couples sera utilisée facilement pour trouver la demande processeur maximum dans n'importe quel intervalle de temps *t* (algorithme Compute-dbf(t) [65]).

En exploitant la périodicité de la fonction de la demande maximum $dbf(\tau_i, t)$, une transformation de système S à un ensemble de tâches sporadiques (système S') modélisant cette fonction est possible. Le système S est ordonnançable sur une architecture monoprocesseur, si et seulement si, le système S' des tâches sporadiques correspondant est ordonnançable (théorème 2 dans [65]).

1.4.10 Outils de vérification d'ordonnancement temps réel

Plusieurs outils de vérification d'ordonnancement temps réel ont été développés afin d'automatiser les différentes techniques d'analyse présentées dans la littérature. Nous pouvons citer :

- Cheddar [67] : développé à l'université de Brest,
- MAST [68] : développé à l'université de Cantabria,
- Rapid RMA [69]: développé par la société Tri pacific software.

1.5 Conclusion et problématiques

Dans ce premier chapitre nous avons présenté dans un premier temps la structure hétérogène et distribuée d'une radio logicielle, ainsi que certaines recommandations du SCA.

Dans un deuxième temps, nous avons présenté les différentes méthodologies et langages dont dispose le concepteur afin de spécifier son système. Dans le domaine de la radio logicielle, l'utilisation de méthodologies de conception à haut-niveau d'abstraction s'avère aujourd'hui indispensable compte tenu de la complexité grandissante d'un modem radio. En effet le besoin de supporter plusieurs standards radios sur un même poste, entraîne un grand nombre de composants logiciels de la couche physique à s'exécuter simultanément sur un même processeur. Cependant ces composants logiciels qui correspondent aux traitements en bande de base ont une notion duale d'exactitude : exactitude logique et exactitude temporelle. Or les méthodologies présentées dans ce chapitre sont soit très génériques, soit adaptées à une conception de composants de traitements en bande de base pour une architecture matérielle de type FPGA. La question se pose alors de savoir quelle méthodologie de conception adopter pour la vérification temps réel de composants logiciels de traitements du signal s'exécutant simultanément sur un processeur donné ? Comment calculer puis exprimer les contraintes temporelles lors de la modélisation des applications de traitement du signal avec le nouveau profil pour un système temps réel embarqué (MARTE), de manière à ce que ces contraintes puissent être exploitées par un outil d'analyse d'ordonnançabilité temps réel ?

Dans un troisième temps, nous avons présenté dans ce chapitre les différents modèles de tâches ainsi que les différentes techniques d'analyse permettant la validation de l'ordonnançabilité temps réel de ces tâches. Le modèle de tâche multi-trames généralisé se rapproche le plus du modèle de tâche rencontré dans les applications de traitement de signal réalisées à ce jour car il autorise le temps d'exécution, l'échéance temporelle et le temps de garde d'une tâche à varier d'une activation à l'autre en fonction d'un pattern d'activation spécifique. Cependant pour les applications logicielles de traitement du signal dans les postes radio, on ne peut pas considérer un pattern spécifique d'activation car le système est dirigé par des évènements venant d'utilisateurs. Par exemple si une tâche logicielle a deux temps d'exécutions et deux échéances temporelles parce qu'elle traite deux types de trames différentes (représentant deux services différents), on ne peut pas prédire pendant la phase de conception un ordre d'arrivée fixe des trames qui se répétera durant toute la durée de vie du système. Dans ce contexte, où il n'y a pas de motif spécifique d'activation, comment vérifier que l'ensemble de tâches logicielles s'exécutant sur un processeur donné respecteront leurs échéances temporelles ? Puis, se pose aussi la question de savoir quelle politique d'ordonnancement et quel outil de vérification temps réel seront adaptés pour les composants logiciels mettant en œuvre les traitements en bande de base d'un schéma de modulation de type OFDM (Orthogonal Frequency Division Multiplexing) par exemple ?
Nous répondrons à ces questions dans les chapitres suivants.

Chapitre 2

Contributions :

Ordonnancement temps réel

Résumé : *Le modèle de tâche multi-trame généralisé (GMF) a été proposé afin de modéliser une tâche dont le temps d'exécution, l'échéance et le temps de garde changent en fonction d'un motif spécifique d'activation. Dans ce chapitre nous relâchons l'hypothèse consistant à prendre en compte un motif spécifique d'activation, ce qui nous conduit au modèle de tâche GMF non cyclique. Dans ce contexte, les techniques d'analyse d'ordonnancement pour le modèle de tâche GMF ne peuvent pas être appliquées. Ce chapitre présente une nouvelle formule pour le calcul du temps de réponse d'une tâche GMF non cyclique s'exécutant sur un monoprocesseur suivant la politique d'ordonnancement EDF. Aussi, un nouveau test d'ordonnançabilité basé sur la densité est donné. En dernière partie, nous présentons une nouvelle approche efficace, permettant la détermination exacte de la faisabilité temps réel d'un ensemble de tâche GMF non cyclique, par simulation.*

2.1 Introduction

La plupart des études sur les algorithmes de vérification d'ordonnancement temps réel ont relâché les hypothèses du modèle de tâche classique de Lui et Layland [35]. Mok et Chen [60] sont les premiers à introduire le concept de multi-trames (multiframe). Une tâche multi-trames est une tâche dont le temps d'exécution change périodiquement en fonction d'un motif spécifique. Par exemple une tâche avec un temps d'exécution de 5 ms et de 8 ms, est considérée comme ayant deux trames et ces deux temps d'exécution changent à chaque activation en fonction d'un motif spécifique. Le modèle de tâche multi-trames généralisé (GMF) est une extension du modèle multi-trames dans les trois directions suivantes : (1) L'échéance d'une trame peut être différente de sa période, (la période est désormais considérée comme un temps de garde) (2) les trames différentes peuvent avoir des échéances différentes, (3) les trames différentes peuvent avoir des temps de garde différents. Le temps de garde est défini comme étant le temps minimum qui doit s'écouler entre deux activations successives d'une même tâche. Nous introduisons dans cette thèse un nouveau modèle de tâches auquel se rapporte notre contexte de radio logicielle : le modèle de tâche GMF non cyclique. La principale

différence entre le modèle de tâche GMF et celui non cyclique vient du fait qu'il n'y a pas de motif spécifique d'activation de la tâche. En d'autres mots, dans le modèle GMF non cyclique il est possible d'activer n'importe quelle trame à la fin du temps de garde de la trame précédente. Le terme « non cyclique » est utilisé pour mettre en évidence le fait que le motif d'activation qui crée le cycle n'est pas spécifié.

La Figure 2.1 représente une séquence d'exécution des deux modèles de tâches, pour celui d'en haut, il s'agit d'une tâche GMF τ_i ((1,4,4),(2,5,6),(3,6,7)) constitué de 3 trames. Le temps d'exécution de la 2ème trame est 2, son échéance relative est 5, et son temps de garde est 6. Autrement dit, lorsque la 2ème trame est activée, la trame suivante, qui est nécessairement la trame avec pour temps d'exécution 3, est activée après au moins 6 unités de temps. Nous choisissons de représenter le motif d'activation par un vecteur composé de la suite des temps d'exécution des trames. De manière évidente nous observons que la tâche GMF τ_i, du haut de la Figure 2.1, a un motif égal à [1 ;2 ;3]. Et ce motif se répète dans un cycle de 17 unités de temps. En revanche, le scénario d'en bas de la Figure 2.1 représente une séquence d'exécution possible de la même tâche sans considérer le motif d'activation (tâche GMF non cyclique). La différence est que la suite des trames ne suit pas un motif spécifique. En fait, chacune des trois trames peut être activée à la fin du temps de garde de la trame précédente.

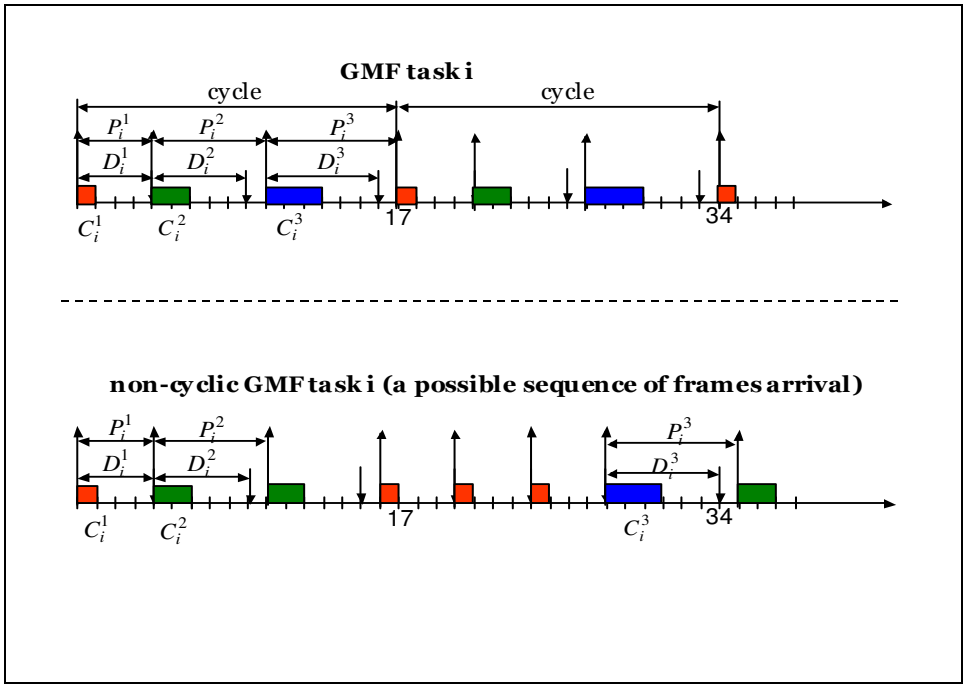

Figure 2.1 – Comparaison entre une tâche GMF et une tâche GMF non cyclique

Baruah *et al* [65] ont défini la propriété L-MAD (définie formellement dans le chapitre précédent). Ils ont proposé un algorithme basé sur la densité déterminant l'ordonnançabilité d'un ensemble de tâches GMF ordonnancées avec la politique EDF. Cet algorithme a une complexité polynomiale.

La méthodologie consiste à déterminer la demande processeur (dbf – Demand Bound Function) pour chaque tâche et vérifier qu'il n'existe pas un intervalle de temps *t* tel que $\sum_i dbf(\tau_i, t) \succ t$.
Puisqu'il n'existe pas de motif d'activation qui se répète, l'algorithme proposé dans [65] pour calculer la demande processeur ne peut pas être appliqué.

Dans l'optique de prendre en compte la relation de décalage entre les tâches, les auteurs dans [71], ont proposé le modèle de tâches avec décalage (ou transactions). L'analyse par le calcul du temps de réponse pour les transactions ordonnancées avec EDF a été proposée par Palencia et Harbour dans [72], suivi par Pellizzoni et Lipari dans [73]. Traoré *et al* [74] ont mentionné dans leur article que le modèle multi-trame est un cas particulier du modèle avec décalage, ainsi ils considèrent que l'analyse pour les tâches avec décalages peut s'appliquer au modèle multi-trames. Plus récemment Rahni a montré dans [51], comment transformer une tâche GMF en transaction. Suivant cette approche, la transformation d'une tâche GMF non cyclique en une transaction, donne une transaction avec des périodes différentes. Ainsi la période de la transaction change d'un événement à l'autre sans que nous ne sachions dans quel ordre, vu qu'il n'y a pas de motif qui se répète. Il est donc clair que l'analyse par le calcul du temps de réponse proposé pour les transactions ordonnancées avec EDF ne peut pas être appliquée au modèle de tâche GMF non cyclique.

Un exemple de modèle de tâche GMF non cyclique, régulièrement rencontré dans les modems radio industriels, consiste en la version multi-utilisateur d'un schéma de modulation OFDM (Orthogonal Frequency Division Multiplexing). Le modem radio peut servir plusieurs utilisateurs à travers la gestion de trames de tailles différentes. Les trames vidéo nécessitent habituellement plus de temps de traitement que les trames audio, et les trames audio ont une latence plus faible que celle des trames vidéo. Ceci implique pour les tâches de l'application, un temps d'exécution, une échéance temporelle et un temps de garde qui varient en fonction du type de trame en cours de traitement. Evidemment on ne peut pas présupposer un motif d'arrivée des trames car le système est dirigé par des événements venant d'utilisateurs (l'utilisateur A effectue un appel audio pendant que l'utilisateur B regarde une vidéo en diffusion).
La Figure 2.2 représente un modem radio typique. Il est composé de 3 processeurs, et nous nous intéressons à l'ordonnancement temps réel sur un processeur d'intérêt qui exécute une sous-partie des traitements du modem. L'application contient une chaîne de transmission, une chaîne de réception et une chaîne pour contrôler la radio. Chaque chaîne est constituée d'algorithmes de traitement du signal. Les flèches entre les fonctions représentent le flux de données. Typiquement, G est un algorithme de démodulation pour lequel le temps d'exécution est proportionnel à la taille de la trame, alors que I est un algorithme de décodage flexible pour lequel les différents temps d'exécution sont dus aux différents paramètres pour les trames audio et vidéo.
Sans perte de généralité, nous choisissons de « mapper » chaque algorithme sur une tâche, ce qui nous conduit au modèle GMF non cyclique.

Ce chapitre fournit les contributions suivantes :

– La formule pour le calcul du temps de réponse d'une tâche GMF non cyclique ordonnancée avec EDF. Puis un test d'ordonnançabilité suffisant basé sur la densité avec une complexité polynomiale.
– Une approche efficace, pour la détermination exacte de la faisabilité temps réel, en utilisant la simulation.

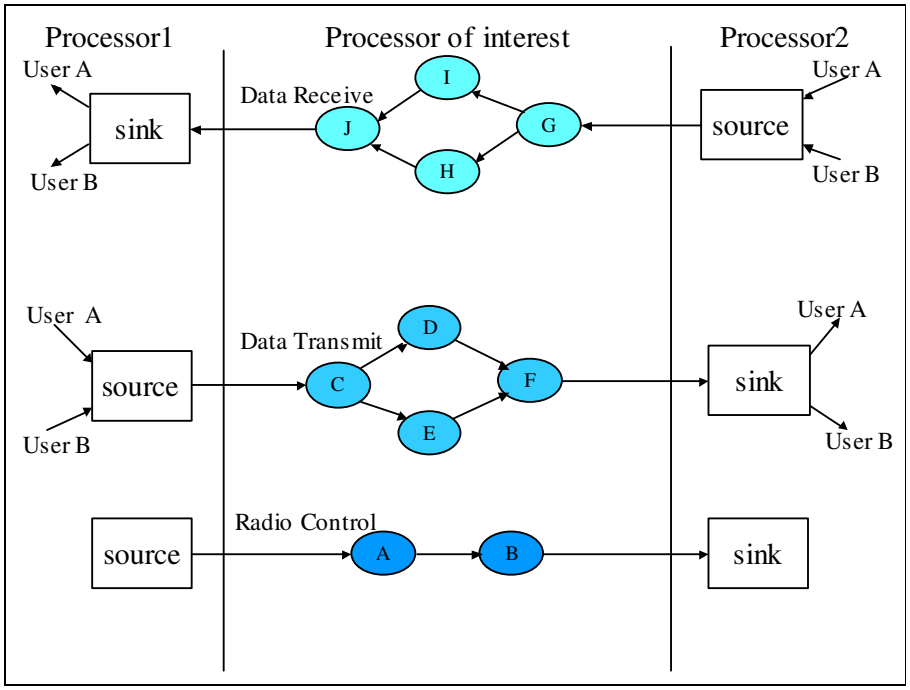

Figure 2.2 – Un modem radio typique

Le reste de ce chapitre est organisé comme suit. La section suivante rappelle nos notations. La section 2.3 explique comment nous trouvons la formule pour le temps de réponse et présente le test de faisabilité suffisant. Une approche efficace pour la détermination exacte de la faisabilité temps réel est présentée à la section 2.4. Les conclusions sont fournies en section 2.5.

2.2 Rappel sur les notations

L'analyse dans ce chapitre considère un système composé de *m* tâches indépendantes GMF non cycliques avec une durée d'activité que nous appelons « non-stop-runtime ». Le « non-stop-runtime » est l'intervalle de temps pendant lequel le système reçoit une rafale de trames. Après cet intervalle de temps, s'écoule un très court intervalle de temps où le système ne reçoit aucune trame à émettre ou à recevoir, puis revient de nouveau une rafale de trame et ainsi de suite. Typiquement, dans un modem radio, la réception d'une rafale de trames survient lorsqu'il faut transmettre des données utiles sur la voie radio. L'intervalle de temps durant lequel il n'y a pas de trames correspond au moment de la synchronisation d'horloge entre émetteur et récepteur. Pendant cet intervalle de temps,

on ne reçoit pas de données utiles. Une tâche GMF non cyclique constituée de N_i trames est caractérisée par une séquence de triplets ((C_i^0, D_i^0, P_i^0), ... ($C_i^{N_i-1}, D_i^{N_i-1}, P_i^{N_i-1}$)) où C_i^j , D_i^j , et P_i^j représente le pire temps d'exécution de la j-ième trame de la tâche τ_i , l'échéance relative de la trame, et le temps de garde entre la j-ième trame et la trame suivante, respectivement. Chaque tâche effectue une première activation à l'instant O_i .
Nous considérons les tâches GMF non cycliques qui satisfont la propriété « Frame Separation », ce qui signifie que l'échéance absolue d'une trame n'est pas plus tardive que la date d'arrivée de la trame suivante, où que $D_i^j \le P_i^j$ est vraie quelque soit j $(0 \le j \le N_i - 1)$ [66].

2.3 Condition suffisante de faisabilité

Dans l'optique de déterminer un test de faisabilité suffisant, nous considérons d'une part une activation initiale simultanée de toutes les tâches à un instant t ($\forall i \in [0, m-1]$, $O_i = t$) et d'autre part que leurs trames suivantes arrivent le plus tôt possible ($\forall i \in [0, m-1], \forall j \in [0, N_i - 1]$, $D_i^j = P_i^j$). Ceci crée une demande de calcul maximale pour le processeur, produisant ainsi le pire scénario pour les tâches afin qu'elles respectent leurs échéances temporelles. Nous nous attaquons d'abord au problème qui consiste à trouver une formule pour le calcul du temps de réponse des tâches GMF non cycliques.

2.3.1 Calcul du temps de réponse

Pour l'activation d'une tâche GMF non cyclique τ_i à un instant *t*, due à l'arrivée de sa j-ième trame, le temps de réponse de la tâche est composé du temps d'exécution de la trame correspondante, plus l'interférence dont souffre la tâche τ_i , due aux activations d'autres tâches du système avec des dates d'activations et échéances absolues comprises entre [*t*,*t*+ D_i^j]. Nous définissons l'interférence comme étant l'impact que subit une tâche τ_i en raison des activations d'autres tâches du système avec des échéances plus proches que celle de la tâche τ_i . En effet comme nous considérons un ordonnanceur EDF, les tâches activées avec une échéance plus proche se verront toujours attribuer le processeur.

La Figure 2.3 présente un exemple d'interférence subie par une tâche τ_1 (avec une échéance temporelle à D_1) après son activation simultanée à l'instant 0 avec les tâches τ_2 et τ_3 . Cette interférence est due à l'exécution des tâches τ_2 et τ_3 qui ont leurs échéances temporelles (D_2 et D_3) avant D_1 .

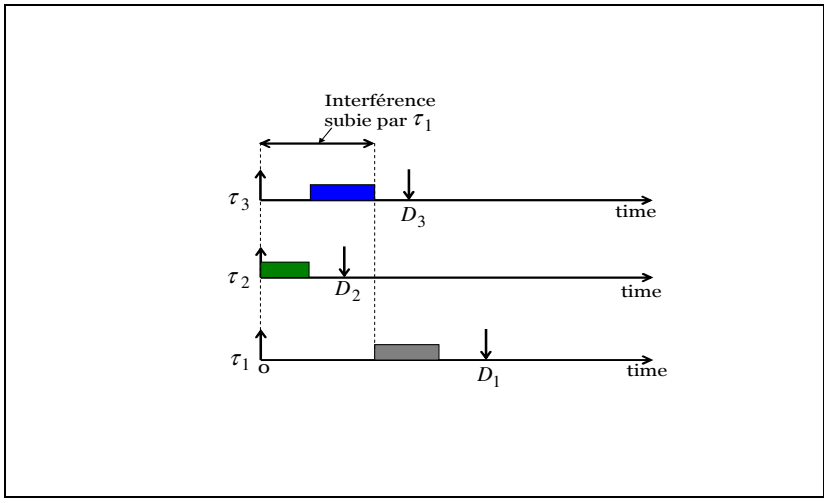

Figure 2.3 – Exemple d'interférence subie par une tâche

Soit $R_{i,t+D_i^j}$, le temps de réponse de τ_i, nous avons $R_{i,t+D_i^j} = C_i^j + I_{i,t+D_i^j}$ où $I_{i,t+D_i^j}$ représente l'interférence dont souffre la tâche τ_i. Nous présentons à présent l'analyse pour le calcul de $I_{i,t+D_i^j}$.

Afin d'illustrer comment nous trouvons une formule pour le calcul de l'interférence, nous considérons l'exemple du système du Tableau 2.1.

task	O_i	C_i^j	$D_i^j = P_i^j$
τ_1	0	2	4
		3	17
		1	5
τ_2	0	3	16

Tableau 2.1 – Système d'exemple 1

Le système est composé de deux tâches. La tâche τ_1 a 3 trames alors que la tâche τ_2 a juste une trame. Nous sommes intéressés par le respect de l'échéance temporelle de la tâche τ_2, en considérant une activation initiale simultanée avec la tâche τ_1. Pour calculer l'interférence due aux activations de τ_1, nous éliminons d'abord la seconde trame de la tâche τ_1 (la trame pour laquelle l'échéance temporelle est à 17), ceci parce que l'arrivée de cette trame ne peut pas retarder l'exécution de la tâche τ_2, car cette échéance est plus grande que celle de la tâche τ_2. La Figure 2.4 représente sous forme d'un arbre les différentes séquences d'activations de τ_1 qui peuvent retarder l'exécution de τ_2 jusqu'à l'instant 16. Chaque chemin dans l'arbre représente une séquence d'exécution qui peut interférer, car les dates d'arrivées et les échéances temporelles sont comprises dans l'intervalle [0 ; 16]. Par exemple, le chemin 4 correspond à la séquence [2 ;1 ;1].

Le temps de réponse est calculé pour chaque séquence possible d'activation de la tâche τ_1. Soit L_n, représentant une séquence d'activation de la tâche τ_1.
Pour $L_1 = [2,2,2,2]$, chemin 1, nous avons

$R_{2,16} = 3+2*4 = 11 \le 16$, l'échéance temporelle est respectée

Pour $L_2 = [2,2,1]$, chemin 2, nous avons

$R_{2,16} = 3+2*2+1 = 8 \le 16$, l'échéance temporelle est respectée

Pour $L_3 = [2,1,2]$, chemin 3, nous avons

$R_{2,16} = 3+2*2+1 = 8 \le 16$, l'échéance temporelle est respectée

Pour $L_4 = [2,1,1]$, chemin 4, nous avons

$R_{2,16} = 3+2+2*1 = 7 \le 16$, l'échéance temporelle est respectée

Pour $L_5 = [1,1,1]$, chemin 5, nous avons

$R_{2,16} = 3+3*1 = 6 \le 16$, l'échéance temporelle est respectée

Pour $L_6 = [1,1,2]$, chemin 6, nous avons

$R_{2,16} = 3+2*1+2 = 7 \le 16$, l'échéance temporelle est respectée

Pour $L_7 = [1,2,1]$, chemin 7, nous avons

$R_{2,16} = 3+2*1+2 = 7 \le 16$, l'échéance temporelle est respectée

Pour $L_8 = [1,2,2]$, chemin 8, nous avons

$R_{2,16} = 3+1+2*2 = 8 \le 16$, l'échéance temporelle est respectée

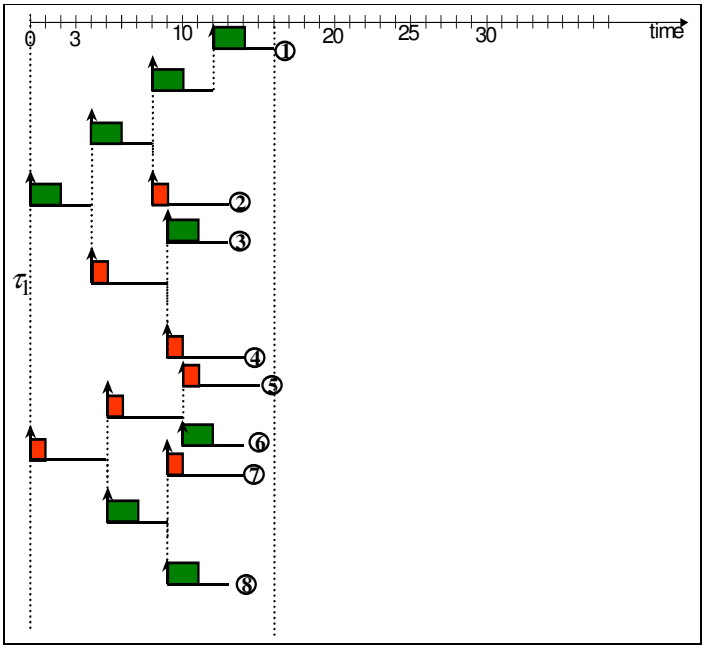

Figure 2.4 – Interférence générée par τ_1 dans l'exemple 1

Soit S_1 représentant l'ensemble des différentes séquences d'activations de τ_1 qui peuvent interférer sur l'exécution de τ_2, nous avons :

$S_1 = \{[2;2;2;2],[2;2;1],[2;1;2],[2;1;1],[1;1;1],[1;1;2],[1;2;1],[1;2;2]\}$.

De manière formelle, pour une tâche τ_u, l'ensemble des séquences d'activations qui peuvent interférer avec l'exécution d'une autre tâche τ_i est donné par :

$S_u = \left\{[C_u^{l_1};...;C_u^{l_1}],[C_u^{l_2};...;C_u^{l_2}],...,[C_u^{l_n};...;C_u^{l_n}]\right\}_{l_1 \in L_1, l_2 \in L_2, ..., l_n \in L_n}$ tel que pout tout $l_n \in L_n$ nous avons $t + \sum_{l_n} D_u^{l_n} \le t + D_i^j$

En général l'interférence est calculée comme suit :

Soit $\Delta = \{\tau_i\}_{i\in[0;m-1]}$ un ensemble de tâches. Considérons une tâche $\tau_i \in \Delta$ activée à un instant *t,* pour calculer le temps de réponse de τ_i, nous éliminons d'abord pour chaque tâche dans $\Delta - \{\tau_i\}$, les trames avec des échéances temporelles supérieures à D_i^j. En effet l'arrivée de ces trames ne peut pas retarder l'exécution de τ_i. Evidemment si toutes les trames d'une tâche sont éliminées, alors la tâche est éliminée. Ainsi pour les tâches et leurs trames respectives restantes nous avons $\forall u(u \neq i), \forall k \in [0, N_i - 1], D_u^k \leq D_i^j$. Nous appelons ce nouvel ensemble Δ_i^* qui est défini de manière formelle par :

$\Delta_i^* = \{\tau_u\}_{u\neq i / \forall k, D_u^k \leq D_i^j}$ (Le caractère « / » correspond à l'expression « tel que »)

L'interférence dont souffre la tâche τ_i est donc donnée par :

$$I_{i,t+D_i^j} = \sum_{\tau_u \in \Delta_i^*} \left(\sum_{l_n / t+\sum_{l_n} D_u^{l_n} \leq t+D_i^j} C_u^{l_n} \right)$$

Dans la formule de l'interférence, nous avons l'expression

$$\sum_{l_n / t+\sum_{l_n} D_u^{l_n} \leq t+D_i^j} C_u^{l_n}$$

qui correspond à la somme des temps d'exécution des séquences correspondant aux activations dont les échéances temporelles sont avant D_i^j.

Ainsi, pour chacune des tâches τ_u, nous combinons une de ses séquences avec une des séquences des autres tâches.

Le temps de réponse est alors égal à :

$$R_{i,t+D_i^j} = C_i^j + \sum_{\tau_u \in \Delta_i^*} \left(\sum_{l_n / t+\sum_{l_n} D_u^{l_n} \leq t+D_i^j} C_u^{l_n} \right) \quad (1)$$

Nous proposons dans la section suivante, un test d'ordonnançabilité basé sur la densité pour un ensemble de tâches GMF non cycliques.

2.3.2 Test basé sur la densité

La situation la plus critique

Le pire scénario pour une tâche τ_i lors la réception de sa j-ième trame (.i.e. le scénario qui lui génère un temps de réponse maximal pour le traitement de cette trame) est atteint lorsque l'interférence qu'elle subit pour le traitement de cette trame est maximale (de part la définition du temps de réponse). Cette interférence maximale est obtenue lorsque toutes les tâches du système sont activées simultanément, au même moment où la tâche τ_i reçoit sa j-ième trame. A titre de preuve, prenons l'exemple de la Figure 2.5, où est représentée deux tâches (τ_i et τ_{i+1}).

Nous nous intéressons à l'interférence subie par la tâche τ_i due aux activations de la tâche τ_{i+1}. Considérons une activation de la tâche τ_i à l'instant t1 et de la tâche τ_{i+1} à l'instant t2. Durant l'intervalle $[t_1, t_1 + D_i^j]$, la préemption de la tâche τ_i par la tâche τ_{i+1} va causer un retard sur la fin de l'exécution de τ_i, à moins que la tâche τ_i ait fini son exécution à l'instant t2. De plus, si l'activation de la tâche τ_{i+1} a lieu à une date supérieure à t2, l'interférence subie par la tâche τ_i, sera soit inchangée, soit diminuée. Par

conséquent, pour maximiser l'interférence que subie la tâche τ_i, il faut ramener l'activation de la tâche τ_{i+1} à t1 (activation simultanée des deux tâches).

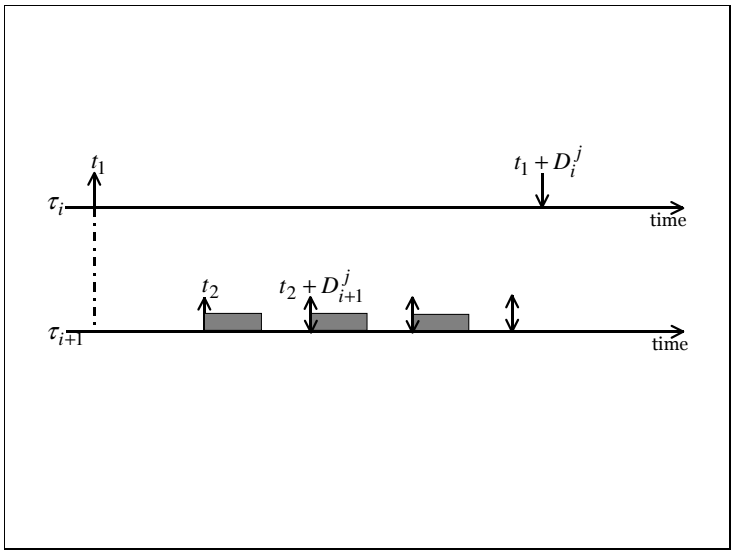

Figure 2.5 – Interférence maximale subit par une tâche

Nous proposons le théorème suivant :

Théorème 2.1

Etant donné un ensemble de m tâches GMF non cycliques traitant N_i trames, si

$$\sum_{i=0}^{m-1} \max_{0 \le j \le N_i - 1} \left(\frac{C_i^j}{D_i^j}\right) \le 1 \qquad (2)$$

alors l'ensemble des tâches est ordonnançable avec EDF sur une architecture monoprocesseur.

Preuve :

En considérant une activation simultanée de toutes les tâches, la preuve consiste à montrer que, lorsque la condition (2) est satisfaite, toutes les tâches respectent leurs échéances temporelles pour leur 1ère activation. En effet les activations suivantes seront : soient simultanées et nous retombons sur la même situation que celle de la 1ère activation, soient non simultanées et nous avons une situation moins critique car l'interférence dont souffre chaque tâche n'est pas maximale.

Pour des besoins de clarté, considérons la x_i -ième trame pour chaque tâche τ_i, où $\frac{C_i^{x_i}}{D_i^{x_i}} = \max_{0 \le j \le N_i - 1} \left(\frac{C_i^j}{D_i^j}\right)$, la condition (2) devient $\sum_{i=0}^{m-1} \frac{C_i^{x_i}}{D_i^{x_i}} \le 1$.

Sans perte de généralité, considérons que la 1ère activation de la tâche τ_i est initiée par sa j-ième trame. Pour que la tâche τ_i respecte son échéance temporelle, il suffit que le temps de réponse de sa 1ère activation soit inférieur à D_i^j, i.e. :

$$C_i^j + \sum_{\tau_u \in \Delta^*} \left(\sum_{l_n / \sum_{l_n} D_u^{l_n} \le D_i^j} C_u^{l_n} \right) \le D_i^j \qquad (3)$$

Nous montrons à présent que l'équation (3) est vérifiée lorsque la condition (2) est vraie. Nous avons

$\forall l_n$

$$\frac{C_u^{l_n}}{D_u^{l_n}} \le \frac{C_u^{x_u}}{D_u^{x_u}}$$

$$\Rightarrow C_u^{l_n} \le D_u^{l_n} * \frac{C_u^{x_u}}{D_u^{x_u}}$$

$$\Rightarrow \sum_{l_n / \sum_{l_n} D_u^{l_n} \le D_i^j} C_u^{l_n} \le \sum_{l_n / \sum_{l_n} D_u^{l_n} \le D_i^j} D_u^{l_n} * \frac{C_u^{x_u}}{D_u^{x_u}}$$

$$\Rightarrow \sum_{\tau_u \in \Delta_i^*} \left(\sum_{l_n / \sum_{l_n} D_u^{l_n} \le D_i^j} C_u^{l_n} \right) \le \sum_{\tau_u \in \Delta_i^*} \left(\sum_{l_n / \sum_{l_n} D_u^{l_n} \le D_i^j} D_u^{l_n} * \frac{C_u^{x_u}}{D_u^{x_u}} \right)$$

En utilisant cette inégalité nous avons finalement

$$C_i^j + \sum_{\tau_u \in \Delta_i^*} \sum_{l_n / \sum_{l_n} D_u^{l_n} \le D_i^j} C_u^{l_n}$$

$$\le C_i^j + \sum_{\tau_u \in \Delta_i^*} \left(\sum_{l_n / \sum_{l_n} D_u^{l_n} \le D_i^j} D_u^{l_n} * \frac{C_u^{x_u}}{D_u^{x_u}} \right)$$

$$\le C_i^j + \sum_{\tau_u \in \Delta_i^*} \frac{C_u^{x_u}}{D_u^{x_u}} \left(\sum_{l_n / \sum_{l_n} D_u^{l_n} \le D_i^j} D_u^{l_n} \right)$$

$$\le C_i^j + \sum_{\tau_u \in \Delta_i^*} \frac{C_u^{x_u}}{D_u^{x_u}} * D_i^j$$

$$\le D_i^j \left(\frac{C_i^j}{D_i^j} + \sum_{\tau_u \in \Delta_i^*} \frac{C_u^{x_u}}{D_u^{x_u}} \right)$$

$$\le D_i^j$$

car $\frac{C_i^j}{D_i^j} + \sum_{\tau_u \in \Delta_i^*} \frac{C_u^{x_u}}{D_u^{x_u}} \le \frac{C_i^{x_i}}{D_i^{x_i}} + \sum_{\tau_u \in \Delta_i^*} \frac{C_u^{x_u}}{D_u^{x_u}} \le 1$ (d'après la condition (2)).

Cette méthode peut être appliquée à chaque tâche τ_i, ce qui prouve le Théorème 2.1.

L'analyse par le calcul du temps de réponse et le test de faisabilité présentés précédemment sont tous les deux suffisants et pas forcément nécessaires. Lorsque les tâches peuvent avoir des dates de premières activations arbitraires, une activation simultanée de toutes les tâches peut ne jamais se produire durant toute la durée de vie du système. De plus on n'a pas nécessairement $D_i^j = P_i^j$ (les trames suivantes arrivent le plus tôt possible). Néanmoins l'avantage du test de faisabilité suffisant est sa complexité polynomiale. Ainsi dans les modems radio reconfigurables soumis aux contraintes de l'embarqué (temps de réponse borné), ce test peut être fait en ligne. La section suivante présente une approche efficace pour une analyse exacte.

2.4 Analyse exacte de la faisabilité temps réel

Pour illustrer le problème de l'analyse exacte, un autre système d'exemple avec deux tâches (τ_1 et τ_2) est représenté dans le Tableau 2.2. La tâche τ_1 est une tâche GMF non cyclique avec 2 trames et une date de 1ère activation à l'instant 3. La tâche τ_2 a juste une trame avec une date de 1ère activation à l'instant 0. Le système a un non-stop-runtime égale à L.

tâche	O_i	C_i^j	D_i^j	P_i^j
τ_1	3	2	3	6
		1	4	4
τ_2	0	2	4	8

Tableau 2.2 – Système d'exemple 2

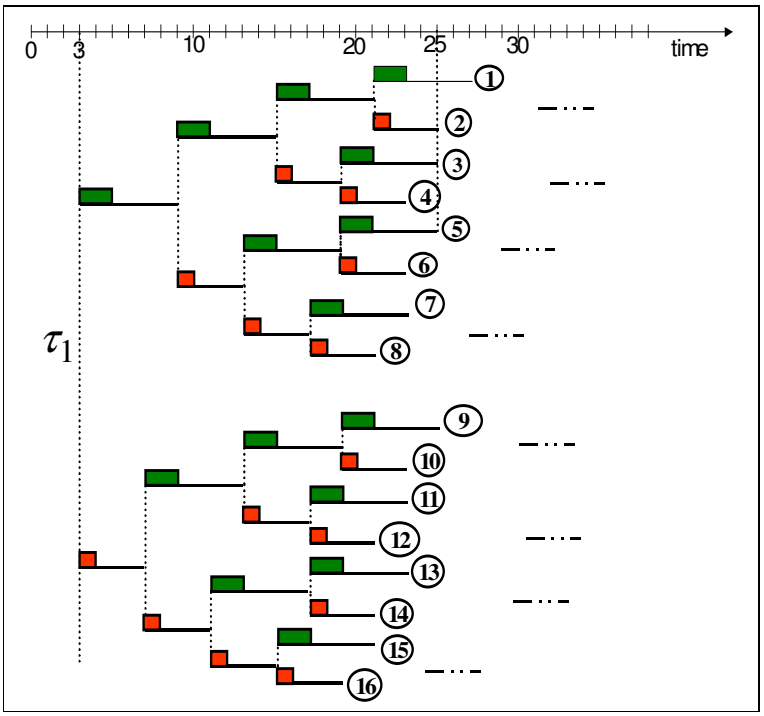

Figure 2.6 – Les différentes possibilités d'exécution de la tâche τ_1 de l'exemple 2

Les différentes possibilités d'exécution de τ_1 sont représentées dans la Figure 2.6 sous la forme d'un arbre. Chaque chemin dans l'arbre représente une séquence d'exécution possible. Par exemple le chemin 5, a la séquence [2 ;1 ;2 ;2] à l'instant 25.

Pour la détermination de la faisabilité par simulation, nous proposons de mettre en œuvre un simulateur EDF avec une queue contenant les tâches activées, prêtes à s'exécuter et

une horloge représentant la progression du temps. Lorsqu'une tâche est activée, elle est mise dans la queue. Précisément un quadruplet (X_i^q, Y_i^q, Z_i^q , #i) est ajouté dans la queue q, où X_i^q est le temps d'exécution de la tâche, Y_i^q son échéance temporelle relative, Z_i^q la date de la prochaine activation de la tâche, et "#i" est l'identifiant de la tâche. La queue est triée en fonction des échéances temporelles croissantes. Lorsque l'horloge avance, le temps d'exécution de la tâche en 1ère position dans la queue est décrémenté. L'échéance temporelle relative de chacune des tâches de la queue est aussi décrémentée. Ainsi, lorsque le temps d'exécution d'une tâche devient plus grand que son échéance temporelle relative ceci revient à dire que cette échéance temporelle ne sera pas respectée. Lorsque le temps d'exécution d'une tâche dans la queue devient égale à zéro, ces paramètres X_i^q et Y_i^q ne sont plus modifiées jusqu'à la prochaine activation de la tâche.
Soit $Pa_i^{k_i}$ la k_i-ième séquence d'exécution d'une tâche τ_i durant un non-stop-runtime L. La propriété formelle à vérifier est la suivante :
« Existe t-il un vecteur [$Pa_0^{k_0}, Pa_1^{k_1}, ..., Pa_{m-1}^{k_{m-1}}$] qui peut être exécuté sur le simulateur EDF sans atteindre la situation où une échéance temporelle n'est pas respectée»

2.4.1 Approche naïve

Dans l'approche naïve, nous ferions d'abord une liste de tous les vecteurs possibles [$Pa_0^{k_0}, Pa_1^{k_1}, ..., Pa_{m-1}^{k_{m-1}}$], $\forall k_0, k_1, ..., k_{m-1}$.

Puis de cette liste, nous vérifions que chaque vecteur ne conduit pas vers une situation où une échéance temporelle n'est pas respectée. Cette approche nécessite la prise en compte de toutes les combinaisons d'exécutions possibles de toutes les tâches et aboutira rapidement à une explosion combinatoire. La complexité s'aggrave lorsque le non-stop-runtime est grand et qu'il y'a beaucoup de tâches.
La section suivante présente une approche efficace, qui réduit le nombre de combinaisons nécessaires.

2.4.2 Approche efficace

Dans notre approche, nous construisons chaque vecteur de la gauche vers la droite, ainsi à chaque instant, nous simulons l'exécution avec une queue, et testons s'il existe une queue produite par un autre vecteur qui est identique, dans ce cas nous remplaçons immédiatement les deux queues par une seule.

Définition 2.1

Considérons Q_1 et Q_2, deux queues avec lesquelles nous avons construit un ordonnancement partiel suivant la politique EDF à un instant t1. Si pour toute tâche τ_i, nous avons ($X_i^1 = X_i^2$ et $Y_i^1 = Y_i^2$ et $Z_i^1 = Z_i^2$) ou ($X_i^1 = X_i^2 = 0$ et $Z_i^1 = Z_i^2$), alors les deux queues sont identiques (du point de vue de l'ordonnancement temps réel). En d'autres termes, à partir de t1, elles construiront le même ordonnancement.

Successivement, pendant la construction des différentes séquences d'exécutions possibles des tâches, nous appliquons cette technique aux queues et remplaçons les

queues identiques par une seule, économisant ainsi l'effort de construction des vecteurs et le temps de simulation des queues identiques.

Ainsi, lors de l'activation initiale de la 1ère tâche, le nombre de queues correspondant aux séquences possibles d'exécution sont créées. Lorsqu'une nouvelle tâche est activée, une routine combine les séquences possibles d'exécution de la nouvelle tâche avec celles précédemment activées. Ceci duplique le nombre de queues. A chaque instant, chaque queue construit un ordonnancement tel que décrit plus haut suivant la politique EDF et les queues identiques sont remplacées par une seule.

Afin d'illustrer comment notre approche marche, nous revenons au système pris en exemple dans le Tableau 2.2.

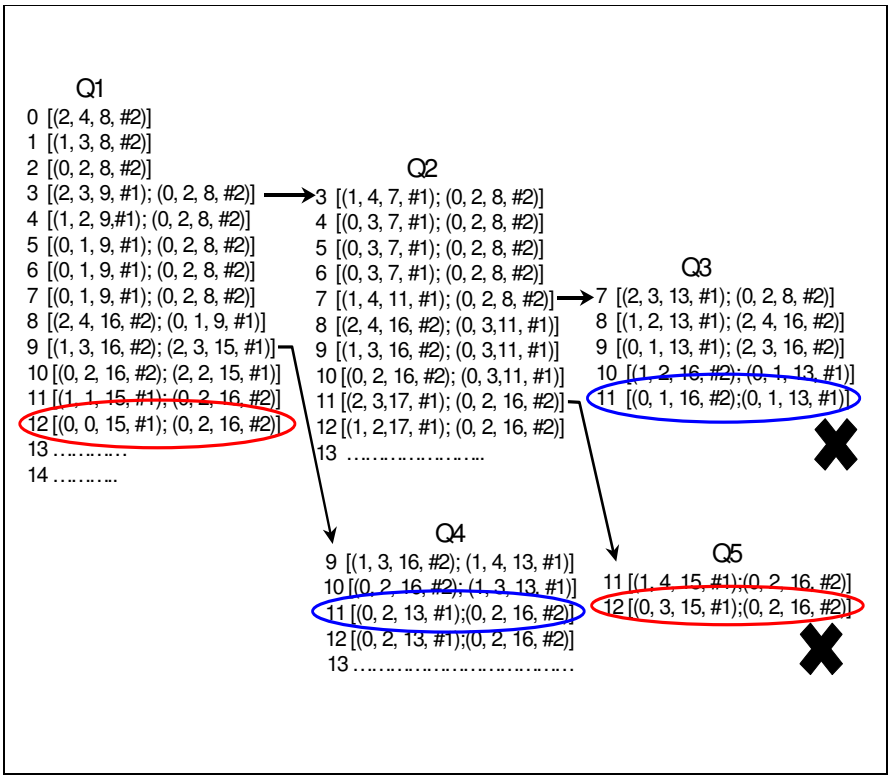

Figure 2.7 – Détermination de la faisabilité dans l'exemple 2

Comme montré dans la Figure 2.7 (la figure est lue du haut vers le bas), à l'instant 0, τ_2 est ajouté dans la 1ère queue Q1. La queue simule la politique d'ordonnancement EDF. Pour simuler l'exécution de la tâche nous diminuons son temps d'exécution d'une unité de temps entre l'instant 0 et l'instant 1. En même temps nous diminuons son échéance temporelle correspondante pour simuler l'échéance qui se rapproche. A l'activation initiale de la tâche τ_1 (instant 3), le quadruplet (2, 3, 9, #1) est ajouté à Q1. Q1 va donc simuler le cas où τ_1 (à l'instant 3) reçoit sa 1ère trame. Simultanément, une nouvelle queue Q2 est créée pour le second cas possible de τ_1. Cette nouvelle queue contient le quadruplet qui se trouvait déjà dans Q1 (0, 2, 8, #2), plus le quadruplet (1, 4, 7, #1). Q2 simule le cas où à l'instant 3, τ_1 reçoit sa deuxième trame.

Puis les deux queues (Q1 et Q2) construisent un ordonnancement suivant la politique EDF. A l'instant 7, qui correspond au moment de la prochaine activation de τ_1 dans Q2, Q3 est créée pour simuler le cas où à cet instant il reçoit sa 1ère trame, et ainsi de suite.
En continuant cette procédure, nous observons qu'à l'instant 11, Q3 et Q4 sont identiques car nous avons $X_1^3 = X_1^4 = 0$ avec $Z_1^3 = Z_1^4 = 13$ et $X_2^3 = X_2^4 = 0$ avec $Z_2^3 = Z_2^4 = 16$. A partir de cet instant, il n'est plus nécessaire de continuer la procédure avec les deux queues, elles construiront le même ordonnancement, nous supprimons donc Q3. A l'instant 12, Q1 et Q5 sont aussi identiques, nous supprimons Q5.
Progressivement pendant la construction des différentes possibilités d'exécutions des tâches, les queues identiques sont remplacées par une seule. Dans la Figure 2.8, nous présentons le pseudo code de l'algorithme pour cette approche.

Feasibility Determination Algorithm

```
time = 0;
while (time < NONSTOPRUNTIME)

  if(all tasks have not been released at least once)
    for τ_i in remaining tasks to initiate their first release
      if ( O_i == time)
        if (the list is empty)
          for each frame of τ_i
            q = create new queue;
            addtuple(q, τ_i );
            addqueuetolist(list, q);
          endfor
        else
          adddifferentpossibilitieswithprevioustaskreleased(list, τ_i )
        endif
      endif
    endfor
  endif

  time++;
  for each queue in list
    executeEDFstrategy(q);
    if(deadlinemissed(q)) return unschedulable;
      for each tuple in q
        if(exist i: Z_i^q ==time) adddifferentcombination(list, τ_i );
      endfor
   endfor
  eliminateidenticalqueue(list);
endwhile

for each queue in list
  executeEDFstrategyforremainingworkload(q);
  if(deadlinemissed(q)) return unschedulable;
endfor

return schedulable;
```

Figure 2.8 – Algorithme de détermination de la faisabilité temps réel exacte

L'algorithme se termine en exécutant la politique EDF pour la charge processeur restante après le non-stop-runtime. Cette charge doit être inférieure au délai après le non-stop-runtime, fixé par le concepteur, avant que le système ne reçoive une nouvelle rafale de trames.
L'algorithme a été développé et testé sur un ordinateur en langage C++. L'efficacité de cette approche est mesurée en fonction du nombre de queues gérées à chaque instant, car chaque combinaison duplique le nombre de queues en fonction du nombre de trames de chaque tâche.
Nous avons expérimenté cette approche avec l'exemple du Tableau 2.2. La Figure 2.9 montre que pour un non-stop-runtime de 300 unités de temps l'approche naïve devrait gérer 149 queues alors qu'avec la nouvelle approche nous n'utilisons que 4 queues. Ce qui correspond à un gain de 97,4%. Après 300 unités de temps, l'ensemble de tâches est correctement ordonnancé (i.e. toutes les échéances temporelles sont respectées pour toutes les tâches). La charge maximale restante parmi les 4 queues est de 1 unité de temps. Ainsi si ce délai après le non-stop-runtime, est acceptable par le concepteur, le système peut être réalisé avec succès.

Cette approche a été expérimentée sur l'exemple du système du Tableau 2.3. Il est composé de 3 tâches. Chaque tâche possède 2 trames.
Tel que le montre la Figure 2.10, le gain est aussi significatif. Après 300 unités de temps, toutes les échéances temporelles sont respectées. Nous obtenons une charge maximale restante de 13 unités de temps. Alors que l'approche naïve aurait nécessité la gestion de 203046 queues, la nouvelle approche nécessite 32794 queues. Ce qui correspond à un gain de 83,85%.
La Figure 2.10 montre clairement, comment la nouvelle approche est plus efficace (i.e. moins exponentielle) que l'approche naïve.

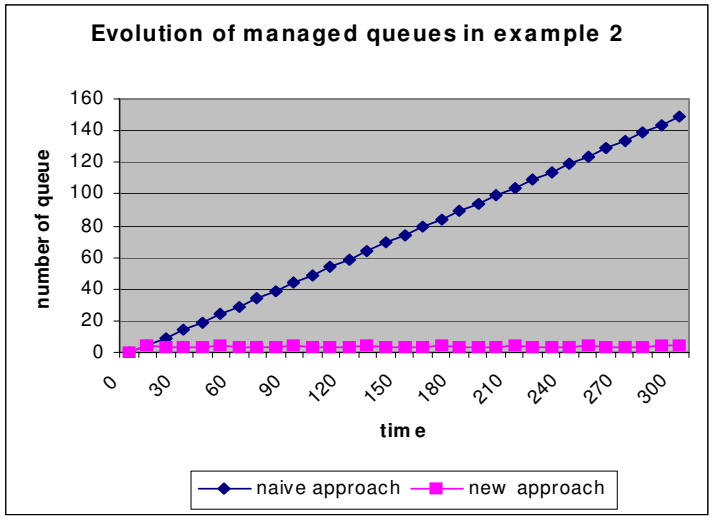

Figure 2.9 – Evolution du nombre de queues dans l'exemple 2

tâche	O_i	C_i^j	D_i^j	P_i^j
τ_1	0	2	8	12
		3	9	14
τ_2	3	4	11	16
		5	12	17
τ_3	13	6	14	18
		7	17	20

Tableau 2.3 – Système d'exemple 3

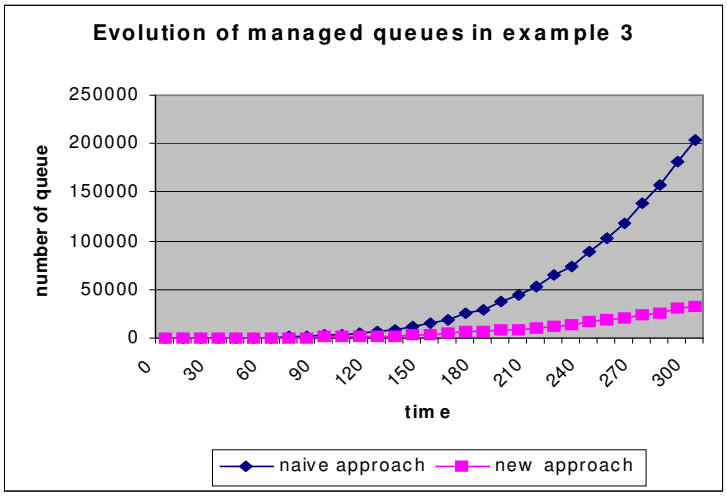

Figure 2.10 – Evolution du nombre queues dans l'exemple 3

Nous avons testé l'approche avec l'exemple du Tableau 2.4. Cet exemple est composé de 6 tâches où chaque tâche possède 2 trames.

tâche	O_i	C_i^j	D_i^j	P_i^j
τ_1	0	2	8	20
		4	15	35
τ_2	8	1	6	18
		2	12	32
τ_3	12	2	8	19
		5	17	37
τ_4	29	3	14	20

		5	16	40
τ_5	17	2	12	15
		4	20	38
τ_6	24	1	4	22
		1	5	43

Tableau 2.4 – Système d'exemple 4

Pour cet exemple, une échéance temporelle n'est pas respectée à l'instant 105.

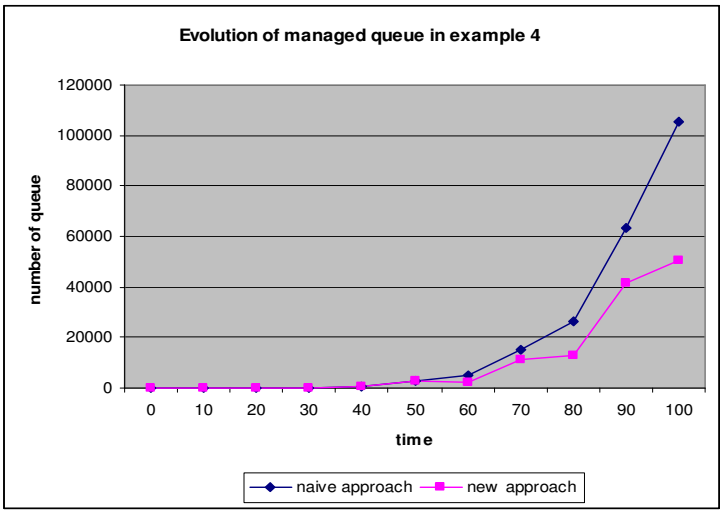

Figure 2.11 – Evolution du nombre de queues dans l'exemple 4

La Figure 2.11 montre que l'approche naïve utilise 105456 queues lorsque la nouvelle approche utilise 50228 queues, un gain de 52,37%. Notons que pour chacune de ses expérimentations, la durée effective de la simulation est de quelques minutes. Nous précisons aussi que le Théorème 2.1 n'est pas vérifié pour chacun ces exemples.

Discussion

Le modèle GMF non cyclique proposé dans ce chapitre est moins restrictif que le modèle GMF cyclique proposé en 1999 dans [65]. En effet le modèle GMF cyclique représente une séquence d'exécution possible parmi une infinité de possibilité dans le modèle non cyclique. Clairement, en considérant le modem radio de l'exemple introductif, les modèles de tâches précédents [60] [49] [65] [85] ne réussissent pas à modéliser ce type de système. Cependant en utilisant notre modèle, nous pouvons construire un modèle pour ce type de système.

2.5 Conclusion

Dans ce chapitre nous avons relâché l'hypothèse qui consiste à considérer un motif d'activation pour le modèle de tâches GMF. Dans ce contexte, nous avons présenté la formule du temps de réponse, et un test d'ordonnançabilité suffisant pour un ensemble de tâches GMF non cycliques s'exécutant suivant la politique d'ordonnancement EDF. Nous avons également présenté une approche efficace pour l'analyse exacte de la faisabilité temps réel d'un ensemble de tâches GMF non cycliques. Cette analyse utilise la simulation sur ordinateur pour déterminer la faisabilité temps réel.
En général, le modèle de tâche GMF non cyclique est fréquemment rencontré dans les applications de traitement du signal car ces applications deviennent de plus en plus flexibles et adaptatives, ainsi les flots de données ne suivent pas de motif spécifique.

Après publication de nos travaux dans une conférence phare de la communauté temps réel, ils ont été poursuivis par le professeur Sanjoy BARUAH dans [75] et [76]. Dans [75], il propose un test de faisabilité exact à complexité pseudo polynomiale avec la condition que le taux d'utilisation du processeur soit strictement inférieur à 1. Notre analyse exacte est générale pour les systèmes de tâches non cycliques car elle ne présuppose aucune hypothèse. Puis dans [76], il propose un nouveau modèle de tâche plus général que le modèle GMF non cyclique que nous avons introduit. Il présente un test de faisabilité exact pour ce modèle toujours avec la condition que le taux d'utilisation du processeur soit strictement inférieur à 1. Nous prenons en compte ce nouveau modèle dans le chapitre suivant. Bien que notre étude fût basée sur les architectures monoprocesseurs, nous précisons tout de même que nous avons généralisé le Théorème 2.1 pour les architectures multi-processeurs. Cette généralisation a fait l'objet du dépôt d'un brevet d'invention.

Chapitre 3

Contributions : Méthodologie de conception et outillage

Résumé : *Dans la radio logicielle idéale, tous les algorithmes de traitement du signal doivent être implantés sous forme logicielle. Etant donné le grand nombre de composants logiciels à exécuter simultanément sur un même processeur pour les futurs équipements radios haut débit, les concepteurs s'interrogent sur les méthodologies de conception pour l'ordonnancement temps réel en adéquation avec ces évolutions. Dans ce chapitre nous proposons une méthodologie de conception qui permet d'effectuer l'analyse d'ordonnancement temps réel des tâches dans une radio logicielle. Aussi, nous présentons dans ce chapitre une approche efficace permettant de vérifier l'ordonnancement temps réel des tâches implantant des algorithmes de traitement du signal flexibles et s'exécutant sur un processeur suivant une politique d'ordonnancement hybride (combinaison de la politique d'ordonnancement à priorité fixe et de la politique d'ordonnancement « Earliest Deadline First »). Ces méthodes permettent aussi de mieux repartir les composants à implanter sous forme logicielle et ceux à implanter sous forme matérielle dans un équipement radio.*

3.1 Introduction

Avec la multiplication des réseaux (et standards) mobiles et sans fil, la couche physique des systèmes de communications (i.e. la partie du modem du système) doit être complètement flexible afin que les équipements puissent s'adapter à plusieurs modes de fonctionnement. La couche physique doit supporter une large variété d'algorithmes complexes qui dépendent de nombreux paramètres. Certains de ces algorithmes en outre ne sont pas encore finalisés ou standardisés et peuvent évoluer ou changer. Un grand nombre de modems radios actuellement commercialisés ont des composants majoritairement mis en œuvre sous forme matérielle (i.e. sur FPGA ou sur ASICs). La partie logicielle de ces modems radio est restreinte à la réalisation de fonctions complémentaires telles que la programmation des fréquences porteuses ou le suivi de la performance. Il est généralement considéré irréalisable d'ajouter plus d'algorithmes de traitement du signal sous forme logicielle à cause de leurs complexités en temps de calcul et des problèmes d'ordonnancement temps réel que cela induit. Cependant une solution

logicielle (telle que voulue dans l'approche SDR) pourrait apporter quelques avantages par rapport à son équivalent matériel en termes de :

- faible coût de développement,
- réduction du temps de mise sur le marché,
- grande portabilité des IP (Intellectual Property) de traitement du signal,
- mise à jour facile lors des évolutions.

Dans les perspectives offertes par les futurs équipements radio haut débit, une approche logicielle entrainera plusieurs tâches implantant des algorithmes de traitement du signal (émanant de standard radio différent) à s'exécuter simultanément sur un même processeur. Cependant, ces tâches doivent produire des données dans un intervalle de temps borné. Elles sont soumises à des contraintes temporelles strictes.

Les méthodologies de conception proposées dans la littérature [11] [79], et les outils commerciaux [80] [81] fournissent différentes aides afin de faciliter le processus de conception, allant du développement de modèles visuels, en passant par l'évaluation des performances, jusqu'à la génération automatique du code pour le déploiement sur cible.

Malheureusement, aucun de ces outils de conception ne fournit un support pour vérifier l'ordonnancement temps réel des tâches implantant des algorithmes de traitement du signal flexibles dans une radio logicielle. Pire encore, il n'est pas possible d'utiliser les outils libres ou commerciaux d'analyse d'ordonnancement temps réel disponible à ce jour car il existe un fossé entre le modèle d'analyse des outils d'ordonnancement temps réel et les modèles générés par les outils de conception. Une raison qui justifie cela vient du fait que les modèles de réalisations générés par les outils de conception sont basés sur des applications dirigées par des évènements. Dans une telle application, chaque tâche gère plusieurs types d'évènements, ce qui fait en sorte que son temps d'exécution, son échéance temporelle et son temps de garde varient d'une activation à une autre en fonction du type d'évènement qui est traité. Ceci est complètement différent du modèle de tâche considéré dans les outils d'ordonnancement temps réel à ce jour, où les tâches gèrent en général un seul type d'événement et sont soumises à une seule contrainte de bout en bout.

Ce chapitre fournit les contributions suivantes :

- Une méthodologie de conception qui permet l'analyse d'ordonnancement temps réel des tâches dans une radio logicielle. Nous présentons un ensemble de règles à suivre pendant les phases de modélisation PIM (Platform Independent Model), PDM (Platform Description Model) et PSM (Platform Specific Model), en considérant en particulier le profil MARTE [7].
- Une approche pour la vérification de l'ordonnancement temps réel des tâches implantant des algorithmes de traitement du signal flexibles et s'exécutant sur un processeur en fonction d'une politique d'ordonnancement hybride.

Le reste de ce chapitre est organisé comme suit. La section 3.2 rappelle les travaux connexes. La section 3.3 décrit la nouvelle méthodologie de conception que nous proposons. La section 3.4 présente notre algorithme pour l'analyse d'ordonnancement

temps réel des tâches réalisant des algorithmes de traitement du signal dans une radio logicielle. Une conclusion est présentée en section 3.5.

3.2 Travaux connexes

Un grand nombre de travaux de recherche ont été entrepris dans le domaine des méthodologies de conception pour les applications de traitement du signal. Plusieurs de ces tentatives sont des méthodologies basées sur HDL (Hardware Description Language) [10] [19] [70] [79] [82], alors que d'autres [83] [84] sont des méthodologies pour des réalisations entièrement logicielles. Dans [83], Saksena et al ont intégré les résultats d'analyse d'ordonnancement pour le modèle de tâches avec transaction [49] [56], sur la conception objet en utilisant le profil « UML-RT ». Bartolini et al présentent dans [84] un algorithme permettant de « mapper » un flot de données sur un ensemble de tâches temps réel. Aussi, certaines approches génériques [86] [87] [88] ont été proposées dans la littérature pour la conception de systèmes temps réel embarqués. Notre méthodologie de conception est différente car : (1) nous considérons le nouveau modèle de tâche récurrente non cyclique introduit dans [76] après la publication de nos premiers travaux, puisque ce modèle représente encore mieux le comportement des tâches dans une radio logicielle, (2) nous utilisons le nouveau profil UML pour les systèmes temps réel embarqués récemment standardisé à l'OMG (MARTE). Par conséquent, les approches précédentes de conception ne peuvent pas être appliquées.

3.3 Notre méthodologie de conception

Notre méthodologie de conception de modems radio logiciels est basée sur l'approche MDA (Model Driven Architecture). Pour cette raison le processus de conception est divisé en trois étapes distinctes. Dans la première phase, les aspects fonctionnels du modem radio sont abordés. Il s'agit des fonctionnalités métiers du modem radio (schéma de modulation, type de codeur etc.) sans aucun détail de la plateforme où ces fonctionnalités vont être implantées. Dans la deuxième phase, la plateforme matérielle est spécifiée, en termes de processeurs de traitement du signal numérique (Les DSPs par exemple), de la quantité de mémoire disponible, du mécanisme de connectivité entre les tâches, du système d'exploitation temps réel utilisé etc. Dans la dernière phase, la spécification fonctionnelle est « mappée » sur la plateforme matérielle, les différentes activités à effectuer sur un processeur sont assignées sous forme de tâches temps réel. Ensuite les temps d'exécutions et les échéances temporelles des tâches sont calculés puis renseignés dans le modèle. La taille de buffer de données nécessaire pour chaque tâche est aussi renseignée.

Après ces étapes, nous proposons de transformer les modèles venant de l'outil de modélisation vers un modèle compréhensible par l'outil d'analyse d'ordonnancement temps réel que nous avons développé dans cette thèse. L'outil d'analyse d'ordonnancement supporte le nouveau modèle de tâche récemment introduit par Baruah [76] suite à des échanges sur nos travaux. Il effectue une simulation du comportement

temporel des tâches en se basant sur des arrivées de trames distribuées de manière aléatoire dans le temps. Nous présentons également les concepts de MARTE qui doivent être utilisés pour exprimer des propriétés non fonctionnelles.

La Figure 3.1 résume la démarche de notre flot de conception. Le concepteur réalise d'abord un « état de configuration fonctionnel », il s'agit de l'état vers lequel le poste radio va se reconfigurer. Il est composé d'un ensemble de formes d'ondes. Puis le concepteur réalise un « état de configuration physique ». Il s'agit du « mapping » de « l'état de configuration fonctionnel » vers les différentes unités d'exécutions. Cette projection est suivie du rajout des différentes caractéristiques temporelles des tâches et des services logiciels déployés sur le processeur d'intérêt. Enfin des fichiers générés par l'outil de modélisation sont transformés et utilisés pour l'analyse d'ordonnancement temps réel.

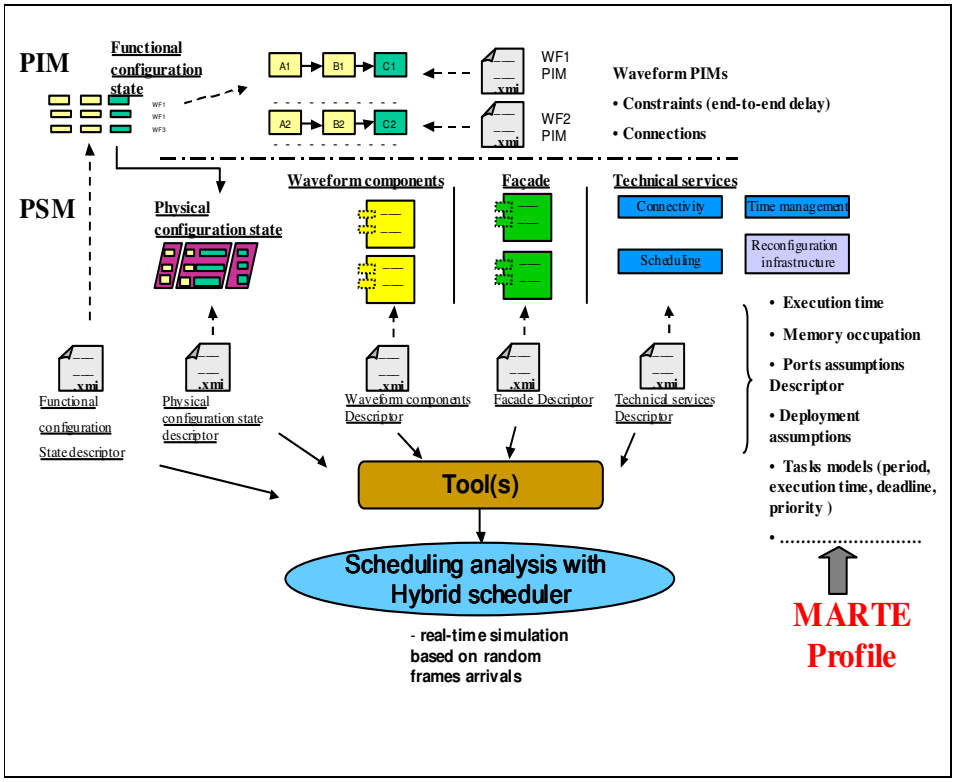

Figure 3.1 – Notre flot de conception

Dans l'approche MDA, la première phase consiste à réaliser un modèle indépendant de toute plateforme.

3.3.1 Modélisation PIM

Le modèle PIM peut contenir plusieurs chaînes de traitement radio (chacune correspondant à un standard). Pour chaque chaîne radio, nous représentons deux vues : une vue architecturale du modem radio, et une vue comportementale.

L'architecture de l'application est décrite de manière « top-down » et nous proposons cinq niveaux de décomposition (comme présenté dans la Figure 3.2) : le niveau capacité,

le niveau en couche, le niveau applicatif, le niveau de support fonctionnel, et le niveau du module de base.
Nous proposons l'utilisation des diagrammes de structures composites d'UML pour décrire progressivement chacun de ces niveaux. Ce type de diagramme est apparu avec UML 2.0, il permet de décrire la structure interne d'une classe. Ainsi, pour la modélisation PIM, chaque élément dans notre flot de conception est représenté sous forme d'une classe UML, et sa structure interne est décrite avec un diagramme de structure composite.

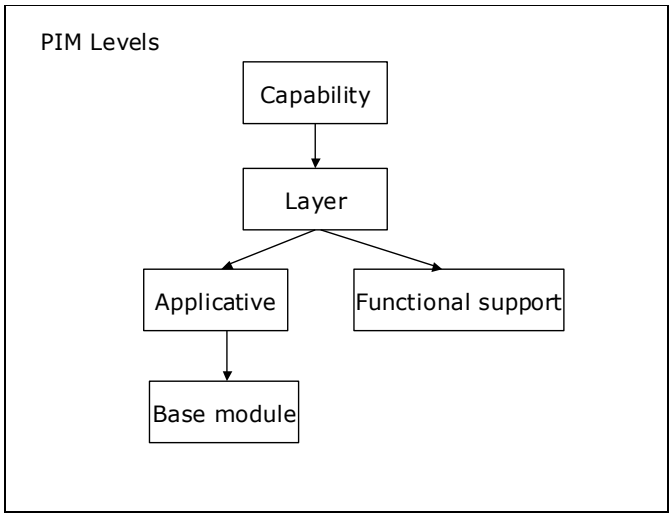

Figure 3.2 – Différents niveaux de décomposition PIM

Le niveau capacité (« capability ») permet d'exprimer les « exigences boîtes noires ». Ce sont les interfaces externes (micro, socket, etc.), les entrées utilisateurs, le canal de communication, la topologie du réseau, le débit etc. A ce niveau le concepteur doit renseigner la latence de bout en bout pour l'analyse d'ordonnancement temps réel.
Le niveau en couche (« Layer ») est en conformité avec le modèle OSI, autrement dit, à ce niveau on retrouve toutes les différentes couches du modèle OSI. En fonction du délai de bout en bout, le concepteur alloue une latence globale pour chaque composant. De cette latence globale sera dérivée les échéances temporelles des blocs internes de chaque composant. Une couche se décompose en deux sous systèmes : l'applicatif (« Applicative ») et le support fonctionnel (« Functional Support »). La raison de cette décomposition est qu'ils réalisent des fonctionnalités métiers différentes. Par exemple la couche physique d'une radio logicielle va se décomposer en un modem qui produit le signal en bande de base et un transceiver qui réalise par exemple les conversions numériques/analogiques, l'amplification du signal etc. Ainsi, un support fonctionnel peut être défini comme étant le traitement du domaine radio, dont l'implantation nécessite du matériel et quelques autres spécificités du domaine radio. La Figure 3.3 illustre cet exemple de décomposition avec un diagramme de structure composite. Il s'agit d'une couche physique générique d'une radio logicielle. La classe « PHY » représente la couche physique et à l'intérieur, nous retrouvons une classe pour le composant « MODEM », une

classe pour le composant « Transceiver » et une classe pour l'antenne. Les ports des composants sont reliés grâce à des connecteurs. En haut des connecteurs, nous avons le nom des interfaces réalisées et utilisées par les différents ports. Les interfaces « ReceiveDataPush » et « TransmitDataPush » permettent l'envoi et la réception des données utiles. Les interfaces « ReceiveControl » et « TransmitControl » permettent la programmation des fréquences porteuses. Ce sont les interfaces standardisées au SDR Forum [92] pour l'échange d'informations entre la partie modem d'une radio et son transceiver (convertisseur numérique/analogique, carte RF/IF etc.). Les ports d'entrée/sortie (« inofdm », « outofdm », « inphy », et « outphy ») permettent les échanges d'information entre le modem de la couche physique et les autres parties du système. Le port « rfsignal » correspond au signal radio.

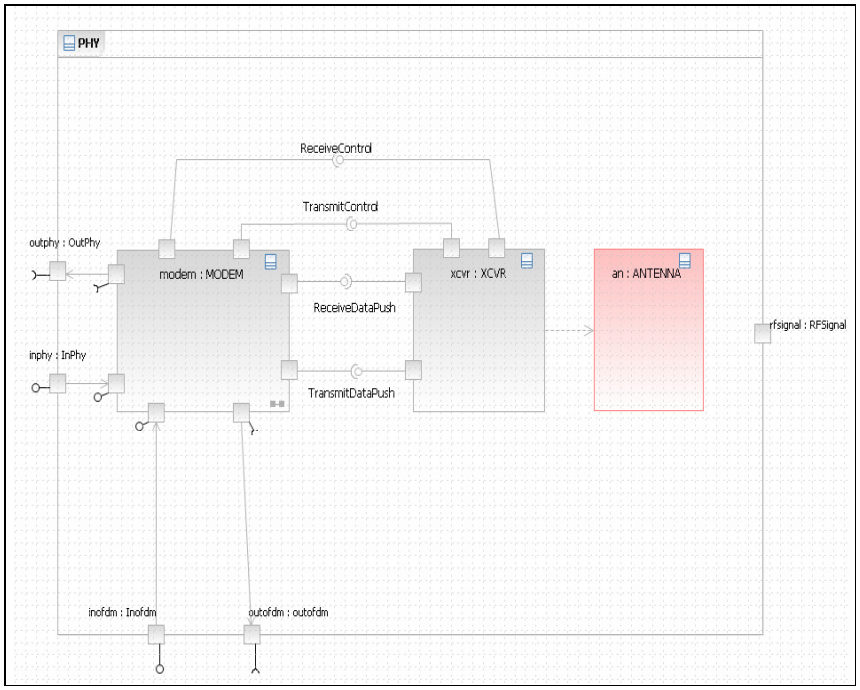

Figure 3.3 – Illustration d'une couche physique générique

L'applicatif se décompose en un ensemble de modules de base. Un module de base est défini comme un module élémentaire qui ne peut pas être divisé en sous parties dans le but de distribuer les sous parties dans des processeurs différents. La raison principale d'une telle situation est que, compte tenu de l'interaction récurrente entre les sous parties, un temps de transfert plus long entre sous parties distribuées produira une implantation moins efficace.

Nous avons défini un profil UML pour les différents niveaux de notre méthodologie. Ce profil s'appelle « WFP » (« Waveform Properties » – en anglais) et est composé de stéréotypes correspondant chacun à un des niveaux PIM définis dans cette section. Pour pouvoir utiliser ce profil il faut l'ajouter en tant que profil disponible pour le modèle que nous sommes en train de réaliser. La Figure 3.4 présente notre profil. A gauche nous pouvons voir la liste des stéréotypes et à droite nous avons un exemple d'application du

stéréotype « Basemodule » correspondant au niveau du module de base. A gauche du nom du module on peut également voir apparaître l'icône que nous avons réalisé pour ce stéréotype.

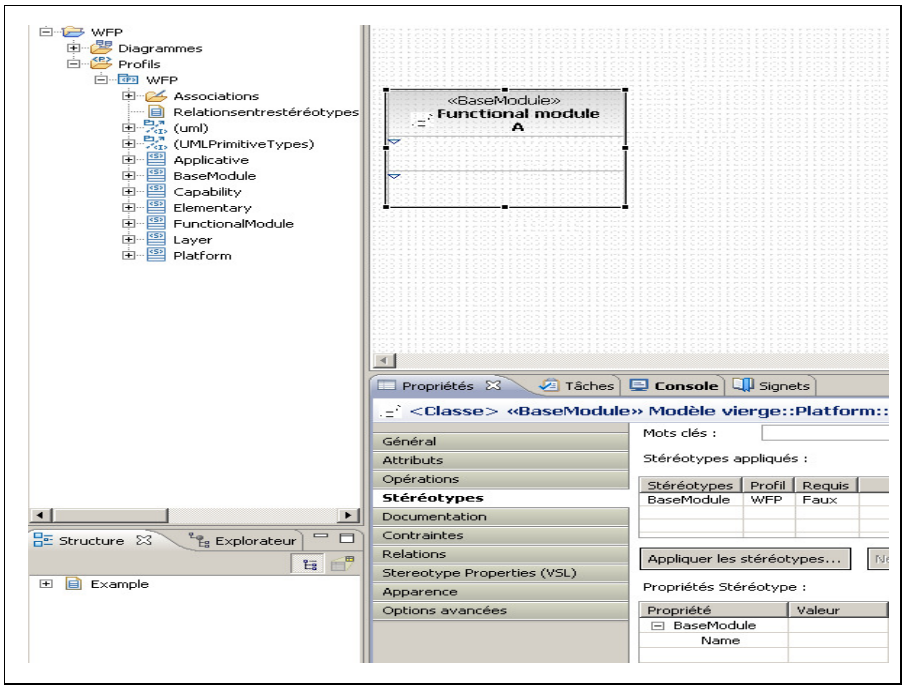

Figure 3.4 – Profil correspondant à notre flot de conception

Pour la représentation comportementale, nous décrivons le comportement des composants à chaque niveau PIM. On peut par exemple retrouver un diagramme qui décrit les échanges de messages entre les modules de base. Nous proposons l'utilisation des diagrammes de séquences pour représenter le comportement des composants. Un exemple de représentation est présenté dans le chapitre suivant.

3.3.2 Modélisation PDM

La deuxième phase consiste à décrire la plateforme. Une plateforme matérielle devant accueillir une radio logicielle est généralement composée de plusieurs processeurs, mais nous nous intéressons à la conception sur un processeur d'intérêt. Pour la modélisation PDM, nous proposons dans notre méthodologie, l'utilisation des diagrammes de classes.
Le stéréotype MARTE « GRM ::Scheduler » est utilisé afin de caractériser l'ordonnanceur du système d'exploitation temps réel. Typiquement, l'attribut « ispreemptible » est utilisé pour spécifier le fait que les tâches gérées par cet ordonnanceur sont préemptives ou pas. L'attribut « schedulingpolicy » est utilisé pour définir la politique d'ordonnancement.
La Figure 3.5 présente sur un diagramme de classes, un exemple simple de modélisation PDM. Il est composé de deux classes : une classe représentant un processeur et une autre représentant un système d'exploitation temps réel.

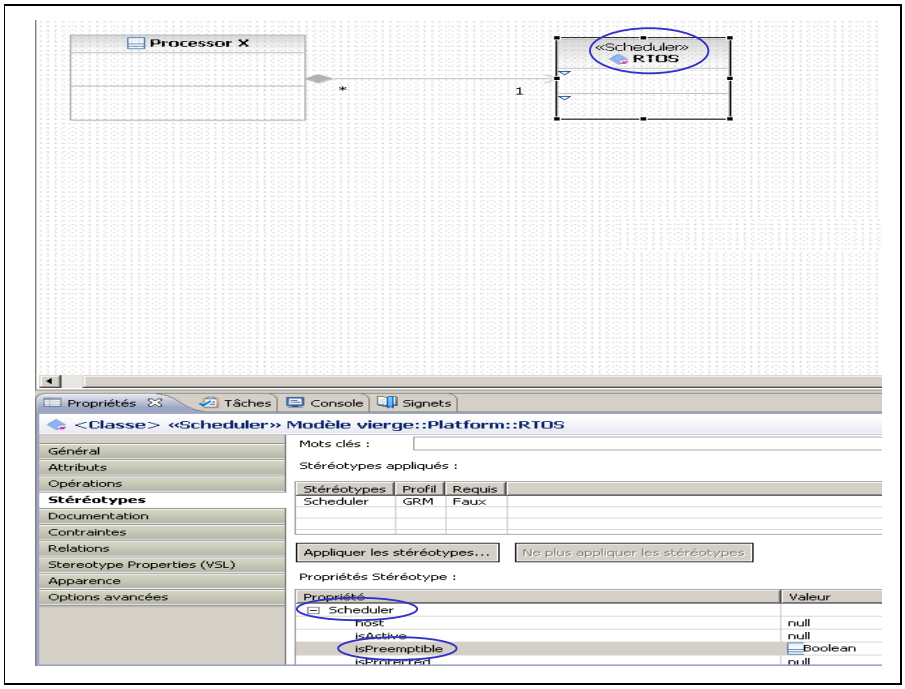

Figure 3.5 – Exemple de modèle de plateforme

Après avoir appliqué le stéréotype MARTE, le nom du stéréotype apparaît au-dessus du nom de l'élément sur lequel nous avons appliqué le stéréotype. Par exemple sur la Figure 3.5, nous voyons le nom « scheduler » apparaître au dessus du nom « RTOS ». Apparaît également à coté du nom de chaque élément l'icône correspondant au stéréotype. Aussi, dans les propriétés de l'élément, nous voyons apparaître le stéréotype et la liste de ses attributs. Comme illustré dans la Figure 3.5, nous proposons d'utiliser une relation de composition entre le processeur et ses différents composants logiciels ou matériels.

Le stéréotype MARTE « HRM ::Hwmemory » permet de décrire la taille de la mémoire ram sur le processeur d'intérêt.

Le stéréotype MARTE « GRM ::ResourceUsage » est utilisé pour décrire les caractéristiques temporelles du mécanisme de connectivité utilisé entre les tâches.

L'attribut « Exectime » permet d'exprimer le temps de transfert d'un buffer entre deux tâches.

Dès que la plateforme est suffisamment décrite, la dernière étape consiste à « mapper » le modèle PIM sur une plateforme spécifique.

3.3.3 Modélisation PSM

Pendant la modélisation PSM, différentes plateformes peuvent être utilisées pour évaluer un même modèle. Pour chaque module de base du PIM, un nouveau module doit être créé correspondant à son implantation. Il y a une relation de généralisation entre les deux modules dans le but de spécifier que le module implanté hérite de tous les attributs, propriétés, fonctions et relations définis pour ce même module au niveau PIM. Nous

séparons le module de base défini au niveau PIM et son implantation définie au niveau PSM car il peut y avoir plusieurs implantations d'un même module. Par exemple une implantation optimisée en vitesse d'exécution mais qui occupe beaucoup de mémoire, et une autre prenant peu de mémoire mais avec un grand temps d'exécution.

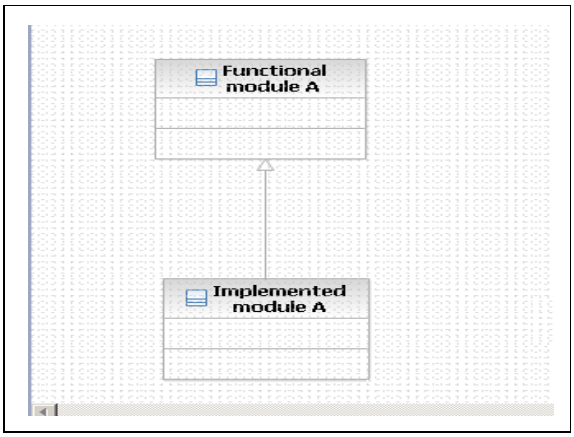

Figure 3.6 – Exemple de relation entre un module PIM et son implantation PSM

Un exemple de relation entre le module PIM et son implantation PSM est présenté dans la Figure 3.6. Nous utilisons les diagrammes de classes pour représenter cette relation.
Nous passons maintenant à la phase d'allocation.

L'allocation

Le stéréotype MARTE « Alloc ::ApplicationAllocationEnd » est appliqué sur les différents modules implantés. Ainsi l'attribut « allocatedto » est utilisé pour sélectionner le processeur où sera implanté le module. Le stéréotype MARTE « alloc ::allocated » est utilisé pour spécifier de manière visuelle la relation d'allocation du module implanté sur une plateforme matérielle. Les modules implantés sont assignés en tant que tâches logicielles temps réel avec le stéréotype MARTE « SRM ::SwschedulableResource ». La Figure 3.7 montre un exemple d'allocation de module. Nous pouvons également voir sur la figure le stéréotype « SRM ::SwschedulableResource » appliqué au module implanté. L'allocation se fait également avec les diagrammes de classes. Le lien entre le module implanté et le processeur sur lequel il va être déployé, correspond à une relation de dépendance.
Bien que la spécification de MARTE propose le package « SAM » pour l'analyse d'ordonnançabilité ([7] page 311), nous n'avons utilisé que le stéréotype « SAM ::SaAnalysisContext » de ce package pour spécifier le type d'analyse. Ainsi nous n'avons pas utilisé les autres stéréotypes proposés dans ce package (« SAM ::Sastep », «SAM::Gascenario », « SAM::Endtoendflow », etc.) car ils permettent de spécifier des contraintes sur une fonction ou une activité. Or nous voulons assigner des contraintes sur des tâches, chacune des tâches réalisant une ou plusieurs fonctions. En effet les systèmes d'exploitation temps réel ordonnancent des tâches (ou « threads » au sens POSIX), c'est la raison pour laquelle les contraintes temporelles sont généralement (y compris dans la

littérature) rapportées sur des tâches. Cependant le profil propose le package « SRM » ([7] page 197) pour définir les ressources logicielles dans un système multitâches. En outre, le stéréotype « Swschedulableresource » du package « SRM » ([7] page 227) est proposé pour définir une ressource logicielle qui s'exécute de manière concurrente à d'autres ressources. C'est pourquoi nous avons choisi ce stéréotype.
Après l'allocation nous pouvons réaliser une caractérisation temps réel de chaque tâche.

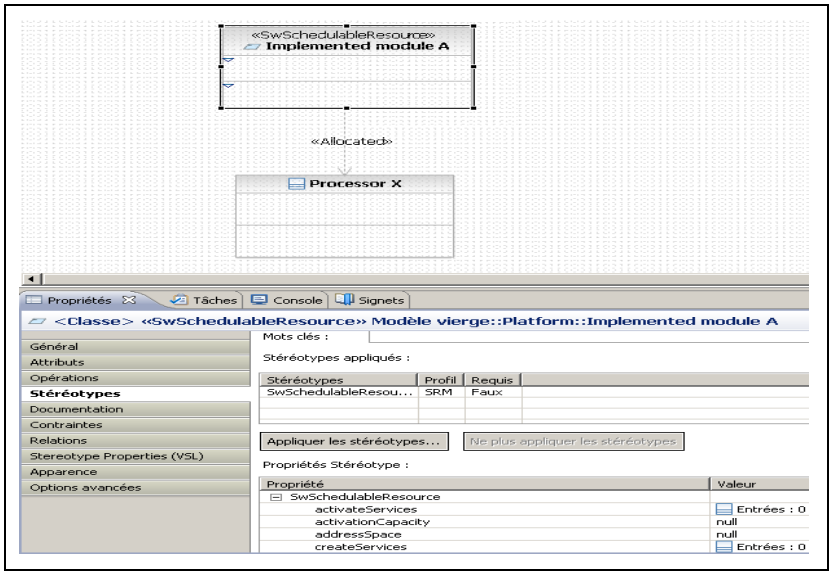

Figure 3.7 –Exemple d'allocation de module

Calcul des contraintes temporelles

Le temps d'exécution de chaque module implanté est calculé en mesurant le temps que prend le module lorsqu'il s'exécute tout seul sur le processeur.
Pour calculer les échéances temporelles relatives de chaque module, nous calculons dans un premier temps la date d'activation au plus tôt de chaque module. Pour cela les algorithmes constituant la partie modem des traitements sont représentés sous forme de graphe direct acyclique où chaque module implanté est représenté par un nœud. Un algorithme utilise la méthode de parcours de graphe en profondeur où le graphe est parcouru en profondeur à partir d'une source avant d'aller sur les autres nœuds. Pour chaque nœud et arc rencontré dans le parcours, l'algorithme somme les latences rencontrées et les dates d'activations au plus tôt sont assignées à chaque nœud.
Dans un second temps, les échéances temporelles absolues sont calculées. Un algorithme de parcours en profondeur identique à celui utilisé pour le calcul des dates d'activation au plus tôt, mais juste en direction inverse, permet de calculer les échéances temporelles absolues. Partant du puits du graphe, l'échéance temporelle globale est diminuée en fonction de la latence des nœuds et des arcs rencontrés jusqu'à la source du graphe. Ainsi l'échéance temporelle absolue correspondante à chaque nœud est assignée.
Enfin l'échéance temporelle relative de chaque nœud correspond à la différence entre son échéance temporelle absolue et sa date d'activation au plus tôt.

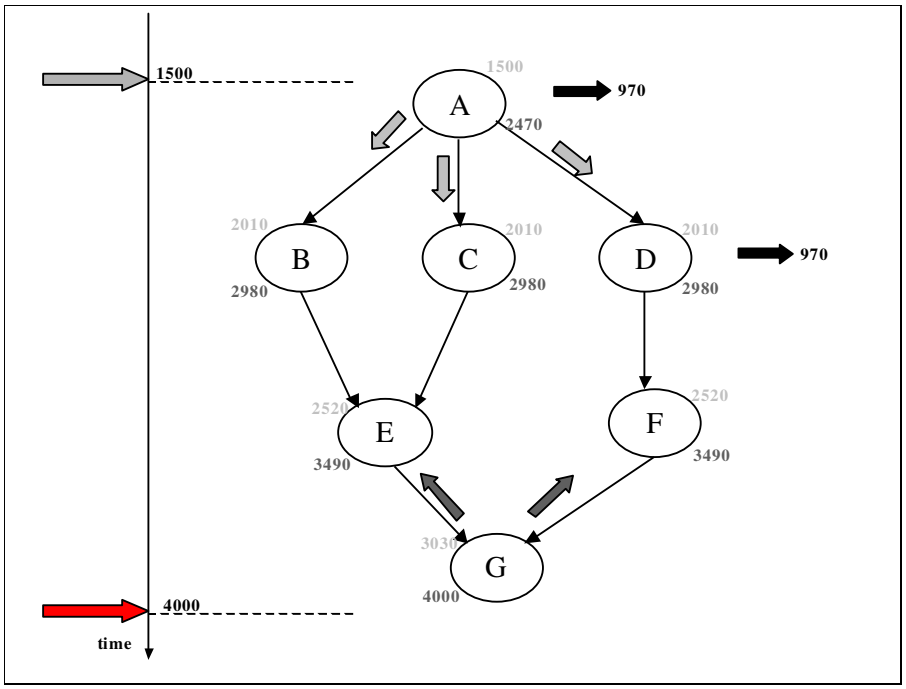

Figure 3.8 – Méthode de calcul des échéances temporelles

La Figure 3.8 montre comment nous déroulons cet algorithme sur un exemple. La figure représente une chaine de traitement radio où A, B, …, G représentent des algorithmes de traitement du signal. Nous considérons par exemple l'arrivée d'une trame à l'instant 1500 et une échéance temporelle absolue à l'instant 4000. Les valeurs sont en millisecondes et chaque algorithme nécessite 500 ms de temps d'exécution. Le transfert des données entre les algorithmes nécessite 10 ms (valable pour chaque arc). Ainsi les valeurs en gris clair (en haut à droite) correspondent aux dates de démarrage au plus tôt tandis que celles en gris foncé (en bas à droite) correspondent aux échéances temporelles absolues de chaque nœud. Les valeurs en noir correspondent aux échéances temporelles relatives. Cependant lors du calcul des dates de démarrage au plus tôt, une attention doit être portée pour les modules déployés sur un même processeur, ainsi si dans notre exemple B et C sont sur un même processeur, la date de démarrage au plus tôt de E doit être la somme des chemins B et C, et non le maximum des deux chemins.

Cet algorithme a été implanté en langage C++ sous forme d'un outil afin d'automatiser le calcul en phase de conception. Nous verrons son utilisation dans le cas d'études présenté dans le chapitre suivant.

Notons que ces calculs doivent être effectués pour chaque type de trame.

Les attributs « deadlineselements », « identifierelements » et « messageresource » du stéréotype « SRM ::Swschedulableresource » permettent de renseigner respectivement les échéances temporelles, l'identifiant, et la taille des buffers de donnés des tâches. L'attribut « notificationsresources » permet de spécifier les tâches suivantes activées à la fin de l'exécution de la tâche courante. Nous utilisons l'attribut « type » afin de spécifier le type de la tâche. Dans les modems radio logiciels, la plupart des tâches sont activées suite

à l'arrivée d'événements (arrivée de trames). De cette première grande famille, nous distinguons quatre sous-types de tâches dans la radio logicielle : Les tâches activées uniquement pendant la phase de transmission (une tâche réalisant une fonction de codage par exemple), les tâches activées uniquement pendant la phase de réception (une tâche réalisant une fonction de décodage par exemple), les tâches activées dans les deux phases (la tâche qui réceptionne les requêtes venant de la couche supérieure), puis les tâches qui sont périodiques pendant un intervalle de temps, après la réception d'un événement (la tâche qui programme les fréquences porteuses). Ce dernier type de tâche est généralement rencontré dans les radios logicielles avec sauts de fréquences. En effet les trames sont subdivisées en paliers et chaque palier est émis à une fréquence porteuse spécifique. Afin d'illustrer le comportement de ce dernier type de tâche, nous considérons deux trames (trame 1 et trame 2) où la trame 1 est subdivisée en 3 paliers et la trame 2 est subdivisée en 7 paliers.

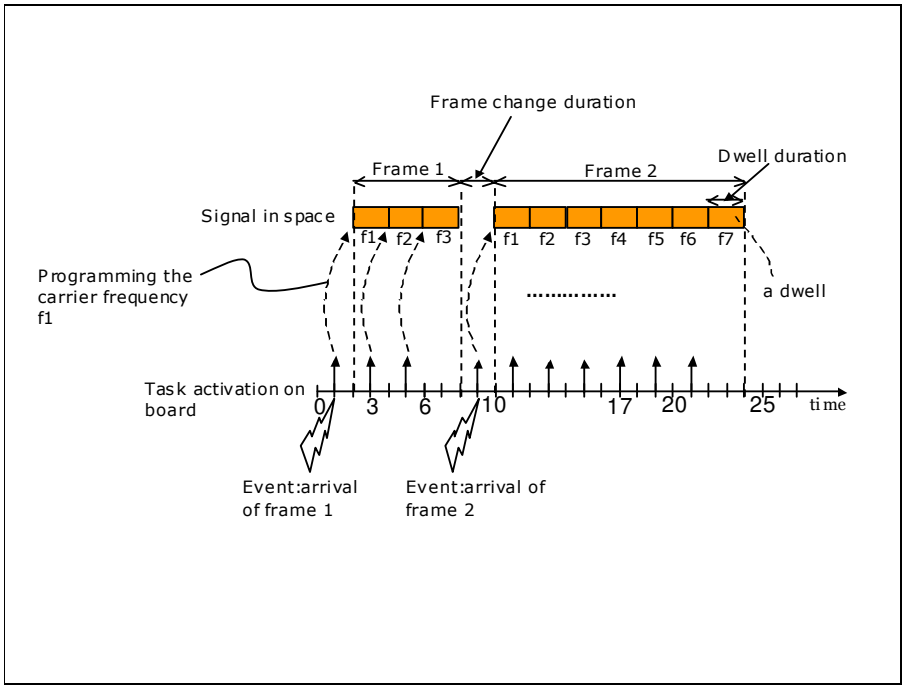

Figure 3.9 – Comportement de la tâche qui programme les fréquences porteuses

Tel que le montre la Figure 3.9, suite à l'arrivée de la trame 1, la tâche est activée 3 fois. Les activations, qui suivent l'activation déclenchée par la réception de la trame, sont séparées de 2 unités de temps (correspondant à la durée du palier). Puis, suite à l'arrivée de la trame 2, la tâche est activée 7 fois. Les activations qui suivent l'activation déclenchée par la réception de la trame sont aussi séparées de 2 unités de temps. Les trames peuvent arriver dans n'importe quel ordre. La tâche qui programme les fréquences porteuses ne doit jamais rater ses échéances temporelles car ceci résulterait en une perte du signal radio, ce qui est inacceptable pour l'utilisateur.
L'attribut « timeslicelements » du stéréotype « SRM ::Swschedulableresource » est utilisé pour spécifier les temps d'exécutions d'une tâche.

Ordonnanceur hybride

Nous proposons l'utilisation d'un ordonnanceur hybride comme politique d'ordonnancement pour les tâches d'une radio logicielle. Un ordonnanceur hybride correspond à la combinaison d'un ordonnanceur à priorité fixe (FP) et d'un ordonnanceur à priorité dynamique (EDF). La motivation pour l'utilisation d'un ordonnanceur hybride, est d'exploiter les avantages des deux types d'ordonnanceurs. L'ordonnanceur à priorité fixe pour sa prévisibilité et EDF pour son efficacité. Ainsi les tâches ayant une criticité très forte ou permettant de contrôler le modem, seront gérées avec l'ordonnanceur à priorité fixe, alors que les autres tâches seront gérées avec EDF, conceptuellement comme des tâches de basse priorité.

L'attribut « scheduler » du stéréotype « SRM ::Swschedulableresource » est utilisé pour spécifier le type d'ordonnanceur qui gère la tâche. Ainsi, l'attribut « priorityelements » est utilisé pour renseigner la priorité des tâches gérées par l'ordonnanceur à priorité fixe. Nous utilisons l'attribut « periodelements » pour renseigner la pseudo-période des tâches qui programment les fréquences porteuses.

Dès que la caractérisation temps réel est terminée, nous proposons la transformation des modèles UML et des concepts MARTE vers un modèle compréhensible par notre outil d'ordonnancement temps réel que nous avons développé durant ces travaux.

3.3.4 Transformation de modèles

Une transformation de modèle est nécessaire afin que les modèles réalisés avec les outils de modélisations puissent être interprétables par un outil d'analyse d'ordonnancement temps réel. Il existe plusieurs outils de transformations de modèles, certains sont basés sur QVT (Query/View/Transformation), un standard OMG pour effectuer la transformation de modèles. Les outils basés sur QVT fournissent des moyens de spécifier la manière dont est produit un modèle cible à partir d'un modèle source. Dans notre cas il faut fournir :

- Le métamodèle d'UML et de MARTE
- Le métamodèle de l'outil d'analyse d'ordonnancement temps réel
- Le modèle de l'application source
- Le modèle de transformations composé de règles, qui « mappent » les concepts MARTE, typiquement les stéréotypes et attributs utilisés pendant la modélisation, vers les concepts de l'outil d'analyse d'ordonnancement.

Le Tableau 3.1 présente la correspondance entre les concepts de l'outil et ceux de MARTE.

	Concepts outil de simulation	Concepts MARTE
Nom	DSPSIMU	« SAM::SaAnalysiscontext »
Description	Une simulation avec un nombre de trames précis générées aléatoirement	Contexte d'analyse racine où sont collectées les informations spécifiques à l'analyse

Règles	L'élément avec le stéréotype SAM ::SaAnalysiscontext correspond à DSPSIMU	
Vérification	Une erreur est remontée si ce stéréotype n'est appliqué à aucun élément	
Nom	Processor	« HRM ::Hwprocessor »
Description	Processeur pour lequel on effectue l'analyse temps réel	Ressource de calcul matérielle
Règles	Le nom de l'élément avec le stéréotype « HRM ::Hwprocessor » correspond au nom du processeur	
Vérification	Un avertissement est remonté si ce stéréotype n'est appliqué à aucun élément	
Nom	Scheduler	« GRM ::Scheduler »
Description	Ordonnanceur du système d'exploitation temps réel	Ordonnanceur du système d'exploitation temps réel
Règles	– L'attribut « Schedpolicy » du stéréotype correspond à la politique d'ordonnancement appelé « Schedulingpolicy » dans l'outil – L'attribut « isPreemptible »du stéréotype précise si l'ordonnanceur est préemptif (variable « preemptible » dans l'outil)	
Vérification	Une erreur est remontée si aucun élément n'a ce stéréotype ou si l'attribut « Schedpolicy » n'est pas renseigné	
Nom	Task	« SRM ::Swschedulableresource »
Description	Tâche logicielle à ordonnancer	Contexte de calcul logique où la compétition pour le processeur est déterminée par l'ordonnanceur. Ressource qui s'exécute de manière concurrente à d'autres ressources
Règles	– Le nom de l'élément correspond au nom de la tâche – L'attribut « deadlineselements » contient la liste d'échéances temporelles (tableau alloué de manière dynamique portant le nom « Deadline » dans l'outil) – L'attribut « priorityelements » contient la priorité de la tâche si elle est gérée avec l'ordonnanceur FP (variable « Prio » dans l'outil)	

	– L'attribut « identifierelements » contient l'identifiant de la tâche si elle est gérée par l'ordonnanceur EDF (variable « Id » dans l'outil) – L'attribut « timeslicelements » contient la liste des temps d'exécution (tableau alloué de manière dynamique portant le nom « WCET » dans l'outil) – L'attribut « notificationsresources » correspond à la liste des tâches suivantes (tableau alloué de manière dynamique portant le nom « Nexttask » dans l'outil) – L'attribut « scheduler » précise le type d'ordonnanceur qui gère la tâche (variable « Policy » dans l'outil) – L'attribut « entrypoints » correspond à l'intervalle de temps après lequel nous devons activer une tâche suite à l'arrivée d'un événement (tableau alloué de manière dynamique portant le nom « Offset » dans l'outil) – L'attribut « type » précise le sous-type de tâche (variable « Type » dans l'outil) – L'attribut « periodelements » correspond à la pseudo-période de certaines tâches (tableau alloué de manière dynamique portant le nom « PseudoPeriod » dans l'outil) – L'attribut « messageresources » correspond à la taille du buffer donc a besoin la tâche pour effectuer son traitement (tableau alloué de manière dynamique portant le nom « BufferR » dans l'outil)
Vérification	Une erreur est remontée si l'attribut « priorityelements » n'est pas renseigné alors que la tâche est gérée par un ordonnanceur FP. Une erreur est remontée si « identifierelements » n'est pas renseigné alors que la tâche est gérée par l'ordonnanceur EDF. Une erreur est remontée si une tâche n'a pas au moins une échéance temporelle, un temps d'exécution et son sous-type ne sont pas renseignés.

Tableau 3.1 – Table de correspondance entre l'outillage et MARTE

Ces règles ont été implantées en langage ATL. La Figure 3.10 résume le processus de transformation de modèles. Le modèle source (« Source Model ») conforme au métamodèle d'UML et de MARTE est transformé en un modèle cible (« Target Model ») conforme au métamodèle de notre outil de simulation. La transformation est définie par le modèle de transformation (« Source2Target ») qui lui-même est conforme au métamodèle du modèle de transformation basé sur QVT (« QVT-Based language »). Ce dernier métamodèle avec le métamodèle d'UML et de MARTE, plus le métamodèle de

notre outil de simulation doivent être conformes au métamétamodèle MOF (Meta-Object Facility).

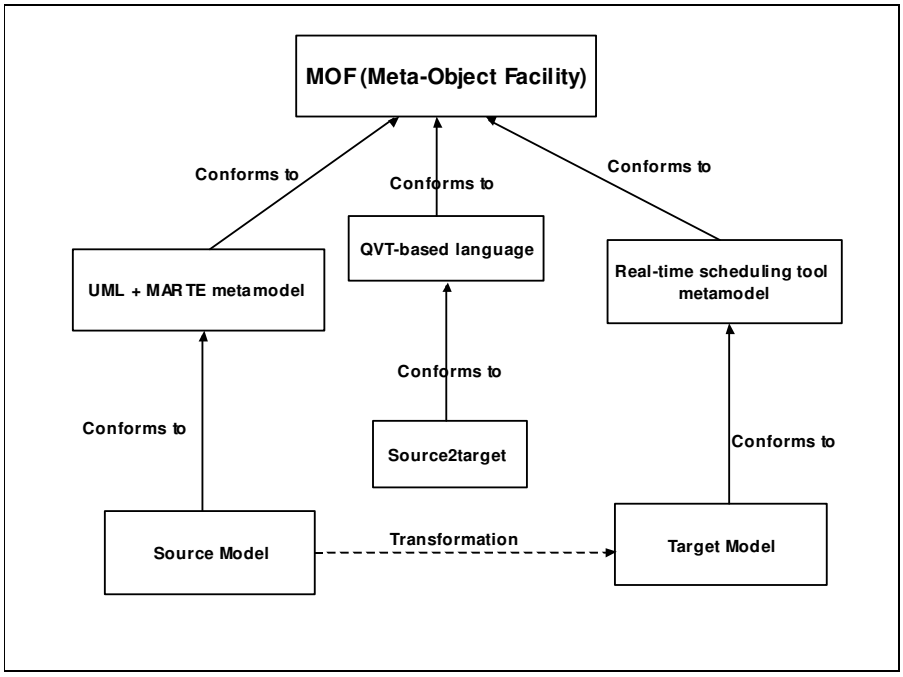

Figure 3.10 – Processus de transformation de modèles

3.4 Ordonnancement temps réel et outillage

Nous proposons pour l'analyse exacte des tâches dans une radio logicielle, une détermination de la faisabilité par simulation. En plus du modèle de tâche GMF non cyclique (introduit dans le chapitre précédent), on retrouve dans une radio logicielle des tâches périodiques pendant un intervalle de temps suite à l'arrivée d'un événement (notamment les tâches qui programment les fréquences porteuses). L'intervalle de temps et donc le nombre d'activations de la tâche pendant cet intervalle varient en fonction de la taille de la trame en entrée. En outre dans une radio logicielle, les tâches sont soumises à des contraintes de précédence. Cette contrainte n'a pas été prise en compte dans le chapitre précédent. Le type de tâches rencontré dans une radio logicielle se représente bien avec le modèle de tâches récurrentes non cycliques [76] introduit à la suite de la publication de nos travaux sur le modèle GMF non cyclique auprès de la communauté temps réel. Cependant, bien que la technique d'analyse d'ordonnancement pour les tâches récurrentes non cycliques présentée dans [76] a une complexité pseudo-polynomial pour les systèmes avec un taux d'utilisation strictement inférieur à 1, cette analyse ne peut pas être appliquée dans notre cas car, (1) nous considérons des tâches avec contraintes de précédences, (2) nous considérons un système avec ordonnanceur hybride (l'analyse dans [76] est pour des tâches indépendantes dans un système avec ordonnanceur EDF), (3) nous relâchons la condition qui consiste à avoir un taux d'utilisation strictement inférieur à 1 (autrement dit nous ne considérons pas cette hypothèse restrictive).

Pour la détermination de la faisabilité par simulation, un ordonnanceur hybride peut être représenté avec deux queues contenant les instances de tâches activées et une horloge représentant la progression du temps. La première queue simule l'ordonnanceur FP alors la seconde simule l'ordonnanceur EDF. Lorsqu'une tâche est activée, en raison de l'arrivée d'une trame (arrivée externe d'une trame, ou l'arrivée des données venant d'une autre tâche), la tâche est ajoutée dans la 1ère queue si elle doit être ordonnancée par l'ordonnanceur FP, sinon elle est ajoutée dans la 2ème queue. Typiquement, un quintuplet est ajouté dans la queue. Il est composé du temps d'exécution de la tâche, son échéance temporelle relative, la taille de buffer de données qu'elle utilise, de son identifiant pour les tâches EDF remplacé par la priorité pour les tâches FP, et de la liste des tâches suivantes. Une tâche n'active une autre tâche qu'à la fin de son exécution. Les instances de tâches dans la 1ère queue sont triées en fonction de la priorité de chacune d'elles. Dans la 2ème queue les tâches sont triées en fonction des échéances temporelles croissantes. Lorsque l'horloge avance, la stratégie FP consiste à décrémenter le temps d'exécution de la tâche en tête de la 1ère queue, ainsi que l'échéance temporelle de chacune des tâches présentes dans les deux queues. La stratégie EDF consiste à décrémenter le temps d'exécution de la tâche en tête dans la 2ème queue ainsi que l'échéance temporelle de chacune des tâches présentes dans cette queue. Dans les deux stratégies, la taille totale de buffer utilisée sur le processeur est mise à jour. Elle correspond à la somme des buffers utilisés par chaque tâche.

Lorsque le temps d'exécution d'une tâche devient plus grand que son échéance temporelle relative, ceci revient à dire qu'une échéance temporelle va être ratée. Lorsque le temps d'exécution devient égal à zéro ceci signifie que la tâche a fini son exécution. Ainsi, la tâche est retirée de la queue et s'il existe des tâches suivantes, elles sont activées. La stratégie EDF n'est exécutée que lorsque la 1ère queue est vide.

Les trames (telles qu'elles pourraient se présenter dans l'air – signaux radio émis/reçus par les utilisateurs) sont générées aléatoirement et distribuées dans le temps. Ensuite un calcul est effectué pour obtenir les dates d'arrivées des trames sur le processeur d'intérêt. Logiquement les trames à émettre sont reçues sur le processeur d'intérêt avant la date à laquelle la trame doit être dans l'air. Le concepteur doit donc spécifier la bonne durée d'anticipation en tenant compte de la latence globale du système. Il en va de même pour une trame à recevoir qui est reçue au niveau de processeur peu de temps après la réception effective de la trame sur l'antenne. La Figure 3.11 illustre ce calcul.

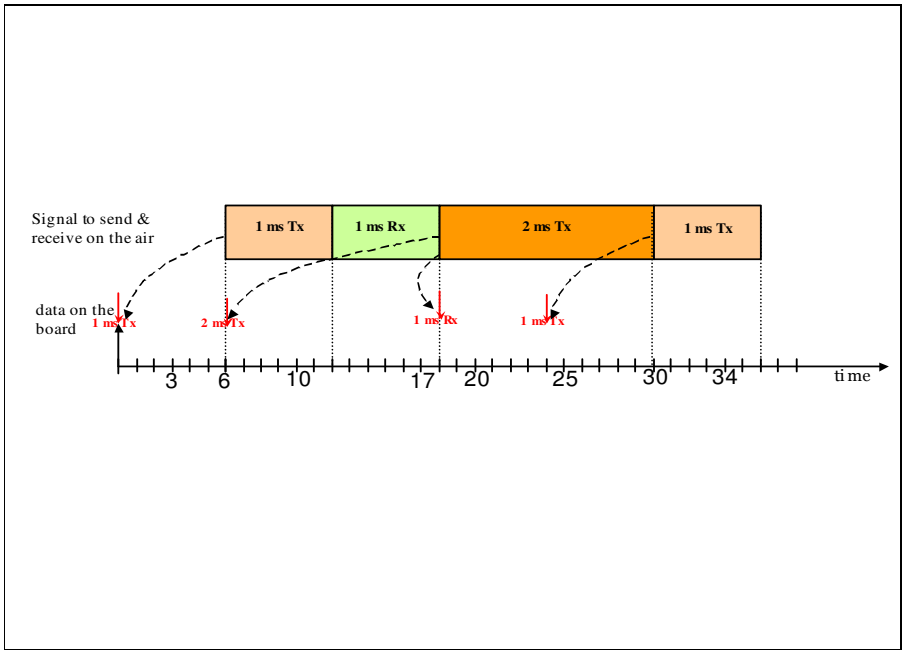

Figure 3.11 – Dates des trames dans l'air et dates des requêtes sur le processeur

La Figure 3.12 représente le pseudo-code de l'algorithme pour la détermination de la faisabilité. La fonction « activatetaskwaitingfortimer() » dans le pseudo-code permet d'activer certaines tâches à des dates précises. Ceci revient à dire que l'activation d'une tâche suite à la fin de l'exécution d'autres tâches ou suite à l'arrivée d'une trame, peut ne pas être immédiate. En effet cette activation peut être programmée à une date précise. C'est le cas de la tâche qui programme les fréquences porteuses. Cet algorithme a été implanté en langage C++, intégré sous forme d'outil et testé sur un ordinateur. L'outil est composé de trois parties. La 1ère partie est un générateur aléatoire de trames, qui génère aléatoirement les trames et calcule les dates correspondantes d'arrivée des trames sur le processeur d'intérêt. Cette partie de l'outil peut être remplacée par une génération de trames suivant une loi de Poisson, ou une loi normale.

N'ayant pas d'information sur la loi d'arrivée des trames nous avons considéré cette approche. La 2ème partie de l'outil correspond à l'analyse d'ordonnancement temps réel proprement dite. Il s'agit principalement de la mise en œuvre de l'algorithme de la Figure 3.12. Pendant la simulation, les états du processeur et la taille totale de la mémoire de données utilisée sont inscrits séparément dans deux fichiers. Lorsqu'une échéance temporelle est ratée la simulation s'arrête et on peut obtenir la séquence de trames qui nous a entraînée dans cette situation. La 3ème partie de l'outil est une simple interface graphique, que nous avons développée sous Visual C++ avec MFC (Microsoft Foundation Class). Le code implanté dans cette partie, lit les deux fichiers générés pendant la simulation et affiche tous les 512 unités de temps l'état du processeur et la taille totale de la mémoire de donnée occupée.
Notons que, la valeur 512, peut être modifiée.

FEASIBILITY DETERMINATION ALGORITHM

```
d = RandomFrameGeneration();
queue q1; // FP scheduler
queue q2; // EDF scheduler
time = 0;
while(time<=lastframearrivaltime)
        time++;
        for all frame f in d
                if(time==arrivaltime(d(f)))
                        activatetaskwaitingforexternalevent();
        endfor
        for each task τ_i in task set Δ
                if(time== releasetime( τ_i ))
                        activatetaskwaitingfortimer( τ_i );
        endfor
        if(Emptyqueue(q1) and Emptyqueue(q2))
                processorstate = idle;
                else
                if (deadlinemissed(q1) or deadlinemissed(q2))
                        processorstate = deadlinemissed;
                else
                        processorstate = running;
                endif
        endif
        if(Emptyqueue(q1))
                executeEDFstrategy(q2);
        else
                executeFPstrategy(q1, q2);
        endif
        for each task τ_i in dependent task set Γ
                if (completionoftaskpriorto( τ_i ))
                        activate( τ_i );
        endfor
endwhile
```

Figure 3.12 – Algorithme de détermination de la faisabilité temps réel

La Figure 3.13 représente les différentes parties de l'outil. Nous vous présentons l'expérimentation de cet outil sur un cas d'études dans le chapitre suivant.

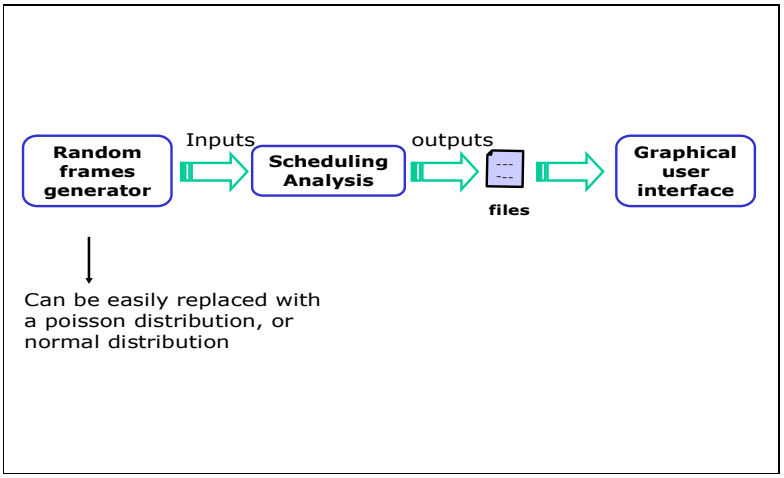

Figure 3.13 – Schéma de l'outil de vérification de l'ordonnançabilité

Ainsi, si la validation temporelle est satisfaisante, pour un certain nombre de trames générées aléatoirement, le système peut être considéré comme une bonne solution. Autrement, les concepteurs doivent revenir sur l'une des phases PIM, PDM ou PSM et changer certaines spécifications. Par exemple, le concepteur pourrait décider qu'un processeur plus performant est nécessaire, ou qu'il a besoin de plus de mémoire, ou que certaines activités doivent être implantées sous forme matérielle. Le concepteur pourrait également décider d'utiliser un algorithme plus simple afin de réduire la complexité en temps de calcul.

3.5 Conclusion

Ce chapitre a considéré et abordé plusieurs problèmes rencontrés lors de la conception de radio logicielle. Nous avons présenté dans ce chapitre des méthodes efficaces pour :

- modéliser une radio logicielle en intégrant des moyens de test d'ordonnancement temps réel. De plus nous prenons en compte le nouveau profil pour systèmes temps réel embarqué (MARTE).
- déterminer la faisabilité temps réel, l'occupation du processeur, et l'occupation de la mémoire, d'un ensemble de tâches dans une radio logicielle implantant des algorithmes de traitement du signal flexibles et s'exécutant sur un processeur en fonction d'une politique d'ordonnancement hybride.

Les méthodes présentées dans ce chapitre, fournissent une solution intéressante au problème qui consiste à développer un grand nombre d'algorithmes de traitement du signal sous forme entièrement logicielle dans un équipement radio logicielle. Les résultats décrits dans ce chapitre ont été présentés dans deux conférences phares de la communauté radio logicielle et ont fait l'objet de la soumission d'un article dans un

journal. Dans le chapitre suivant nous présentons une expérimentation de ces méthodes sur un cas d'études, qui démontre les avantages de notre méthodologie. Nous montrons également le niveau de précision de nos résultats en les comparant aux vrais résultats obtenus sur cible.

Chapitre 4

Cas d'études

Résumé : *Dans ce chapitre nous effectuons une expérimentation des méthodes proposées dans le chapitre précédent sur un projet afin de démontrer les avantages de notre approche. Le projet consiste en une implantation de la version multi-utilisateur de la technique de modulation OFDM sur une carte électronique contenant au moins un DSP. Nous montrons également dans ce chapitre que l'outil de simulation, développé pendant nos travaux, a un bon niveau de précision en comparant les résultats prédits par l'outil avec ceux réellement obtenus sur cible.*

4.1 Introduction

Dans l'optique d'illustrer notre méthodologie de conception, nous nous sommes appuyé sur un projet en cours de réalisation chez THALES Communications. L'objectif du projet est de réaliser une radio logicielle dont la couche physique est basée sur la version multi-utilisateur de la technique de modulation OFDM avec évasion de fréquence (saut de fréquence). La technique de modulation OFDM est utilisée dans plusieurs systèmes tel que les communications par courants porteurs, les systèmes de diffusion audio (DAB) et vidéo (TNT), les systèmes de réseaux sans fil (WIFI) etc. Une des raisons qui justifie cette utilisation est que l'OFDM utilise les sous porteuses orthogonales pour envoyer et recevoir plusieurs symboles de données en parallèle. L'un des avantages du saut de fréquence, notamment dans un contexte militaire, est que le signal est plus difficile à intercepter. Lorsque nous avons repris ce projet, l'évaluation de la faisabilité temps réel, avait été effectuée de manière empirique, i.e. basée sur l'expérience des ingénieurs. En effet l'évaluation consistait à estimer sur un fichier « Excel » la charge du processeur pendant la phase où certaines tâches effectuent des traitements d'émission alors que d'autres effectuent des traitements de réception. Contrairement, à cette approche nous avons utilisé notre méthodologie présentée dans le chapitre précédent. Notre objectif est d'évaluer le nombre de modules de traitement du signal qui peuvent être déployés sur un processeur DSP tout en garantissant le respect de leurs contraintes temporelles. Le résultat d'une telle évaluation fournit non seulement une garantie sur la faisabilité temps réel des modules de traitement du signal, mais permet aussi une implantation de modules sous forme entièrement logicielle comme visé dans l'approche radio logicielle. Pour réaliser notre évaluation, nous sommes partis des documents de conception déjà réalisés pour ce projet. La couche physique traite deux types de trames : les trames d'une longueur de 2,5 ms et les trames d'une longueur de 10 ms. Chaque trame est subdivisée en palier. Chaque palier est émis/reçu à une fréquence porteuse spécifique. Par conséquent

on saute sur plusieurs fréquences à l'intérieur d'une trame. Par exemple pour la trame de 2,5 ms à émettre, si on considère des paliers de 500µs, pendant la durée d'émission de la trame (2,5 ms) on va sauter toutes les 500 µs sur une fréquence porteuse parmi 5. L'émission et la réception en continu sur l'antenne font que les traitements du modem doivent respecter des contraintes temporelles strictes. C'est une radio logicielle, car plusieurs fonctions de traitement du signal sont implantées de manière logicielle. C'est par exemple le cas de la fonction qui contrôle la partie modem de l'équipement radio (i.e. la programmation et le séquencement des fréquences porteuses et de l'amplificateur de puissance). Ainsi l'équipement radio peut changer de bande de fréquence grâce à un changement activé par commande logicielle. L'équipement peut également passer en mode écoute/surveillance des communications radio toujours grâce à un changement logiciel. La Figure 4.1 présente les différents blocs fonctionnels mis en œuvre dans le modulateur et leurs relations de dépendances. Les premiers blocs (la partie correspondant aux traitements à l'échelle de la trame) sont exécutés une seule fois par trame. Les derniers blocs (la partie correspondant aux traitements à l'échelle du palier) sont exécutés plusieurs fois par trame. En effet le nombre de fois où le bloc est exécuté correspond au nombre de paliers dans la trame.

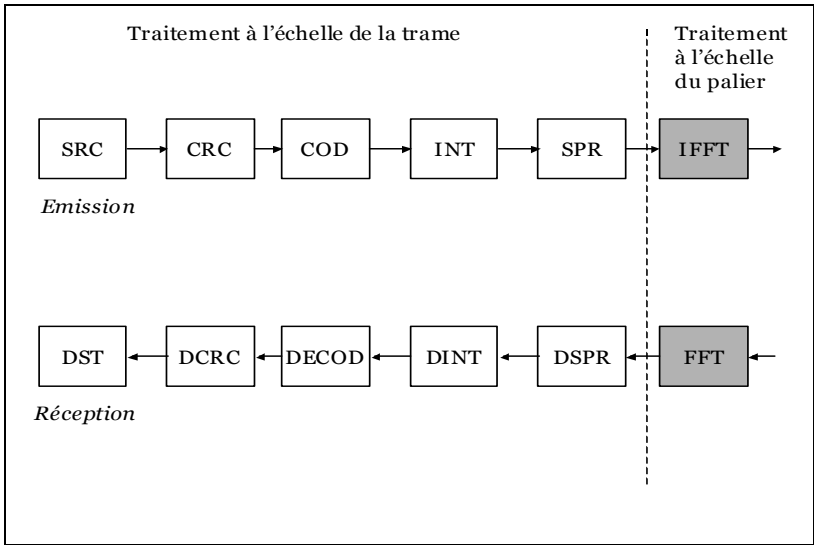

Figure 4.1 – Schéma fonctionnel de notre modulateur OFDM

En phase d'émission, les données sont d'abord complétées par un code de redondance cyclique (CRC) qui peut varier de 0 à 24 bits. Ensuite les données passent par l'étape du codage canal (COD - effectué par un turbo code). Il s'ensuit un entrelacement (INT) qui a pour effet de répartir aléatoirement les erreurs. L'étalement de spectre (SPR) est effectué avec un code d'embrouillage aléatoire. Enfin la modulation OFDM est effectuée grâce à l'inverse d'une transformée de fourrier rapide (IFFT) avant l'envoi des échantillons au transceiver pour les conversions numériques/analogiques, l'amplification du signal, etc.
En phase de réception, nous retrouvons les blocs inverses de ceux de la phase d'émission. Une fois la démodulation effectuée avec une transformée de fourrier rapide (FFT), le

désembrouillage des données est effectué grâce aux code d'embrouillage (DSPR). Les étapes inverses de l'émission se poursuivent par un désentrelacement (DINT), un décodage canal, puis une séparation du code de redondance cyclique, afin de récupérer les données transmises.
Toutes les phases de modélisation ont été effectuées avec l'outil « Rational Software Architecture » (version 7.5.4), en utilisant l'implantation du profil MARTE disponible sous licence libre.

4.2 Structure du projet dans l'outil de modélisation

Notre projet sur l'outil de modélisation est composé de deux modèles : un modèle pour la modélisation PIM appelé « Waveforms_PIM » et un modèle pour la modélisation PSM appelé « Waveforms_PSM ». Tel que le montre la Figure 4.2, nous proposons un modèle PIM composé de trois packages représentant respectivement l'architecture de l'application, le comportement de l'application, et les différents types des données plus les interfaces utilisées et réalisées par les ports. Dans le modèle PSM, nous avons définis 4 packages. Le package « Deployment » est composé des modules à implanter plus le diagramme qui montre sur quelle cible est implanté chaque module. Le package « PIM2PSM_Relation » est composé d'un diagramme qui fait la relation entre les modules PIM et les modules à implanter. Le package « Platform » décrit la plateforme, et le package « RealTimeProperties » contient toutes les contraintes temporelles.

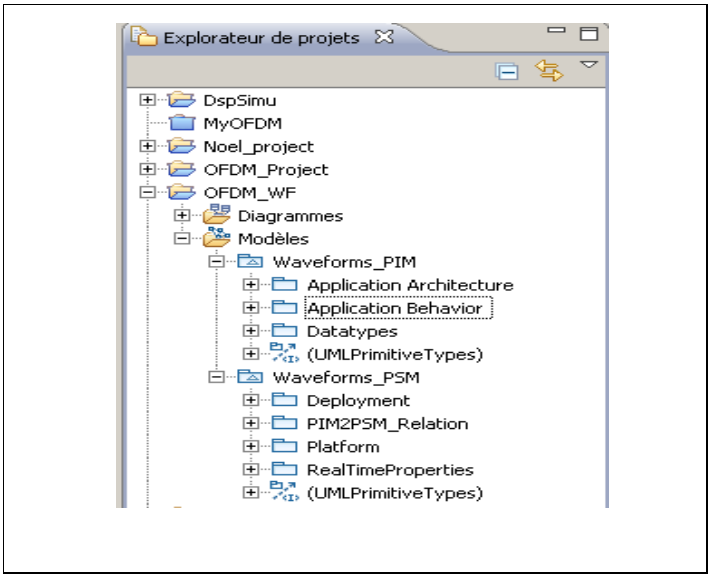

Figure 4.2 – Explorateur de projet

Comme mentionné dans le chapitre précédent la première phase de conception démarre par une modélisation PIM de l'application.

4.3 Modélisation PIM

La Figure 4.3 montre la représentation PIM du niveau en couche. Elle est composée de trois composants : une couche MAC, une couche PHY, et un module INFOSEC. Bien que nous ayons choisi de le représenter à ce niveau, le module INFOSEC n'est pas un composant du modèle OSI. Il s'agit d'une carte de chiffrement qui génère des clés permettant de sécuriser la transmission. Les noms au-dessus des liens entre les composants correspondent aux interfaces entre les différents composants. Pour chaque interface une liste de fonctions est définie. La Figure 4.4 montre la représentation PIM du niveau module de base. Elle est composée de 10 modules de base. Il s'agit des algorithmes de codage et de décodage, des modules pour l'étalement du spectre, des modules pour la modulation et la démodulation, d'un module de contrôle de la radio qui permet de contrôler, de programmer la chaîne radio et l'amplificateur de puissance. Ce module permet également d'obtenir la synchronisation entre les horloges de l'émetteur et du récepteur. Le module « Security » permet la demande des clés de chiffrement. Le module « WFProcessingCtl » calcule les métriques. Le module « Sequencer » dispache les différents types de données (données utiles, fréquences porteuses, clé de chiffrement).

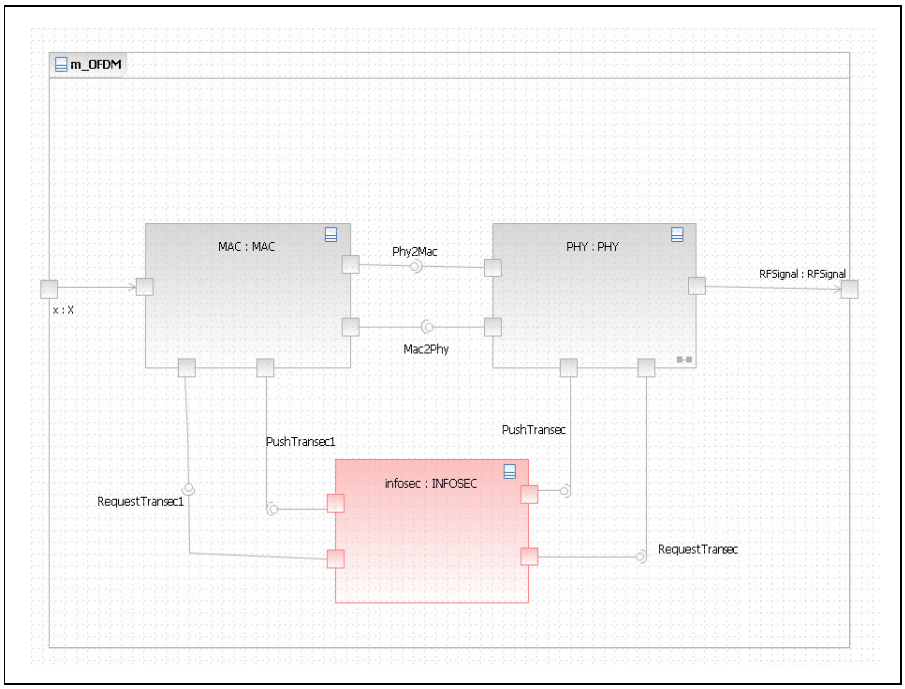

Figure 4.3 – Modélisation PIM (niveau en couche)

Un module de base ne peut pas être raisonnablement divisé en sous modules, par exemple, le module « Decoding » correspond à un turbo décodeur qui lui même est composé de deux décodeurs élémentaires interconnectés entre eux. Les itérations récurrentes entre les deux décodeurs empêchent une séparation des décodeurs élémentaires dans deux unités d'exécution différentes (ceci résulterait en une implantation moins efficace).

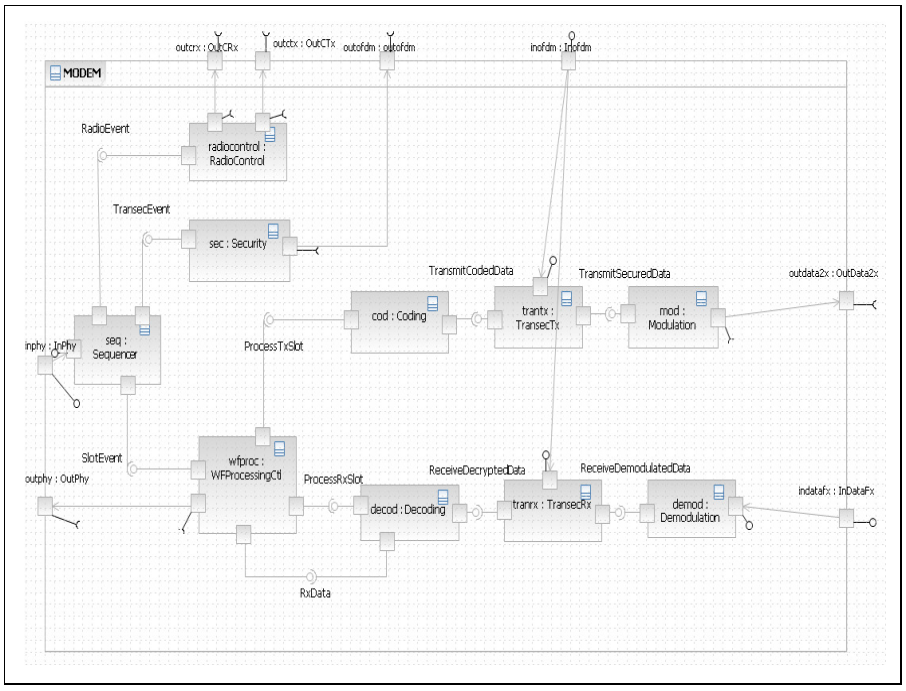

Figure 4.4 – Modélisation PIM (niveau module de base)

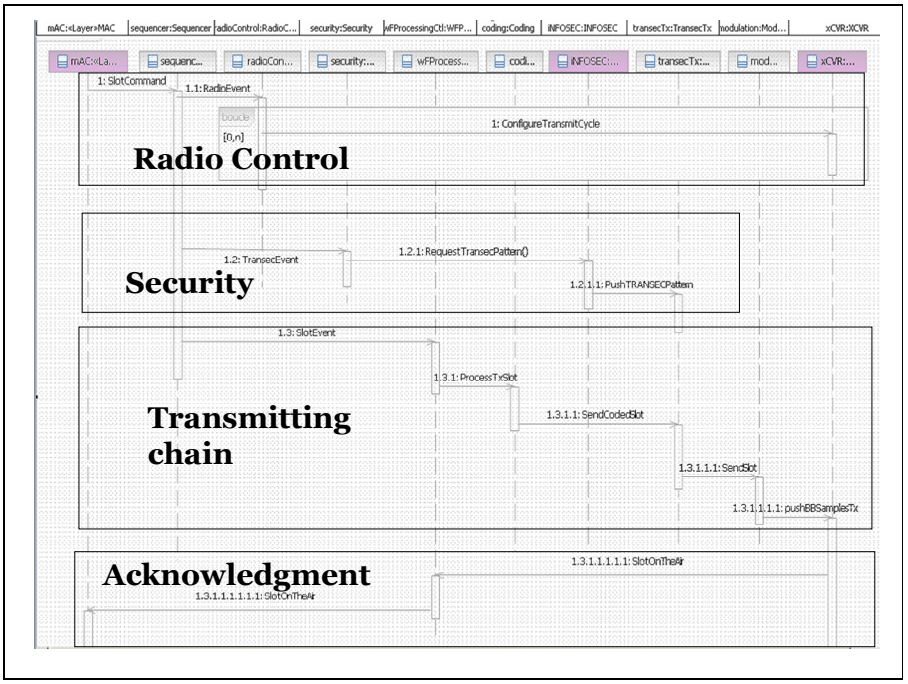

Figure 4.5 – Séquence d'une transmission

La Figure 4.5 présente un des diagrammes de séquence que nous avons réalisés. Ce diagramme représente le comportement des modules de la partie modem de la radio impliqués lors de la phase de transmission. Nous avons une première partie « Radio Control » qui effectue le séquencement et la programmation des fréquences porteuses. La deuxième partie « Security » permet de réaliser la demande de la clé de chiffrement, afin

de sécuriser les données. La troisième partie « Transmitting chain » correspond à la chaîne de traitement des données utiles jusqu'à l'envoi des échantillons au « transceiver ». La quatrième partie « Acquittal » est un acquittement correspondant au début de l'envoi de la trame dans l'air.

4.4 Modélisation PDM

La plateforme d'exécution de la forme d'onde dans le cadre de ce projet est composée d'un PowerQuicc, d'un DSP Texas et d'un FPGA Altera. Comme nous nous intéressons à la faisabilité temps réel sur le DSP, nous avons effectué notre expérimentation sur une carte d'évaluation DSP qui a les mêmes caractéristiques que le processeur DSP du projet. Il s'agit d'une carte DSP TMS320C6416 avec 1Ghz de fréquence, un cache L1 SRAM de 32Ko, une RAM L2 SRAM de 1Mo. Nous avons utilisé MicroC/OS-II comme système d'exploitation temps réel où nous avons développé (en langage C) au-dessus de l'ordonnanceur natif à priorité fixe, un ordonnanceur EDF. MicroC/OS-II permet de créer des tâches dont la priorité va de 1 à 61 (les priorités de 62 et 63 étant réservées pour la tâche de fond et la tâche de statistique). Le principe est que plus le nombre est faible plus la priorité est importante. Dans notre implantation, les tâches de priorité 1 à 10 sont gérées par l'ordonnanceur natif de MicroC/OS-II pendant que les tâches de 11 à 61 sont gérées par notre ordonnanceur EDF. Autrement dit lorsqu'une de ces tâches doit être exécutée, les lignes de code que nous avons rajoutées, permettent que la tâche qui a l'échéance temporelle la plus proche soit exécutée. L'ordonnanceur EDF gère donc conceptuellement les tâches de plus basses priorités. Notre modification de MicroC/OS-II coûte, dans le pire cas, 10 µs supplémentaire à chaque activation d'une tâche EDF, pour 51 tâches EDF actives. Pour 20 tâches actives, notre modification ne coûte que 4 µs supplémentaires. Nous précisons que notre modification ne rajoute pas de surcoût lors de l'activation des tâches à priorité fixe. Le changement de contexte prend 1 µs.
Plusieurs de ces paramètres ont été renseignés dans le modèle de la plateforme tel que spécifié dans la section 3.3.2. A titre d'exemple la Figure 4.6 illustre comment nous renseignons la valeur de la mémoire L2 SRAM dans le modèle.

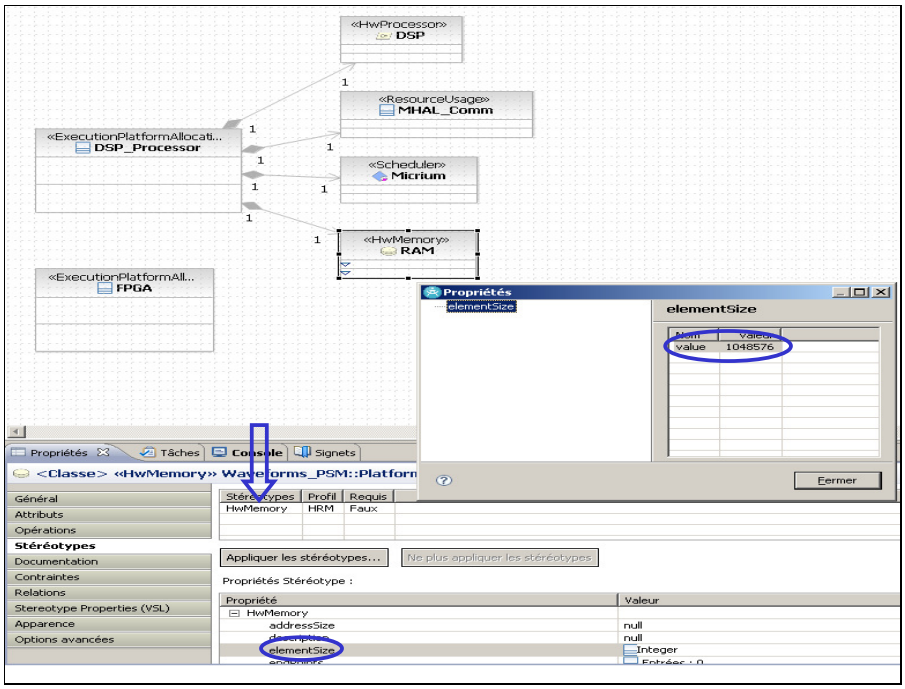

Figure 4.6 – Valeur de la mémoire L2 SRAM dans le modèle

4.5 Modélisation PSM

La Figure 4.7 présente les relations entre les modules PIM et les modules PSM.

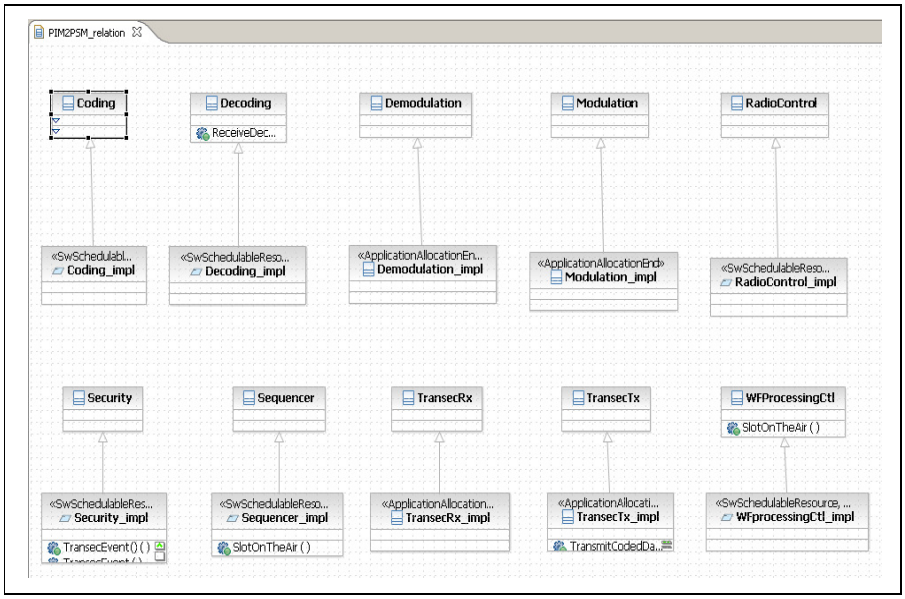

Figure 4.7 – Relation entre les modules PIM et les modules PSM

Dans le cadre du projet, quatre modules ont été implantés sur DSP : « Sequencer_impl », « Radiocontrol_impl », « Security_impl », « WFProcessingCtl_impl ».
Pour notre expérimentation nous avons considéré sur le DSP en plus de ces 4 modules, les modules « Coding » et « Decoding » qui correspondent respectivement à un turbo codeur 1/3 et un turbo décodeur. Dans le projet ces modules ont été implantés sur FPGA car au démarrage du projet il était considéré risqué de déployer ces modules sur le DSP à cause des problèmes d'ordonnancement temps réel. La Figure 4.8 présente notre allocation sur un diagramme de classe.

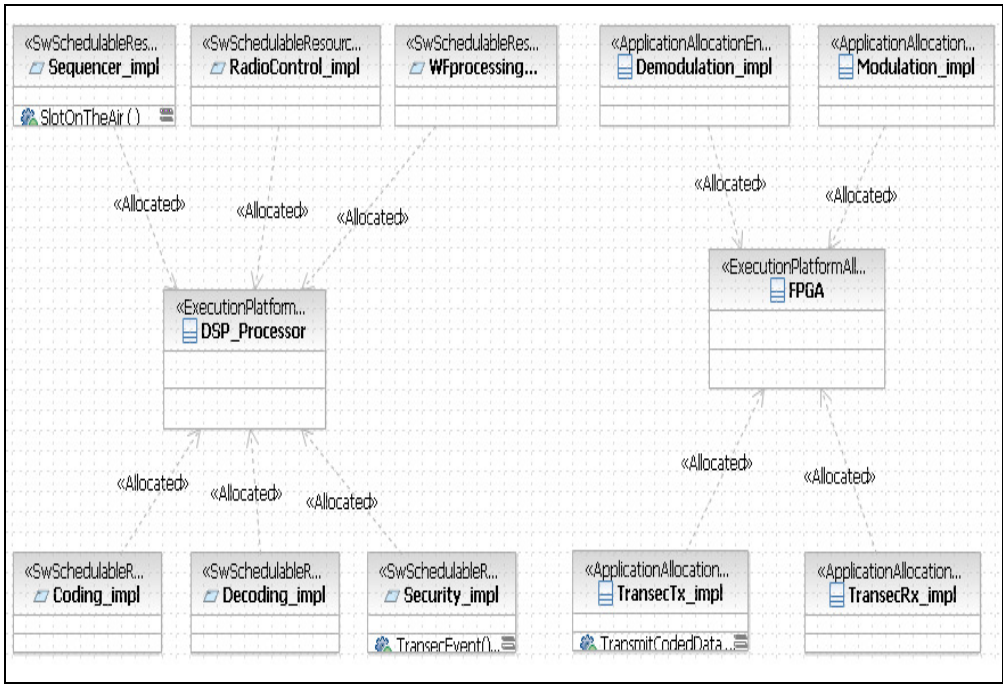

Figure 4.8 – Approche graphique d'allocation de modules

Les échéances temporelles des tâches ont été calculées suivant l'algorithme décrit en section 3.3.3 dans le chapitre précédent. Il faut rentrer dans l'outil implanté, le graphe correspondant à la chaîne radio (émission et réception), puis les temps d'exécution de chaque module, les dates d'arrivées pour chaque type de trame et les échéances temporelles absolues correspondantes. On doit également spécifier le processeur où chaque module est déployé car comme nous l'avons mentionné en section 3.3.3, la date de démarrage au plus tôt dépend dans certains cas du déploiement. La Figure 4.9 est une capture d'écran des résultats de l'outil de calcul des échéances temporelles pour un exemple composé de deux chaînes (émission plus réception) gérant deux types de trames. Chaque module a donc deux échéances temporelles. Par exemple les échéances temporelles de « Sequencer » valent respectivement 726 et 452 tandis que celles de « Security » valent 1322 et 1644.

Le Tableau 4.1 présente les paramètres temporels des tâches que nous avons renseignés dans le modèle. Toutes les valeurs sont en microseconde. Les tâches « Sequencer_impl » et « RadioControl_impl » sont gérées avec l'ordonnanceur FP (nous avons considéré « Sequencer_impl » plus prioritaire que « RadioControl_impl ») alors que les autres tâches sont gérées avec l'ordonnanceur EDF.

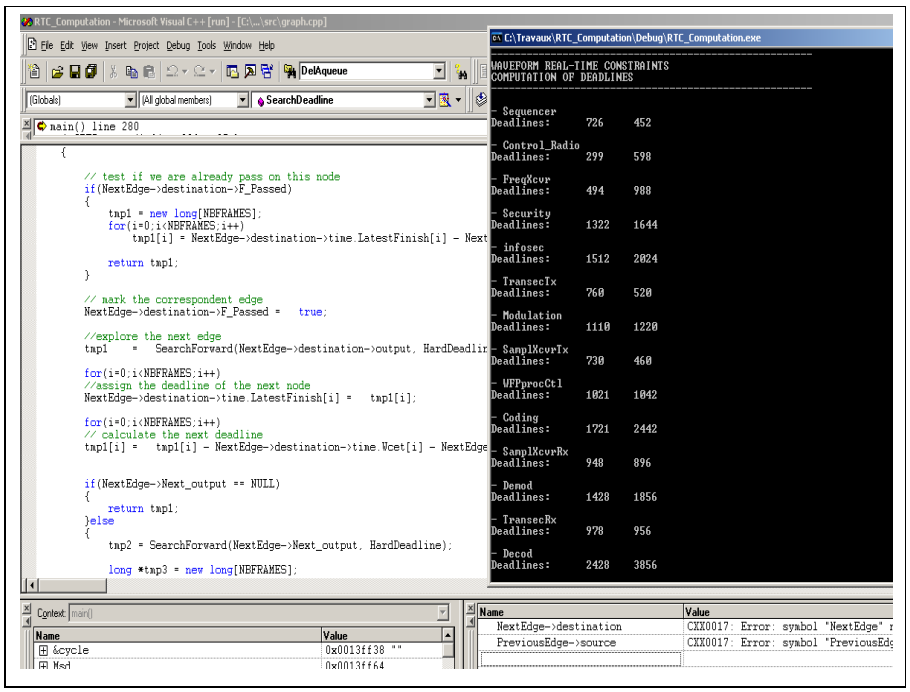

Figure 4.9 – Outil de calcul des échéances temporelles

Tâche	C_i^j	D_i^j	η_i^j	Trame	Tâches suivantes implantées sur DSP
RadioControl_impl	5	500	5	2500	
	10	500	10	10000	
Sequencer_impl	5	726	1	2500	ControlRadio_impl, Security_impl, WFProcessing
	10	1452	1	10000	
Security_impl	10	1322	1	2500	
	40	5288	1	10000	
Coding_impl	1000	1721	1	2500	
	4000	6884	1	10000	
Decoding_impl	1500	2428	1	2500	

	6000	9712	1	10000	
WFProcessing	50	1024	1	2500	Coding_impl ou Decoding_impl (déterminé à l'exécution)
	100	2048	1	10000	

Tableau 4.1 – 1er exemple de tâches

Les paramètres C_i^j et D_i^j représentent respectivement le temps d'exécution et l'échéance temporelle de la j-ième trame de la tâche τ_i. Le paramètre η_i^j représente le nombre de fois que la tâche est activée suite à l'arrivée de la j-ième trame. La pseudo-période de la tâche « RadioControl_impl » vaut 500µs. Le temps de garde entre deux arrivées successives de trames pour une tâche est égale à $\eta_i^j * D_i^j$. A titre d'exemple, la Figure 4.10 illustre comment nous renseignons les échéances temporelles dans le modèle. Une fois toutes les contraintes temporelles renseignées, nous effectuons une transformation de modèle en utilisant l'outil de développement ATL IDE, permettant d'implanter nos règles en langage ATL (Atlas Transformation Language).

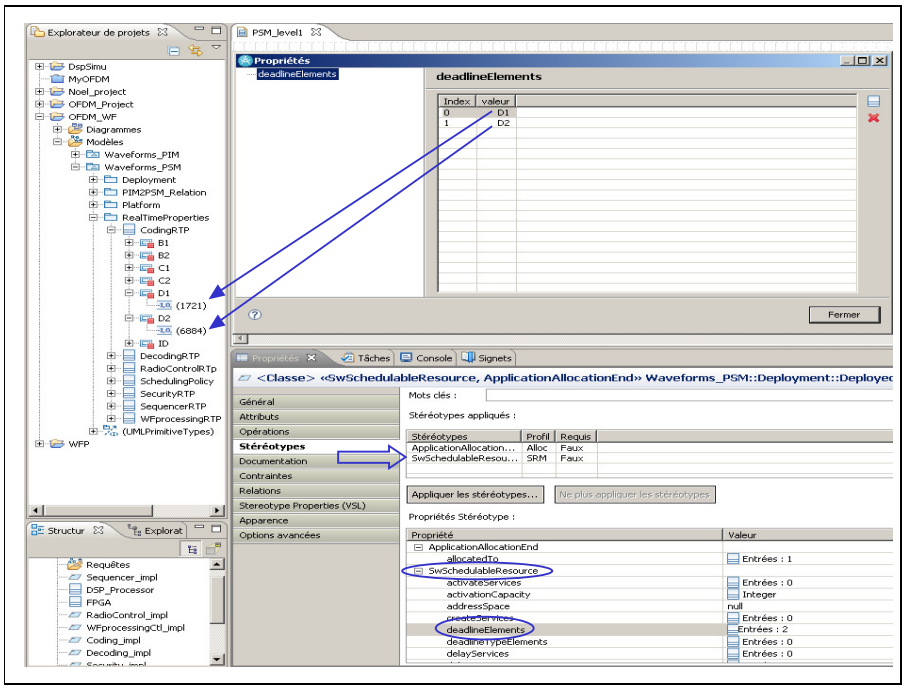

Figure 4.10 – Les échéances temporelles dans le modèle

4.6 Transformation de modèles

Le langage ATL est basé sur QVT. L'outil ATL IDE, permet de spécifier des règles de transformations en langage ATL et d'obtenir (après exécution) un modèle conforme au métamodèle (à la sémantique) de notre outil d'analyse d'ordonnancement temps réel. Nous avons réalisé le métamodèle de notre outil avec le langage UML, qui est conforme à

MOF et donc par transition notre métamodèle est conforme à MOF. La Figure 4.11 présente le métamodèle que nous avons réalisé pour notre outil.

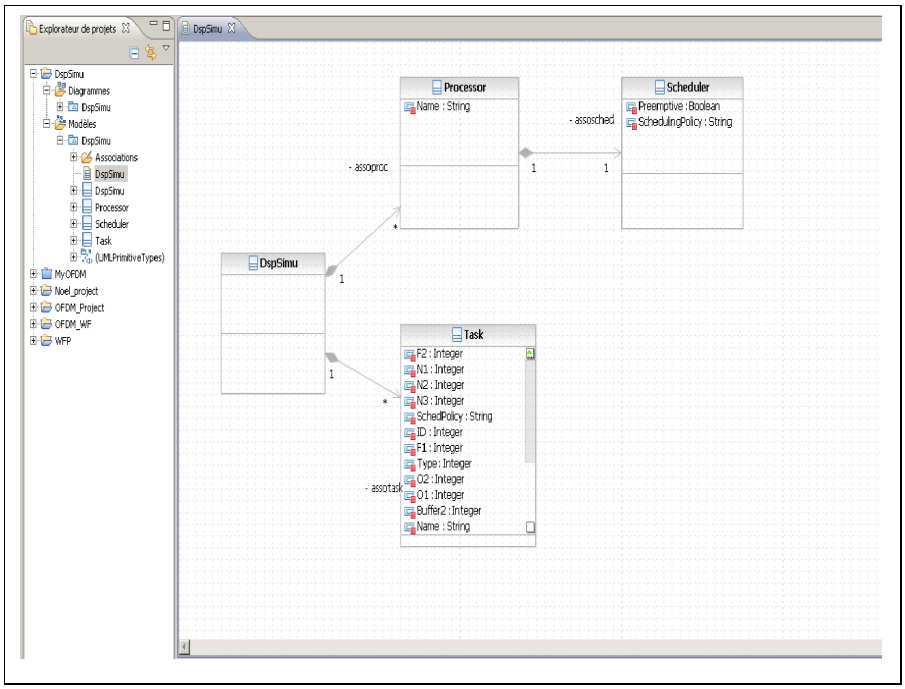

Figure 4.11 – Métamodèle de notre l'outil de simulation

Dans la Figure 4.11, la description que nous faisons de notre outil est qu'il est composé d'un ou plusieurs processeurs, que chaque processeur possède un et un seul ordonnanceur, et que l'outil est composé d'un ensemble de tâches. Chaque tâche possède plusieurs attributs. Il s'agit de la liste des temps d'exécution, la liste des échéances temporelles, la liste des tâches suivantes, le type de la tâche, la pseudo-période (si elle en a) etc.

Les règles de transformations consistent à « mapper » les concepts MARTE vers les concepts de notre outil d'analyse.

La Figure 4.12 représente une capture d'écran des règles de transformation du Tableau 3.1 (présenté au chapitre précédent) que nous avons développées en langage ATL. Le fichier contenant les règles en langage ATL porte le nom « MODEL2DSPSIMU.atl » dans la Figure 4.12. Les fichiers commençant par « MARTE » proviennent du métamodèle de MARTE et permettent d'attribuer une sémantique aux éléments de MARTE dans le modèle. Les fichiers « Default.profile.uml », « Deployment.profile.uml » et « ProfileBase.profile.uml » proviennent de l'outil de modélisation « Rational Software Architecture ». Nous précisons que le format « .uml » correspond au format texte du langage UML, il est basé sur le langage XMI. Le fichier « Waveforms_PIM.uml » correspond au modèle PIM de notre application radio logicielle, tandis que le fichier « Waveforms_PSM.uml » correspond au modèle PSM. Le fichier « DspSimu.ecore » est le métamodèle de notre outil de simulation au format « .ecore », alors que le fichier « Out4DspSimu.ecore » est le fichier que nous avons obtenu après la transformation. Ce

fichier est conforme au métamodèle de notre outil. Les fichiers de format « .ecore » sont conformes au métamodèle Ecore de l'« Eclipse Modeling Framework » (EMF). Ce métamodèle permet de décrire de manière textuelle les modèles et les supports d'exécution pour les modèles, utilisant le format XMI et une API (Application Programming Interface) efficace pour manipuler les objets EMF.

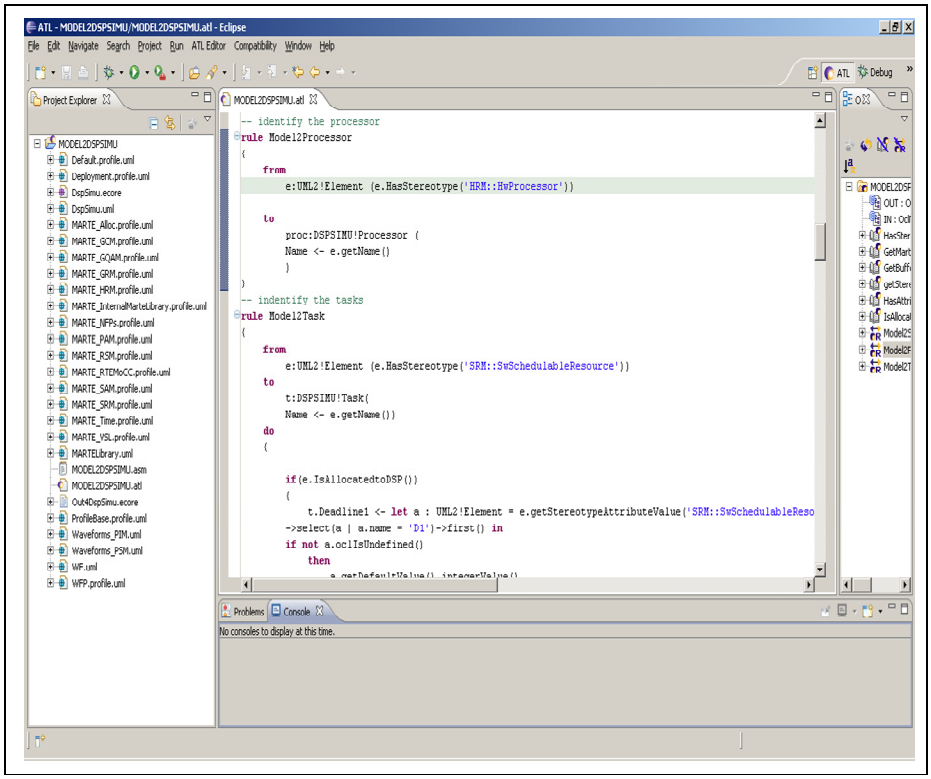

Figure 4.12 – Règle de transformation en ATL

La Figure 4.13 résume le processus de transformation dans l'outil ATL IDE.

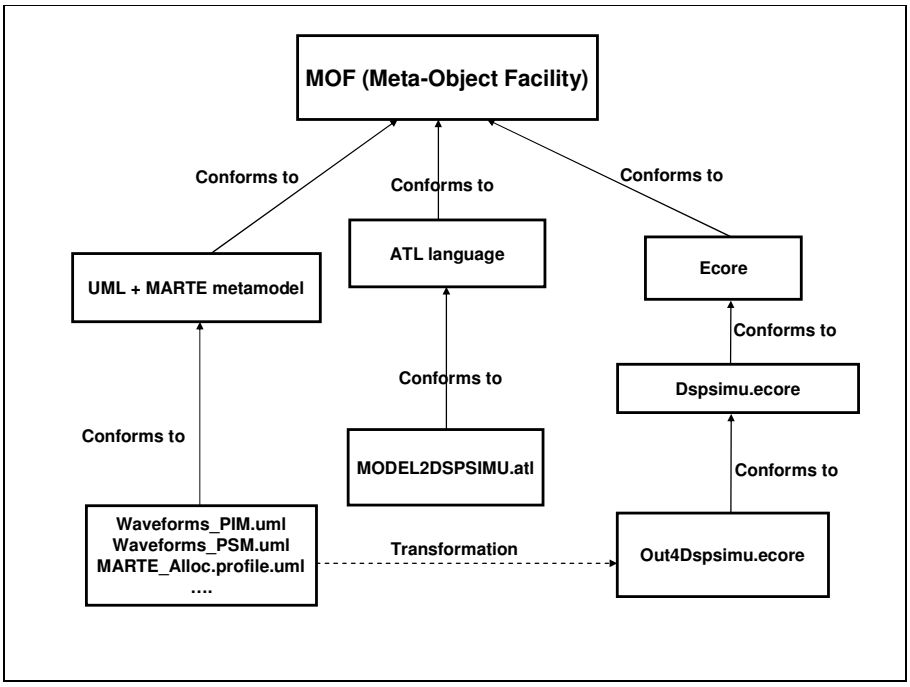

Figure 4.13 – Transformation de modèle avec l'outil ATL

La Figure 4.14 présente un extrait du fichier « Waveforms_PSM.uml » (à gauche) et le fichier obtenu (« Out4Dspsimu.ecore ») après la transformation (à droite).

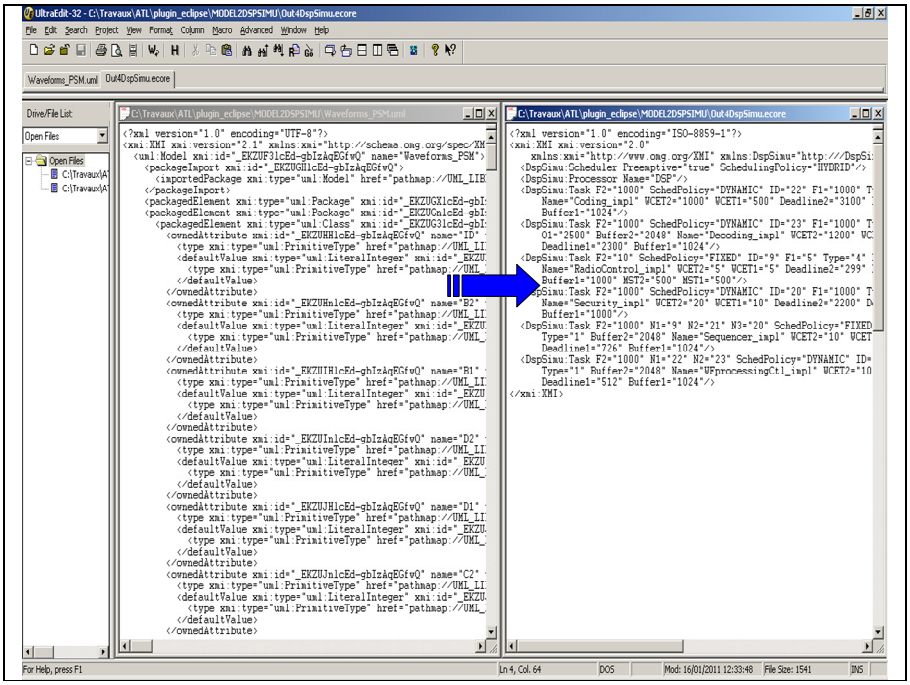

Figure 4.14 – Comparaison entre le fichier issu du modèle et le fichier généré

4.7 Vérification d'ordonnancement temps réel

Après la transformation de modèles, nous utilisons l'outil que nous avons développé durant cette thèse afin de valider l'ordonnancement temps réel des tâches sur le DSP. L'outil lit le fichier généré par ATL (pour obtenir les tâches et leurs contraintes temporelles), génère aléatoirement 20000 trames tel qu'elles pourraient se présenter dans l'air, et calcule les dates d'arrivées correspondantes des trames sur le processeur. Evidemment le nombre de trames peut être modifié. Les trames générées correspondent à un mélange de trames de 2,5 ms et 10 ms (en émission et en réception). Une horloge avance, et les tâches sont activées suite à l'arrivée des trames, suite à la fin de l'exécution d'autres tâches, ou suite à l'expiration d'un « timer ». Durant la simulation l'état du processeur et la taille totale de la mémoire de donnée utilisée sont inscrits dans deux fichiers distincts. Les deux fichiers sont ensuite lus et les informations qu'ils contiennent sont affichées sur une interface graphique. La Figure 4.15 représente l'interface graphique de l'outil que nous avons développé en langage C++ avec MFC durant ces travaux. En haut à gauche, nous avons la liste des tâches récupérés du fichier généré par ATL. En haut à droite, nous avons le résultat du test de faisabilité du chapitre 2. Il ne s'applique dans ce cas car nous n'avons plus le même modèle de tâche. En bas à gauche, nous avons un graphique à barres qui représente l'occupation du processeur toutes les 512 unités de temps. En bas à droite un graphique en lignes représente l'occupation totale de la mémoire de donnée pendant les mêmes 512 unités de temps.
Nous considérons 3 états du processeur : l'état zéro signifie que le processeur est libre, l'état 1 signifie que le processeur est occupé, l'état 2 signifie qu'une échéance temporelle va être ratée.

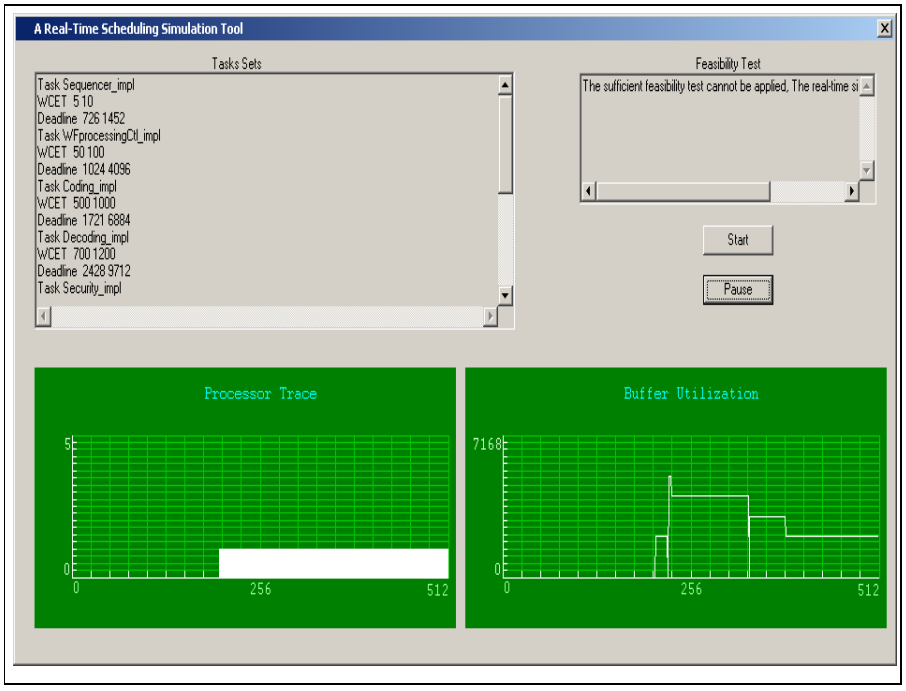

Figure 4.15 – Interface graphique de l'outil d'ordonnancement temps réel

Avec ces 6 modules sur le DSP, et les 20000 trames générées, aucune tâche n'a raté une ses échéances temporelles. Nous avons donc rajoutés deux tâches sur le DSP. Il s'agit des tâches « TransecTx_impl » et « TransecRx_impl » qui effectuent l'étalement de spectre. Nous obtenons la configuration présentés dans le Tableau 4.2. Les deux tâches rajoutées sont gérées avec l'ordonnanceur EDF.

Tâche	C_i^j	D_i^j	η_i^j	Trame	Tâches suivantes implantées sur DSP
RadioControl_impl	5	299	5	2500	
	10	299	10	10000	
Sequencer_impl	5	726	1	2500	ControlRadio_impl, Security_impl, WFProcessing
	10	1452	1	10000	
Security_impl	10	1322	1	2500	
	40	5288	1	10000	
Coding_impl	1000	1721	1	2500	TransecTx_impl
	4000	6884	1	10000	
Decoding_impl	1500	2428	1	2500	
	6000	9712	1	10000	
WFProcessing	50	1024	1	2500	Coding_impl ou TransecRx_impl (déterminé à l'exécution)
	100	2048	1	10000	
TransecTx_impl	50	100	1	2500	
	200	400	1	10000	
TransecRx_impl	50	100	1	2500	Decoding_impl
	200	400	1	10000	

Tableau 4.2 – 2ème exemple de tâches

Pour cette configuration également, aucune tâche n'a raté une de ses échéances temporelles.

Nous avons effectué un autre test avec le système du Tableau 4.3 composé de 6 tâches. Il s'agit des mêmes tâches du Tableau 4.1 où certains paramètres temporels ont été changés.

Tâche	C_i^j	D_i^j	η_i^j	Trame	Tâches suivantes implantées sur DSP
RadioControl_impl	5	299	5	2500	
	10	299	10	10000	
Sequencer_impl	5	726	1	2500	ControlRadio_impl, Security_impl, WFProcessing
	10	1452	1	10000	
Security_impl	10	1322	1	2500	
	40	5288	1	10000	
Coding_impl	1000	1721	1	2500	
	4000	6884	1	10000	
Decoding_impl	1500	2428	1	2500	
	6000	9712	1	10000	
WFProcessing	300	1024	1	2500	Coding_impl ou Decoding_impl (déterminé à l'exécution)
	300	2048	1	10000	

Tableau 4.3 – 3ème exemple de tâches

Avec cette configuration, la simulation s'arrête à l'instant t=28532, car la tâche « coding_impl » implantant le turbo codeur va rater son échéance temporelle. En effet, à cet instant l'outil nous indique que son temps d'exécution restant est égal à 1000 alors que son échéance temporelle relative est égale à 998. Il nous indique également qu'à cet instant deux tâches sont actives (« coding_impl » et « decoding_impl ») et que la tâche « decoding_impl » possède le processeur. Son temps d'exécution restant est égal à 561 et son échéance temporelle relative est égale 993.

Ces tests montrent combien peuvent être utiles en phase de conception d'une radio logicielle, la méthodologie de conception et l'outillage que nous proposons dans cette thèse. Les concepteurs peuvent ainsi mieux dimensionner leur radio logicielle et s'orienter très tôt vers le meilleur choix d'implantation. C'est tout le bénéfice d'une approche de conception de haut niveau. Obtenir rapidement des résultats non fonctionnels sans attendre l'exécution de toute l'application (et donc son développement sur la plate-forme finale). Nous démontrons à la section suivante que les résultats prédits par notre outil de simulation sont proches de ceux obtenus effectivement sur cible.

4.8 Précision de nos résultats

Pour mesurer le niveau de précision des résultats fournis par notre outil de simulation, nous avons comparé ses résultats à ceux obtenus sur une vraie cible DSP. Nous avons considéré l'ensemble des tâches du Tableau 4.3.
Comme, l'implantation de l'application globale nous aurait demandé beaucoup de temps, nous avons émulé certains modules (i.e. les modules déployés sur PowerQuicc et sur FPGA lors du « mapping ») toujours sur le même processeur DSP. Ainsi plusieurs fonctions ont été développées (en langage C) sous forme de tâches sur le DSP TMS320C6416 de Texas Instruments. Nous avons utilisé une même séquence de 40 trames comme données d'entrées de l'outil de simulation et de l'application sur DSP. Nous choisissons de représenter une trame par sa durée (en microseconde), suivi du mot Tx (pour les trames à émettre) ou Rx (pour les trames à recevoir). Voici la séquence de trames que nous avons considérée :

2500Tx2500Tx10000Rx10000Tx10000Rx2500Rx2500Rx2500Rx10000Tx2500Tx10000Rx2500Tx10000Tx10000Tx10000Rx2500Tx2500Tx10000Tx2500Rx10000Tx2500Tx2500Tx10000Rx10000Tx2500Tx2500Rx2500Rx10000Rx2500Tx10000Tx2500Rx10000Tx2500Tx10000Tx10000Rx2500Tx2500Rx2500Rx10000Rx10000Rx.

Après l'exécution, nous avons relevé les 100 premiers changements de contexte sur l'application réelle sur DSP, puis ceux prédits par notre outil de simulation. Nous les avons ensuite comparé sur un même graphique. Dans le graphique chaque point représente la tâche qui possède le processeur juste avant que le changement de contexte ne soit effectué. Des deux cotés (i.e. sur DSP et sur l'outil de simulation) nous ne relevons pas les tâches qui émulent les autres parties du système (i.e. les modules qui devraient être déployées sur PowerQuicc et sur FPGA). Le Tableau 4.4 présente l'identifiant de chaque tâche.

Tâche	Id
Sequencer_impl	1
RadioControl_impl	2
WFprocessingctl_impl	3
Security_impl	4
Coding_impl	5
Decoding_impl	6

Tableau 4.4– Identifiants des tâches

La Figure 4.16 montre la superposition des deux résultats. Nous remarquons d'abord une correspondance quasi parfaite entre les activations simulées et celles qui ont effectivement lieu dans la réalité. Seuls quelques décalages sans conséquences sont à

relever, ainsi qu'une légère différence. En effet tel que nous pouvons le voir sur la Figure 4.16, pendant les 20 premiers changements de contextes, la tâche « Decoding_impl » (tâche numéro 6) obtient une fois le processeur sur le DSP alors qu'elle ne l'obtient pas sur notre outil de simulation.

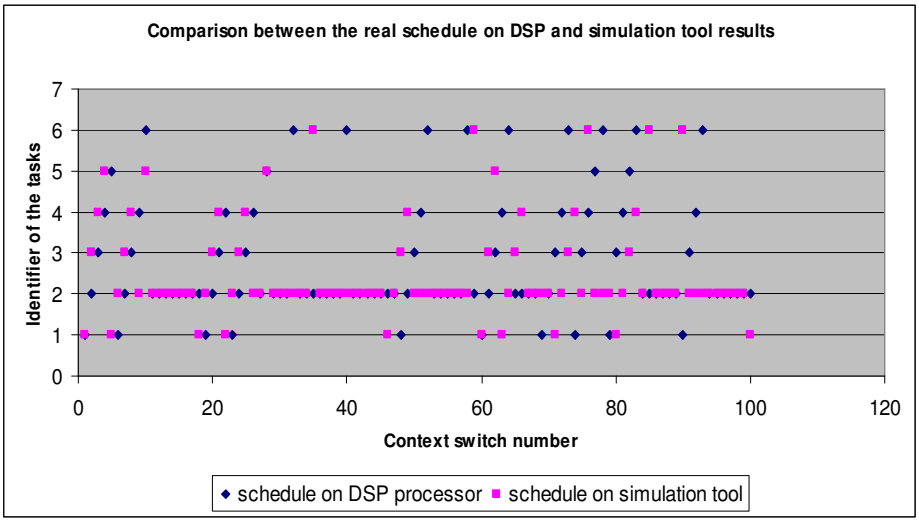

Figure 4.16 – Résultats prédits par l'outil et ceux obtenus sur cible

Ceci est principalement dû à la fonction de démodulation qui précède la fonction de décodage. Sur le DSP, la tâche de décodage reçoit les paramètres de décodage et est endormie pendant un intervalle de temps correspondant au temps de la démodulation. Ainsi sur le DSP la tâche obtient d'abord le processeur ensuite s'endort avant de se réveiller plus tard pour effectuer le décodage. Ce qui n'est pas le cas dans l'outil de simulation où le réveil de la tâche de décodage est directement programmé au bon moment. C'est pour cette raison que nous avons cette première différence. D'ailleurs c'est ce qui explique que l'on ait dans le graphique exactement deux fois plus de points de la tâche de décodage sur le DSP que sur notre outil de simulation.

Nous avons le même nombre de points pour la tâche de codage (tâche numéro 5) sur DSP comme sur notre outil de simulation (le deuxième point pour le DSP est complètement caché par le troisième point de l'outil de simulation). Aussi, la raison pour laquelle tous les points ne sont pas complètement superposés les uns aux autres vient du fait que dans le DSP, la gestion des interruptions matérielles retarde parfois l'exécution des tâches EDF (qui ont conceptuellement les priorités les plus faibles) jusqu'à l'activation des tâches FP.

En résumé, en fonction du niveau de granularité des informations temporelles renseignées dans notre outil (rajout du temps de gestion des interruptions matérielles), l'ordonnancement effectué sur notre outil est bien proche de l'ordonnancement effectué sur la cible. Par conséquent du point de vue temporel si nous ratons une échéance temporelle sur cible nous la raterons également sur notre outil de simulation, et inversement.

Ainsi l'outil permet bien au concepteur de prédire et valider le comportement temps réel du système conçu ce qui était l'objectif de la thèse : un outillage permettant de représenter

et capturer les paramètres temporelles d'une part, et d'exécuter un test de validation d'ordonnançabilité d'autre part.

4.9 Conclusion

Un challenge important lors de la conception de radio logicielle est l'ordonnancement temps réel sur le processeur DSP. Ce qui consiste à vérifier très tôt en phase de conception que toutes les tâches respecteront leurs échéances temporelles dans tous les scénarios possibles.

Dans ce chapitre nous avons présenté :

- une expérimentation qui démontre les avantages de notre méthodologie de conception de radio logicielle,
- une comparaison des résultats prédits par notre outil par rapport à ceux obtenus sur cible, qui démontre la bonne précision de nos résultats de simulation.

Ce travail permet clairement d'accélérer le processus de conception de radio logicielle car le concepteur anticipe mieux les problèmes d'ordonnancement temps réel, ce qui réduit les problèmes rencontrés en phase d'intégration sur cible. Notre outil peut également servir comme moyen de certification logicielle puisqu'il permet de produire une garantie de l'ordonnancement temps réel.

Conclusion et perspectives

Le besoin croissant de supporter plusieurs standards élargit le nombre de composants logiciels qui s'exécuteront simultanément sur le même processeur dans les futurs équipements radio. Cependant une radio logicielle est soumise à des contraintes temporelles strictes. C'est pourquoi, pour pouvoir bénéficier d'une radio flexible intégrant les différents standards radio existants, les concepteurs doivent pouvoir disposer de méthodologies de conception et d'outils leurs permettant de valider l'ordonnancement temps réel sur le processeur de traitement du signal numérique.
Tout au long de ce document, nous avons proposé des solutions efficaces au problème qui consiste à vérifier l'ordonnancement temps réel sur un processeur dans une radio logicielle. Les solutions présentées consistent en :

(1) Un nouveau modèle de tâches (le modèle de tâche multi-trames généralisés non cycliques). Ce modèle caractérise mieux le comportement des tâches dans une radio logicielle car il ne considère pas un motif spécifique pour l'activation des tâches. Ainsi dans notre modèle, l'ordre d'arrivée des trames est aléatoire ce qui correspond bien à l'arrivée des trames en mode rafale (Burst) dans une radio logicielle. Nous avons présenté pour ce modèle de tâche une nouvelle formule pour le calcul du temps de réponse de ces tâches ordonnancées avec EDF. De cette formule nous avons présenté un nouveau test de faisabilité suffisant à complexité polynomiale.

(2) Un algorithme efficace, pour la détermination exacte de la faisabilité temps réel d'un ensemble de tâches multi-trames généralisés non cycliques ordonnancés avec EDF. Cet algorithme parcourt toutes les exécutions possibles en parallèle en fusionnant les séquences d'exécutions identiques.

(3) Une méthodologie de conception IDM permettant l'analyse d'ordonnancement temps réel des tâches dans une radio logicielle. Nous avons présenté un ensemble d'étapes à suivre pendant les différentes phases de modélisation, une méthodologie pour calculer les contraintes temporelles, les différents éléments du standard MARTE à utiliser, des règles de transformations de modèles, et l'utilisation d'un ordonnanceur hybride pour les tâches implantant des algorithmes de traitement du signal flexibles dans une radio logicielle. Toutes ces méthodes ont été expérimentées sur un modulateur OFDM afin de démontrer les avantages de notre flot de conception.

(4) Une approche pour la vérification de l'ordonnancement temps réel des tâches implantant des algorithmes de traitement du signal flexibles et s'exécutant sur un processeur en fonction d'une politique d'ordonnancement hybride. Cette approche consiste à simuler le comportement temporel des tâches s'exécutant sur un processeur dans une radio logicielle en fonction des différentes possibilités de

trames (dans l'air) à émettre et à recevoir. Nous avons implanté cette approche sous forme d'un outil de simulation et avons démontré la bonne précision de nos résultats en les comparant avec ceux obtenus dans une application réelle sur cible matérielle.

Nos travaux sur le modèle de tâche que nous avons proposé, ont beaucoup intéressé la communauté temps réel, d'ailleurs dans [75] le professeur Baruah écrit :

« *However, as far as we are aware all such models assume that the entire task is characterized by a single "period" parameter indicating the minimum amount of time that must elapse between successive invocations of the task; to our knowledge, the non-cyclic GMF model is the first model for recurrent tasks considered by the real-time scheduling community, in which each recurrent task is not characterized by a single period parameter (thereby ruling out the possibility of modeling its behavior by a cyclic schedule). Moyo et al. [13, 14] call their model non-cyclic to highlight this lack of any obvious periodicity to the task's behavior. This fact – that system behavior need not be cyclic – is also why this model is not simply a special case of these prior DAG-based models for conditional real-time tasks.* »

Avec ces méthodes le concepteur peut donc mieux dimensionner son système dans l'optique de supporter les différents standards radio. Nous estimons entre 2 à 4 semaines le temps nécessaire pour l'utilisation de notre méthodologie et de notre outillage dans le cadre de la conception d'une radio logicielle. Cette courte durée passée pour déterminer la faisabilité temps réel sur un processeur dans une radio logicielle, très tôt en phase de conception, permet un gain de temps considérable dans le processus de conception d'une radio logicielle qui dure plusieurs mois. Dans une perspective industrielle et en particulier militaire, les résultats fournis par notre outil de simulation peuvent servir de certification temps réel du logiciel et peuvent être portés à la connaissance des clients lors des réponses à des appels d'offres sur les équipements radios logiciels haut débit.
Bien que les méthodes présentées dans ce document fournissent une avancée significative dans la validation temps réel d'une radio logicielle, certaines améliorations peuvent être apportées :

- Pendant la modélisation, nous avons choisi l'attribut « Timeslicelements », du stéréotype MARTE « SRM ::Swschedulableresource », pour renseigner les temps d'exécutions d'une tâche, or la spécification de MARTE ne prévoit pas une telle utilisation pour cet attribut. Comme aucun attribut de ce stéréotype n'est prévu pour les renseigner, une piste de solution serait d'utiliser l'extension VSL (Value Specification Language) de MARTE.

- Nous avons mis en place un profil avec des stéréotypes pour notre méthodologie, des contraintes OCL (Object Constraint Language) peuvent être rajoutées aux stéréotypes pour des vérifications de cohérences.

- Lorsque nous avons mesuré le niveau de précision de notre outil de simulation, nous n'avons pas intégré le temps de gestion des interruptions matérielles. Une intégration de cette valeur pourrait améliorer la comparaison des résultats prédits avec ceux obtenus sur cibles.

- Une expérimentation de notre méthodologie de conception et de notre outillage en considérant 2 standards radios pourrait aussi être envisagée.

Comme perspectives à ces travaux, on peut d'ores et déjà citer :

- Notre méthodologie de conception et outil de simulation sont adaptés pour une architecture monoprocesseur. Il serait intéressant de voir comment ces méthodes s'appliquent dans un contexte multi-processeurs. A ce propos, dans l'outil de simulation, il faudra augmenter le nombre de queues pour simuler l'exécution dans les autres processeurs et prendre en compte l'accès à la mémoire commune.

- Après la présentation de nos travaux auprès des équipes chez THALES Communications, les ingénieurs ont bien apprécié notre méthodologie et notre outillage. Cependant pour une utilisation industrielle il serait intéressant que le calcul des échéances temporelles, ainsi que les règles transformations de modèles soient développés sous forme de plugins Eclipse. Ainsi comme notre outil de modélisation est basé sous Eclipse, notre flot de conception jusqu'à la génération du fichier interprétable par l'outil d'ordonnancement temps réel sera intégré sur un seul outil.

Annexe A

Notions de base en ordonnancement temps réel

Algorithmes d'ordonnancement à priorité dynamiques (suite)

- **Least Laxity First (LLF)**

Avec l'algorithme LLF (Least Laxity First), introduit par A. K. Mok et M. Dertouzos [40] [41], à chacune de ses invocations l'ordonnanceur élit la tâche dont la laxité est la plus faible.

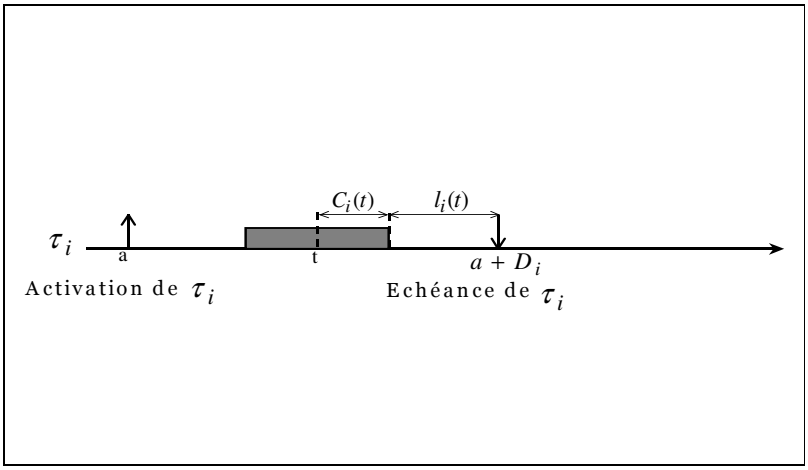

Fig. 2 – Illustration de la laxité d'une tâche activée à la date a

Comme illustrée sur la figure 2, pour une tâche τ_i activée à la date a, la laxité est définie à la date t par : $l_i(t) = a + D_i - t - C_i(t)$ où $C_i(t)$ est la durée d'exécution restante pour la tâche τ_i à la date *t*. Autrement dit, pour une tâche τ_i, à la date *t,* sa laxité $l_i(t)$ représente le temps libre qui lui reste avant sa prochaine échéance.

Théorème. 1 [41]

L'algorithme LLF, comme EDF, est optimal pour un ensemble de tâches périodiques et sporadiques en contexte préemptif.

L'algorithme LLF présente l'inconvénient suivant : lorsque plusieurs tâches possèdent la même la laxité, il engendre un grand nombre de changement de contexte, ce qui explique qu'il soit aussi peu utilisé dans le cas monoprocesseur.

Ordonnancement des tâches dépendantes (suite)

Protocole d'allocation de pile (SRP)

Baker [46] a proposé le protocole d'allocation de la pile PAP (SRP, pour Stack Resource Policy), qui est plus adapté à l'algorithme d'ordonnancement EDF. Ce protocole est une évolution du PPP, notamment vers le cas de ressources à plusieurs instances. Il consiste à attribuer un nouveau paramètre π_i, appelé **niveau de préemption**, à chaque tâche τ_i. Ce paramètre est fixé hors-ligne et est indépendant du type d'algorithme d'ordonnancement utilisé (priorité fixe ou dynamique) ; par exemple on peut attribuer les π_i suivant DM, ce qui réduit le nombre de changements de contexte avec l'algorithme d'ordonnancement EDF. Chaque ressource R possède une valeur plafond C_R, qui est la valeur maximale des niveaux de préemption des tâches actives ayant besoin de plus d'instances de la ressource R qu'il n'y en a de disponibles ; donc C_R est un paramètre dynamique.

Une tâche τ_j ne peut préempter une tâche τ_i, que si les conditions suivantes sont vérifiées :

- τ_j est plus prioritaire que τ_i (suivant l'algorithme d'ordonnancement)
- le niveau de préemption de τ_j est supérieure à celui de τ_i $\pi_j \succ \pi_i$
- le niveau de préemption de τ_j est supérieur au plafond système (la plus forte valeur plafond parmi celle des ressources).

En utilisant ce protocole, une tâche n'est pas autorisée à démarrer son exécution tant que toutes les ressources qui lui sont nécessaires ne sont pas disponibles, ce qui permet d'éviter l'interblocage.

Nous avons présenté ci-dessus des conditions d'ordonnançabilité, basées sur le facteur d'utilisation processeur. Dans le cas où aucune décision ne peut être prise, nous pouvons utiliser des techniques de test d'ordonnançabilité plus précises, basées sur l'analyse des temps de réponse ou sur l'analyse de la demande processeur.

Tests basés sur l'analyse des temps de réponse

Etant donné un système de tâches ordonnancées en ligne par un algorithme à priorités (fixes ou dynamiques), la validation temporelle du système consiste à vérifier que toutes les échéances de toutes les tâches seront respectées durant toute la durée de vie du système. L'analyse par le temps de réponse consiste à valider l'ordonnançabilité d'un système, tâche par tâche, en calculant leurs pires temps de réponse. Une tâche est ordonnançable, si son pire temps de réponse est inférieur ou égal à son échéance. Un système est ordonnançable si, et seulement si, toutes ses tâches sont ordonnançables. Puisqu'un dépassement d'échéance ne survient jamais lorsque le processeur est libre, alors le test d'ordonnançabilité est limité dans des intervalles de temps, où le processeur exécute des tâches (sans temps creux), appelés périodes d'activités.

Définition. 1

Une période d'activité du processeur est un intervalle de temps durant lequel le processeur est pleinement utilisé.

Dans le contexte des tâches périodiques, la plus longue période d'activité se produit lors d'une activation simultanée de toutes les tâches du système, cet instant est appelé instant critique [35]. Soit $t_0 = 0$ la date d'activation simultanée de toutes les tâches d'un système.

La période d'activation d'une tâche τ_j permet de calculer le nombre d'instances, de la tâche, activées dans n'importe quel intervalle de temps de longueur *t* (ces instances ne sont pas nécessairement terminées à l'instant *t*), ce nombre est égal à $\left\lceil \frac{t}{T_j} \right\rceil$.

Ainsi la durée cumulée des exécutions (appelée interférence) causée par τ_j dans *t* est :

$$W_j(t) = \left\lceil \frac{t}{T_j} \right\rceil * C_j$$

La charge totale du processeur $W(t)$ du processeur dans un intervalle de temps *t* est la somme des charges causées par toutes les tâches.

$$W(t) = \sum_{j=1}^{n} W_j(t) = \sum_{j=1}^{n} \left\lceil \frac{t}{T_j} \right\rceil * C_j$$

La figure 3 représente graphiquement la fonction de la charge processeur $W(t)$, du système de tâche du Le modèle de tâches étudié dans le cadre de cette thèse se rapporte à la catégorie des tâches multi-trames c'est pourquoi nous nous y intéressons particulièrement dans cet état de l'art., lorsque toutes les tâches s'activent simultanément. La fonction de la charge $W(t)$ est en escalier, et la ligne f(t) = t définit la capacité maximum de traitement du processeur. La première date vérifiant la condition W(t) = t, donne la longueur de la période d'activité L. L correspond à un point fixe de l'équation W(L) = L. Dans l'exemple de la figure 3 à la date t=14, W(t) = t = 14. La détermination de la longueur de la période d'activité revient alors à calculer itérativement le plus petit point fixe de l'équation W(L) = L.

τ_i	C_i	T_i
τ_1	1	4
τ_2	1	5
τ_3	2	8
τ_4	3	18

Tab. 1 - Exemple d'un système de tâches périodiques

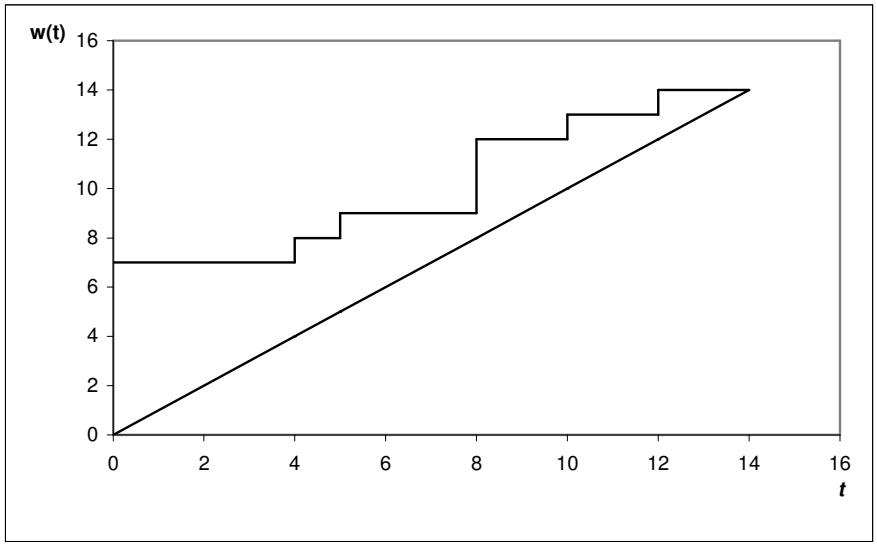

Fig. 3 - Fonction de charge processeur et période d'activité.

Puisque toutes les tâches sont réveillées à l'instant t = 0, alors L est initialisé par la somme des durées d'exécutions de toutes les tâches.

$$L^{(0)} = \sum_{j=1}^{n} C_j$$

$$L^{(k+1)} = W(L^{(k)})$$

Le calcul itératif est arrêté lorsque deux valeurs consécutives de L sont égales ($L = L^{(m)} = L^{(m+1)}$).Cette équation converge en un nombre fini d'itérations si le facteur d'utilisation du processeur est inférieur à 1 ($U = \sum_{j=1}^{n} \frac{C_j}{T_j} \leq 1$).

Nous calculons la période d'activité synchrone de notre exemple du système dont les paramètres sont donnés dans le tableau 1.

$$L^{(0)} = \sum_{j=1}^{4} C_j = 7$$

$$L^{(1)} = W(L^{(0)}) = \left\lceil \frac{7}{4} \right\rceil *1 + \left\lceil \frac{7}{5} \right\rceil *1 + \left\lceil \frac{7}{8} \right\rceil *2 + \left\lceil \frac{7}{18} \right\rceil *3 = 9$$

$$L^{(2)} = W(L^{(1)}) = \left\lceil \frac{9}{4} \right\rceil *1 + \left\lceil \frac{9}{5} \right\rceil *1 + \left\lceil \frac{9}{8} \right\rceil *2 + \left\lceil \frac{9}{18} \right\rceil *3 = 12$$

$$L^{(3)} = W(L^{(2)}) = \left\lceil \frac{12}{4} \right\rceil *1 + \left\lceil \frac{12}{5} \right\rceil *1 + \left\lceil \frac{12}{8} \right\rceil *2 + \left\lceil \frac{12}{18} \right\rceil *3 = 13$$

$$L^{(4)} = W(L^{(3)}) = \left\lceil \frac{13}{4} \right\rceil *1 + \left\lceil \frac{13}{5} \right\rceil *1 + \left\lceil \frac{13}{8} \right\rceil *2 + \left\lceil \frac{13}{18} \right\rceil *3 = 14$$

$$L^{(5)} = W(L^{(4)}) = \left\lceil \frac{14}{4} \right\rceil *1 + \left\lceil \frac{14}{5} \right\rceil *1 + \left\lceil \frac{14}{8} \right\rceil *2 + \left\lceil \frac{14}{18} \right\rceil *3 = 14$$

Un test d'ordonnançabilité, utilisant l'analyse du temps de réponse nécessite alors d'identifier le pire scénario d'exécution d'une tâche. Pour cela une connaissance, sur la loi d'arrivée des tâches et l'algorithme d'ordonnancement utilisé est indispensable. Nous allons d'abord présenter l'analyse pour les tâches périodiques avec priorité fixe, puis avec priorité dynamique.

- **Priorité fixe**

Soit S un système de *n* tâches. Sans perte de généralité, supposons que les indices des tâches indiquent leurs niveaux de priorité. Ainsi τ_1 à le niveau de priorité le plus élevé et τ_n à le plus faible. Supposons que les tâches sont affectées à des niveaux de priorités différents. Afin de calculer le pire temps de réponse, d'une tâche analysée τ_i on doit caractériser le ou les scénarios d'arrivées des tâches qui conduisent à lui. Comme τ_i ne peut être interrompue que par des tâches plus prioritaires, ceci introduit la notion de période d'activité de niveau *i*.

Définition. 2 [47]

En priorité fixe, une période d'activité du processeur de niveau i est un intervalle de temps durant lequel le processeur est pleinement utilisé, uniquement par les tâches de priorité supérieure ou égale à i.

Théorème. 2 [48]

L'instant critique d'une tâche τ_i de priorité i correspond à l'instant d'activation conduisant à la plus longue période d'activité de niveau i.

Théorème. 3 [48]

Le pire temps de réponse d'une tâche apparaît dans la période d'activité initiée par l'instant critique dans un système de tâches indépendantes, en priorités fixes.

Pour déterminer le pire temps de réponse d'une tâche τ_i, il suffira d'évaluer sa plus longue période d'activité, de calculer le temps de réponse de chaque instance de la tâche τ_i présente dans cette période d'activité puis de prendre le maximum de ces temps de réponse.

Théorème. 4 [48][35]

Dans un système de tâches indépendantes, en priorités fixes, l'instant critique d'une tâche de priorité i survient lorsque cette tâche se réveille en même temps que toutes les tâches de priorité supérieure ou égale à i.

Notons $W_i(t)$ la charge processeur encore appelée « **interférence** » causée par les tâches plus prioritaires que la tâche τ_i. Il s'agit du retard supplémentaire crée sur le temps de réponse de la tâche τ_i. On a :

$W_i(t) = \sum_{j \in hp(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil * C_j$ où $hp(\tau_i)$ est l'ensemble des indices de tâches de priorité supérieure à celle de τ_i.

Analyse du temps de réponse de la première instance

Lorsque $D_i \leq T_i$, à partir de l'équation précédente, le pire temps de réponse de la première instance de τ_i est donnée par l'équation ci-dessous :

$$R_i = W_i(R_i) + C_i = \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j + C_i$$

Théorème. 5 [48]

Le pire temps de réponse de la première instance de la tâche τ_i est défini par le plus petit point fixe de l'équation

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

La résolution de cette équation nécessite plusieurs itérations, notons R_i^k la valeur de R_i à l'itération k. On commence par $k = 0$ et $R_i^0 = C_i$. Pour trouver la valeur de R_i^{k+1}, on utilise la valeur précédente de R_i c'est à dire R_i^k, dans la partie droite de l'équation.

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

On continue le processus jusqu'à ce que R_i^{k+1} soit égal à R_i^k, il y' a convergence si et seulement si $U \leq 1$.

Analyse du temps de réponse dans le cas général

Lorsque le temps de réponse de la première instance est supérieur à T_i, alors la seconde instance fait partie de la période d'activité initiée par l'instant critique. Son temps de réponse peut donc être pire que celui de la première. Il en va de même pour les instances suivantes qui se trouveraient dans la même période d'activité.

Il est alors nécessaire de modifier l'équation précédente pour évaluer le temps de réponse de toutes les instances de la tâche τ_i situées dans la première période d'activité [47]. La longueur de la période d'activité est obtenue de manière itérative par les équations suivantes :

$$R_i^{(0)} = k * C_i$$

$$R_i^{(n+1)} = k * C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_k^{(n)}}{T_j} \right\rceil * C_j$$

Dans cette équation k est le numéro de l'instance pour laquelle on est train d'évaluer la date de fin d'exécution. Le cas k=1 correspond à la formule dans [48], vue dans la section précédente.

Pour la première instance présente dans la période d'activité (k=1) ; l'instant 0 correspond à l'instant d'activation de la première instance.

Notons $R_i(k)$ la date de fin d'exécution et $R_i^{(*)}(k)$ le pire temps de réponse de l'instance k.

$$R_i^{(*)}(k) = R_i(k) - (k-1) * T_i$$

Pour déterminer le pire temps de réponse d'une tâche τ_i, on applique la formule donnant la valeur de la longueur de la période d'activité, commençant par k=1 :

- Si $R_i^{(*)}(1) \succ D_i$ le système n'est pas ordonnançable et son pire temps de réponse est au moins $R_i^{(*)}(1)$

- Si $R_i^{(*)}(1) \succ T_i$, la deuxième instance fait partie de la période d'activité, il faut donc tester pour k=2. Si $R_i^{(*)}(2) \succ T_i$, il faut tester k=3, etc. Tant que $R_i^{(*)}(k) \succ T_i$, on teste l'instance suivante k+1. Le pire temps de réponse de la tâche est donc le pire temps de réponse des instances présentes dans la période d'activité considérée.

Exemple. 1

Soit un système de tâches non concrètes S={ $\tau_1 = (3;9;13)$, $\tau_2 = (3;10;21)$, $\tau_3 = (6;12;10)$ } données sous la forme (C_i, D_i, T_i).

Nous voulons déterminer si toutes les instances de la tâche τ_3 (tâche ayant la plus faible priorité) respecteront leurs échéances.

- $R_3^{(0)}(1) = 6$
- $R_3^{(1)}(1) = 6 + \left\lceil \frac{6}{13} \right\rceil * 3 + \left\lceil \frac{6}{21} \right\rceil * 3 = 12$
- $R_3^{(2)}(1) = 6 + \left\lceil \frac{12}{13} \right\rceil * 3 + \left\lceil \frac{12}{21} \right\rceil * 3 = 12$

Le pire temps de réponse de la 1ère instance de la tâche τ_3 $R_3^{(*)}(1)$ est égal à 12 ; $R_3^{(*)}(1) \succ T_3$

Il faut alors tester l'instance k=2.

- $R_3^{(0)}(2) = 6 * 3 = 12$
- $R_3^{(1)}(2) = 12 + \left\lceil \frac{12}{13} \right\rceil * 3 + \left\lceil \frac{12}{21} \right\rceil * 3 = 18$
- $R_3^{(2)}(2) = 12 + \left\lceil \frac{18}{13} \right\rceil * 3 + \left\lceil \frac{18}{21} \right\rceil * 3 = 21$
- $R_3^{(3)}(2) = 12 + \left\lceil \frac{21}{13} \right\rceil * 3 + \left\lceil \frac{21}{21} \right\rceil * 3 = 21$

Le pire temps de réponse de la 2ème instance de la tâche τ_3 est :

$R_i^{(*)}(2) = R_3(2) - 10 = 21 - 10 = 11$

L'instance k=3 doit être testée.

- $R_3^{(0)}(3) = 6 * 3 = 18$
- $R_3^{(1)}(3) = 18 + \left\lceil \frac{18}{13} \right\rceil * 3 + \left\lceil \frac{18}{21} \right\rceil * 3 = 27$
- $R_3^{(2)}(3) = 18 + \left\lceil \frac{27}{13} \right\rceil * 3 + \left\lceil \frac{27}{21} \right\rceil * 3 = 33$
- $R_3^{(3)}(3) = 18 + \left\lceil \frac{33}{13} \right\rceil * 3 + \left\lceil \frac{33}{21} \right\rceil * 3 = 33$

Le pire temps de réponse de la troisième instance est égal à 13 qui est supérieur au délai critique. Le système S n'est pas donc pas valide. Ici, on a le choix :

- Si la question concerne l'ordonnançabilité, on peut arrêter les tests en répondant « non »
- Si la question est de déterminer le pire temps de réponse, on continue pour k=4.

Lorsque les tâches partagent des ressources. La tâche τ_i peut être bloquée par une tâche de priorité inférieure. Notons B_i la durée de blocage maximum de la tâche τ_i due aux tâches de priorité inférieure.

Théorème. 6

L'instant critique d'une tâche τ_i avec une durée de blocage maximum B_i survient lorsque la tâche τ_i se réveille en même temps que toutes les tâches de priorité supérieure au moment où les (ou la) sections critiques sont initiées par une ou des tâches de priorité inférieure.

La formule d'évaluation de la plus longue période d'activité de τ_i devient alors :

$$R_i^{(0)} = B_i + k * C_i$$

$$R_i^{(n+1)} = B_i + k * C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil * C_j$$

Analyse du temps de réponse en présence de gigue

Soit τ_i une tâche à gigue de démarrage d'activité. Notons J_i la gigue maximale de démarrage d'activation de la tâche τ_i. Soit k l'instance de la tâche τ_i et A_i^k sa date d'activation au plus tôt. Avec une gigue de démarrage, l'instance k s'active effectivement entre A_i^k et $A_i^k + J_i$.

Théorème. 7 [49]

Dans un système S de tâches avec des gigues de début d'activation, l'instant critique de la tâche τ_i coïncide avec le réveil simultané de toutes les tâches de priorité supérieure avec une activation retardée d'une valeur correspondante à leur gigue maximale.

La prise en compte du facteur de blocage dû aux ressources se fait de la même façon que dans la section précédente. La formule de la plus longue période d'activité de niveau de priorité i est :

$$R_i^{(0)} = B_i + k * C_i$$

$$R_i^{(n+1)} = B_i + k * C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{J_j + R_i^{(n)}}{T_j} \right\rceil * C_j$$

Nous allons présenter dans la section suivante, les résultats d'analyse du temps de réponse des tâches sporadiques ordonnancées avec EDF.

o Priorité dynamique

Le pire temps de réponse d'une tâche τ_i sera déterminé dans une période d'activité où toutes les tâches d'échéances inférieures ou égales sont exécutées avant elle. L'analyse de temps de réponse d'une tâche τ_i nécessite alors d'étudier plusieurs scénarios qui correspondent aux dates d'activations possibles. Sans perte de généralité, supposons la date t=0, la date à laquelle toutes les tâches autre que τ_i sont activées simultanément. Etant donné une instance de τ_i qui à une échéance à la date *d*. Sa date de réveil est la

date $a = d - D_i$. La date $S_i(a)$ de sa première instance activée dans la période d'activité est déduite alors comme suit :

$$S_i(a) = a - \left\lfloor \frac{a}{T_i} \right\rfloor * T_i$$

La date de fin d'exécution de l'instance activée à la date *a* (avec échéance à la date *d*), correspond à la longueur de la période d'activité associée à *d*, qui est définie par les tâches dont les échéances sont inférieures ou égales à *d*. Afin de calculer l'interférence ou la charge processeur causée par une tâche τ_j (j#i) dans un intervalle de temps *t*, nous prenons en compte, parmi les instances activées dans *t*, seulement celles ayant des échéances inférieures à *d* (instances plus prioritaires).

Plus précisément, $\left\lceil \frac{t}{T_j} \right\rceil$ instances sont activées dans *t,* mais au plus $1 + \left\lfloor \frac{a + D_i + D_j}{T_j} \right\rfloor$ ont une échéance inférieure ou égale à *d*. D'où l'interférence totale des tâches plus prioritaires que τ_i est donnée par :

$$K_i(a,t) = \sum_{\tau_j \in hp(\tau_i)} \min\left\{ \left\lceil \frac{t}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} * C_j$$

La charge processeur $I_i(a,t)$ associée aux instances de τ_i est calculée en fonction de la valeur de *t* par rapport à *a* et $S_i(a)$.

$$I_i(a,t) = \begin{cases} \min\left\{ \left\lceil \frac{t - S_i(a)}{T_i} \right\rceil, 1 + \left\lfloor \frac{a}{T_i} \right\rfloor \right\} * C_i, si, t \succ S_i(a) \\ 0, \text{sinon} \end{cases}$$

La pire charge processeur totale est alors

$$W_i(a,t) = K_i(a,t) + I_i(a,t)$$

La longueur de la période d'activité $L_i(a)$ associée à *d*, peut être calculée itérativement en utilisant la formule suivante :

$$L_i^{(0)}(a) = \sum_{\tau_j \in hp(\tau_i)} C_j + \eta_{\{s_i(a)=0\}} * C_i \quad \text{où} \quad \eta_{\{S_i(a)=0\}} = \begin{cases} 1, si, S_i(a) = 0 \\ 0, \text{sinon} \end{cases}$$

$$L_i^{(k+1)}(a) = W_i(a, L_i^{(k)}(a))$$

Cette équation converge vers un point fixe si, et seulement si, la charge processeur est inférieure ou égale à 1 ($U = \sum_{j=1}^{n} \frac{C_j}{T_j} \leq 1$).

$L_i(a)$ représente la date de fin d'exécution de l'instance, de τ_i activée à la date *a*. D'où le temps de réponse $R_i(a)$ de cet instance est donné par :

$$R_i(a) = \max\{C_i, L_i(a) - a\}$$

Ces résultats ont été présentés dans [50].

Tests basés sur la demande processeur

L'analyse basée sur la demande processeur donne une réponse sur l'ordonnançabilité globale du système, au contraire de l'analyse basé sur le temps de réponse qui est appliqué sur chacune des tâches du système.

Définition. 3

L'analyse de la demande processeur revient à tester pour tout intervalle de temps [t1, t2] que la durée maximum cumulée des exécutions des requêtes qui ont leur réveil et leur échéance dans l'intervalle est inférieure à t2-t1 (c.-à-d., n'excède pas la longueur de l'intervalle).

- **Priorité fixe**

Dans l'article [33], les auteurs ont proposé une condition nécessaire et suffisante dans le cas où les tâches sont simultanées (synchrones).

Théorème. 8 [33]

Soit un système S composé de n tâches indépendantes, périodiques à échéance sur requêtes et synchrone, avec $T_1 \le T_2 \le ... \le T_n$ *. S est ordonnançable par RM si et seulement si :*

$$\forall i, 1 \le i \le n : \min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_i} \right\rceil \le 1$$

$$avec, S_i = \left\{ kT_j, 1 \le j \le i, k = 1, ..., \left\lfloor \frac{T_i}{T_j} \right\rfloor \right\}$$

Le théorème suivant donne une condition nécessaire et suffisante [53] pour l'algorithme DM. Cette condition est l'extension du test de [33], aux cas de tâches à échéances inférieures aux périodes.

Théorème. 9 [53]

Soit un système S composé de n tâches indépendantes, périodiques à échéances contraintes $(D_i \le T_i)$ *, avec* $D_1 \le D_2 \le ... \le D_n$ *. S est ordonnançable par DM si et seulement si :*

$$\forall i, 1 \le i \le n : \min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_i} \right\rceil \le 1$$

$$avec, S_i = \{D_i\} \cup \left\{ kT_j, 1 \le j \le i, k = 1, ..., \left\lfloor \frac{D_i}{T_j} \right\rfloor \right\}$$

- **Priorité dynamique**

L'analyse de la demande processeur est généralement utilisée pour l'analyse d'ordonnancement produit par EDF. Dans [54], les auteurs montrent qu'un système de tâches sporadiques est ordonnançable par EDF si le système synchrone, avec un taux d'utilisation inférieure à 1, est ordonnançable. L'analyse est alors basée sur la fonction de la demande processeur causée par des tâches activées et devant être terminées dans l'intervalle [*t1*, *t2*].

Notons *dbf(t1, t2)* la demande processeur causée par les tâches activées et devant être terminées dans l'intervalle [*t1, t2*].

$$dbf(t1,t2) = \sum_{i=1}^{n} n_i(t1,t2)C_i$$

Où $n_i(t1,t2)$ correspond au le nombre d'instances de chaque tâche τ_i dont l'activation et l'échéance est dans l'intervalle [*t1*, *t2*].

Une condition nécessaire d'ordonnançabilité est de vérifier que la demande processeur causée par les tâches dan s tout l'intervalle de temps est toujours inférieure ou égale à la longueur de l'intervalle :

$\forall 0 \le t1 \prec t2 \;\; dbf(t1,t2) \le t2 - t1$

Baruah [54] définit un intervalle d'étude pour EDF débutant à l'instant critique 0, jusqu'à $H = ppcm(T_i)$ pour les tâches périodiques synchrone, ou jusqu'à $O + 2H$ avec ($O = \max\{O_i\}$) pour les tâches asynchrones.

Lemme. 1 [31]

Un système de tâches S est ordonnançable sur une architecture monoprocesseur si et seulement si :

- $U \le 1$
- $\forall 0 \le t1 \prec t2 \le O + 2H$, $dbf(t1,t2) \le t2 - t1$

L'analyse repose sur le constat qu'un dépassement d'échéance ne survient jamais lorsque processeur est oisif (n'a rien à faire). Il est prouvé que le premier dépassement d'échéance s'il existe, se produit dans la plus longue période d'activité. L'analyse d'ordonnançabilité est alors restreinte dans la plus longue période d'activité en vérifiant que toutes les échéances des tâches sont respectées dans cette période d'étude qui débute à l'instant t=0. Dans [31], les auteurs ont prouvé que, pour un système synchrone, la longueur de la période est limitée dans le cas $U \prec 1$ par :

$$\frac{U}{1-U} \max_{i=1..n}(T_i - D_i)$$

Avec les méthodes proposées, l'évaluation du pire temps de réponse d'une tâche τ_i suppose un réveil simultané de cette tâche avec l'ensemble des tâches de priorité supérieure ; pourtant il peut arriver que le réveil de certaines tâches de priorité supérieure ne puisse pas coïncider avec le réveil de la tâche τ_i . Dans les modems radio, c'est le cas, car il existe des contraintes de précédence dans la chaîne de traitement. De plus, les modems radio sont en général half-duplex ; on ne peut pas émettre et recevoir en même temps, ce qui implique qu'il y'a forcément un décalage d'activations entre la première tâche de la chaîne d'émission, et la première tâche de la chaîne de réception.

En outre pour chaque tâche du système, le temps d'exécution pris en compte pour l'évaluation du pire temps est le WCET, c'est à dire le pire temps d'exécution, or dans certains systèmes, le temps d'exécution d'une tâche peut varier sensiblement d'une période à l'autre. C'est le cas des modems radio qui traitent différents types de trames, dans ce contexte, le temps d'exécution et l'échéance temporelle d'une tâche varient en fonction du type de trame en cours de traitement.

Face à ces situations les modèles de tâches présentées précédemment sont très pessimistes. Afin de réduire ce pessimisme, des modèles de tâches plus proches de la réalité ont été proposés.

Analyse de l'ordonnançabilité des tâches avec décalage d'activations (transactions)

Afin de prendre en compte les décalages d'activations entre les tâches, le modèle de tâches à offsets (ou transaction) a été proposé pour regrouper un ensemble de tâches déclenchées par un même événement.
Une transaction est alors une collection de tâches liées par des relations d'offsets.

Définition. 4
Une transaction Γ_i est activée périodiquement ou sporadiquement ; elle est non concrète. Le temps minimum entre les transactions sporadiques est la période. La période de la transaction est notée T_i. Chaque transaction est composée d'un nombre fixe de tâches. En plus des paramètres classiques (le WCET C_{ij}, le délai critique D_{ij}), à chaque tâche τ_{ij} d'une transaction Γ_i sont associés les paramètres suivants :

- *O_{ij} : c'est l'offset de la tâche. La tâche τ_{ij} est activée O_{ij} unités de temps après l'activation de la transaction à laquelle elle appartient.*
- *J_{ij} : gigue maximale de début d'activation. La tâche τ_{ij} se réveille alors entre O_{ij} et $O_{ij} + J_{ij}$ après l'activation de la transaction Γ_i.*

Dans [49], chaque tâche peut être dotée d'un attribut e_{ij} qui exprime le nombre d'activations de la transaction pour une activation de la tâche ; la tâche τ_{ij} a une période $e_{ij} * T_i$.
Le paramètre e_{ij} a été introduit à cause de certaines applications dans l'aérospatial où les données sont rassemblées périodiquement et chaque n périodes, certains traitements supplémentaires sont effectués. Nous supposerons dans la suite que $e_{ij} = 1$.
Les tâches peuvent aussi prendre ou libérer des sémaphores par exemple selon l'algorithme de priorité à héritage plafonné ; ce qui introduit un facteur de blocage B_{ij} représentant le temps maximum qu'une tâche τ_{ij} peut être retardé par les tâches de priorité inférieure.

Un système Γ de tâches est composé d'un ensemble de $|\Gamma|$ transactions Γ_i activées par des évènements externes.
Formellement un système de transaction est défini comme suit :

$$\Gamma : \{\Gamma_1, \Gamma_2, ..., \Gamma_{|\Gamma|}\}$$
$$\Gamma_i : \{\tau_{i1}, \tau_{i2}, ..., \tau_{i|\Gamma_i|}, T_i\}$$
$$\tau_{ij} :< C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij} >$$

La figure 4 présente un exemple de transaction Γ_i contenant trois tâches $\tau_{i1}, \tau_{i2}, \tau_{i3}$ avec pour temps d'exécution $C_{i1} = 3, C_{i2} = 2$ et $C_{i3} = 4$. Les trois tâches ont la même période T_i (période de la transaction). Les tâches $\tau_{i1}, \tau_{i2}, \tau_{i3}$ s'activent respectivement aux instants $O_{i1} = 1, O_{i2} = 6$ et $O_{i3} = 11$ après la date d'activation de la transaction.

Plusieurs travaux ont été effectués sur l'analyse d'ordonnançabilité des transactions. Tous les travaux trouvés dans la littérature portent sur l'analyse de temps de réponse. Dans le contexte des transactions, calculer le pire temps de réponse d'une tâche τ_{ua} repose sur l'approche suivante :

– Identifier et examiner les périodes d'activités de niveau *ua*, ainsi que les instants critiques.
– Calculer le temps de réponse correspondant à chacun des instants critiques ; le maximum est le pire temps de réponse de τ_{ua} .

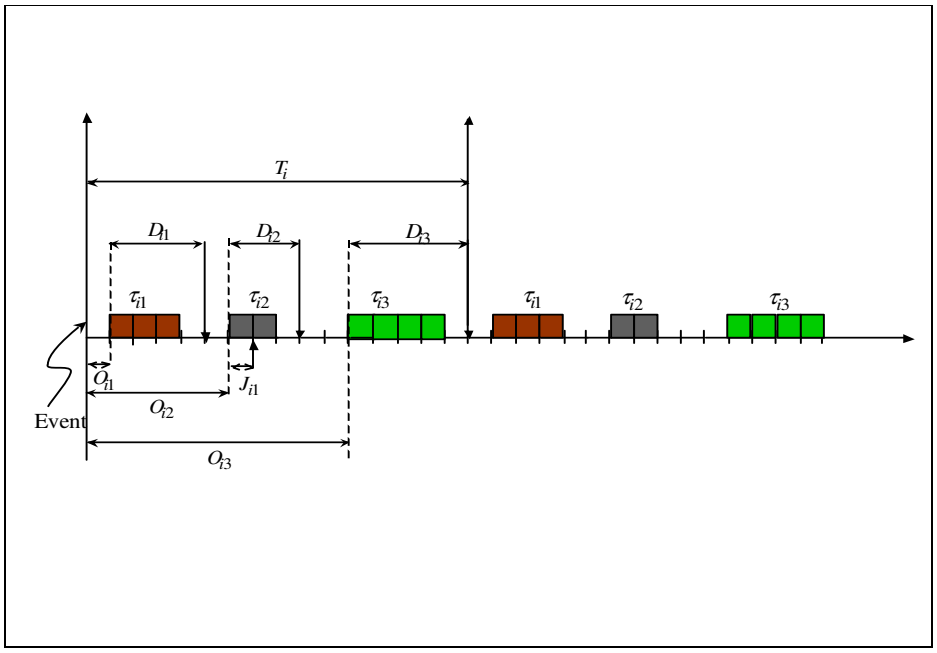

Fig. 4 – Exemple d'une transaction

Identification du pire scénario

Nous savons que le pire temps de réponse d'une tâche se produit toujours dans une de pire période d'activité de niveau *ua*. Afin de pouvoir déterminer le pire scénario d'exécution de τ_{ua} , il est indispensable de caractériser les pire périodes d'activité durant lesquelles le pire temps de réponse peut se produire, et ainsi limiter le nombre de périodes à considérer.

Théorème. 10 [56]

La contribution pire cas d'une transaction Γ_i *lors d'un instant critique d'une tâche analysée* τ_{ua} *est obtenue quand la première activation d'une tâche* τ_{ic} *de chaque transaction* Γ_i *plus prioritaire que* τ_{ua} *coïncide avec cet instant critique après avoir été retardée par sa gigue maximale* J_{ic} .

Le théorème 12 détermine les instances d'une tâche qui peuvent contribuer à une période d'activité de niveau *ua*.

Théorème. 11 [49]

Le temps d'exécution d'une instance d'une tâche τ_{ij} plus prioritaire que τ_{ua} activée avant le début de la période d'activité de niveau ua (ua étant le niveau de priorité de τ_{ua}) peut être ignoré pourvu qu'une période d'activité supplémentaire démarrant au réveil de la tâche τ_{ij} soit examinée.

A ce stade de l'étude, nous connaissons les pires scénarios (les périodes d'activités) à considérer, ainsi que les instants critiques que nous devons étudier pour trouver le pire temps de réponse. Nous allons présenter dans la suite la procédure de calcul du temps de réponse d'une tâche τ_{ua} pour chacun des scénarios possibles.
Les différentes sources d'interférence qui peuvent participer à retarder l'exécution de τ_{ua} sont :

- Interférence causée par les tâches ayant des priorités supérieures à celle de τ_{ua} (pour chacune des transactions)
- Interférence causée par les instances de τ_{ua} activée avant l'instance analysée (dans le cas où le délai critique peut être supérieur à la période)
- Interférence causée par des tâches moins prioritaires partageant des ressources critiques avec τ_{ua} . Cette interférence est bornée par une valeur B_{ua} , estimée selon le protocole d'accès aux ressources considéré.

Dans une période d'activité de longueur *t*, plusieurs instances de la tâche τ_{ij} , d'où l'interférence de la tâche τ_{ij} pour cet instant est représentée par les différentes instances de τ_{ij} .

Palencia et Harbour [56] distinguent ces instances en trois classes :

- Set0 : les instances qui sont activées avant l'instant critique (début de la période d'activité) même avec un retard d'activation par une valeur maximale de la gigue. Ces instances correspondent au résultat du théorème 1.26 et n'auront aucune contribution dans la période d'activité.
- Set1 : les instances activées à l'instant critique après avoir été retardées par une certaine valeur de la gigue pour coïncider avec l'instant critique.
- Set2 : les instances activées après l'instant critique (dans la période d'activité).

Théorème. 12 [56]

Soit t_c l'instant critique d'une tâche τ_{ua} et ϕ le temps entre l'activation de la transaction Γ_i et l'instant critique t_c . La contribution pire cas de la tâche τ_{ij} apparaît quand les instances de l'ensemble Set1 ont une gigue telle qu'elles s'activent toutes à l'instant critique tandis que les activations dans l'ensemble Set2 ont une gigue nulle.

Notons que par la suite, les indices *ijc* utilisés correspondent à la tâche τ_{ij} dans un scénario où τ_{ic} est la tâche de Γ_i dont l'activation correspond à l'instant critique étudié. La phase ϕ_{ijc} est définie comme l'intervalle de temps entre l'instant critique et la première activation de τ_{ij} (avec gigue nulle) après l'instant critique (date d'activation de la première instance dans Set2).

$$\phi_{ijc} = (O_{ij} - (O_{ic} + J_{ic})) \bmod T_i$$

La partie d'interférence causée par les instances de Set1 est indépendante de la longueur de la période d'activité. Elle est principalement liée à la valeur maximale de la gigue. Cette partie d'interférence est donnée par l'équation suivante :

$$I_{ijc}^{Set1} = \left\lfloor \frac{J_{ij} + \phi_{ijc}}{T_i} \right\rfloor C_{ij}$$

La deuxième partie est l'interférence causée par les instances de Set2 qui dépend de la longueur de la période d'activité *t,* son équation est donc donnée en fonction de la longueur de la période d'étude *t*.

$$I_{ijc}^{Set2} = \left\lceil \frac{t - \phi_{ijc}}{T_i} \right\rceil C_{ij}$$

D'où l'interférence totale $W_{ijc}(t)$ de la tâche τ_{ij} sur une tâche de plus faible priorité pendant un intervalle de temps *t* quand la tâche τ_{ic} initie l'instant critique est égale à la somme des interférences des deux ensembles Set1 et Set2 :

$$W_{ijc}(t) = I_{ijc}^{Set1}(t) + I_{ijc}^{Set2}(t)$$

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(\left\lfloor \frac{J_{ij} + \phi_{ijc}}{T_i} \right\rfloor + \left\lceil \frac{t - \phi_{ijc}}{T_i} \right\rceil \right) C_{ij}$$

La difficulté qui survient pour obtenir le pire temps de réponse, est de trouver dans chaque transaction la tâche τ_{ic} qui initie l'instant critique. De plus, celle-ci dépend de la longueur de la période d'activité. Pour effectuer une analyse exacte, il faut déterminer l'interférence en effectuant toutes les combinaisons de toutes les tâches dans chaque transaction et choisir la combinaison donnant le pire temps de réponse. Le lecteur intéressé par la méthodologie détaillée du calcul de temps de réponse pourra se référer aux documents [49][56].

Malheureusement cette méthode d'analyse a une complexité exponentielle. Afin d'éviter ce problème de complexité plusieurs travaux ont été réalisés. Tindell [49] a proposé une méthode approchée fournissant une borne supérieure du pire temps de réponse avec une complexité pseudo-polynomiale. Cette approche a été formalisée et étendue pour les tâches à offsets dynamiques par Palencia et Harbour [56]. La fonction d'interférence approchée utilisée dans cette méthode a été améliorée (fonction d'interférence effective) par Turja et Nolin [57][58][59].Cette nouvelle fonction a permis de réduire le pessimisme de la borne approchée du pire temps de réponse.

Bibliographie liée à l'étude

1. Noël TCHIDJO MOYO, Eric NICOLLET, Frédéric LAFAYE, Christophe MOY
« Model-Based Scheduling Analysis of Flexible Signal Processing Systems »
Submitted to IEEE Communications Magazine

2. BREVET D'INVENTION
« Procédé d'ordonnancement temps réel d'un ensemble de tâches multi-trames non cycliques »
Noël TCHIDJO MOYO, Vincent SEIGNOLE, Frédéric LAFAYE
N°FR2949876, septembre 2009.
N°PCT/EP2010/063109, septembre 2010.

3. Noël TCHIDJO MOYO, Eric NICOLLET, Frédéric LAFAYE, Christophe MOY
« On Schedulability Analysis of Non-Cyclic Generalized Multiframe tasks »
In proceedings of the 22nd Euromicro Conference On Real-time Systems, Brussels, Belgium, July 2010, IEEE Computer Society Press.

4. Noël TCHIDJO MOYO, Eric NICOLLET, Frédéric LAFAYE, Christophe MOY
« Integrating Schedulability Analysis in DSP baseband Processing Design Process for SDR Equipments »
In proceedings of the 6th Karlsruhe Workshop on Software Radios, WSR'10, Karlsruhe, Germany, March 2010.

5. Noël TCHIDJO MOYO, Eric NICOLLET, Frédéric LAFAYE, Christophe MOY
« Real-time Scheduling Analysis for DSP baseband Processing in multi-channel SDR set »
In proceedings of the SDR Forum Technical conference'09, Washington DC, USA, 1-4 December 2009.

Bibliographie

[1] J. Rumbaugh and G. Booch I. Jacobson. *The Unified Modeling Language Reference Manual Second Edition*. Addison Wesley, July 2004.

[2] G. Booch, R.A Maksimchuck, M. W. Engle, B. J. Young, J. Conallen, and K. A. Houston. *Object-Oriented Analysis and Design with applications, 3rd Edition*. Addison Wesley, Redwood City, CA, USA, April 2007.

[3] D. J. Armstrong. *The quarks of object-oriented development*. Communications of the ACM, 49(2):123 128, February 2006.

[4] OMG.*Unified Modelling Language, Superstructure Specification v2.2,* february 2009.

[5] OMG. *UML Profile for Schedulability, Performance an Time*, january 2005.

[6] OMG. *UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms*, April 2008

[7] OMG. *UML Profile for Modeling and Analysis of Real-time and Embedded Systems*, june 2008.

[8] E. A. Lee, S. Neuendorffer. *Actor-oriented models for codesign: balancing re-use and performance*, pages 33-56. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[9] OMG. *PIM and PSM for Software Component Radio*, 2004.

[10] S. Rouxel, J. Diguet, N. Bulteau, J. Carre-gourdin, J. Goubard, C. Moy. *UML Framework for PIM and PSM verification of SDR Systems*. SDR Forum Technical conference'05, Anaheim(USA), November 2005.

[11] E. A. Lee, D. G. Messerschmitt. *Static scheduling of synchronous data flow programs for digital signal processing*. IEEE transactions on computers, volume 36, January 1987.

[12] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall ISBN 0-13-153289-8.

[13] G. Kahn. *The semantics of a simple language for parallel programming*. In Jack L. Rosenfeld(Ed.): information Processing 74, proceedings of IFIP congress 74. stockholm, Sweden, August 5-10, 1974.

[14] E. A. Lee, S. Neuendorffer, *Tutorial : Building Ptolemy II Models Graphically*. Technical Report No. UCB/EECS-2007-129, October 31, 2007.

[15] OMG. *MOF 2.0 query/view/transformation (QVT), v1.0*. April 2008.

[16] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu. *A next generation design framework for platform-based design*. In conference on Using Hardware Design and Verification Languages (DVCon), February 2007.

[17] S. Rouxel. *Modélisation et caractérisation de plates-formes SoC hétérogènes : Application à la radio logicielle*. Thèse de doctorat, Décembre 2006.

[18] D. D. Gajski, R. H. kuhn.*New VLSI Tools*. IEEE Computer, vol. 16, P. 11-14, IEEE Computer Society Press, December 1983.

[19] S. Lecomte, S. Guillouard, C. Moy, P. Leray, P. Soulard. *A co-design methodology based on model driven engineering for SDR equipments*. SDR Forum technical conference'09, December 2009, Wasghington D.C USA.

[20] C. Harel and A. Pnuelli.*On the development of reactive systems*. In logic and models of concurrent systems. Nato Asi Series: Computer and Systems sciences; Vol. 13.

[21] A. Girault, H. Kalla, and Y. Sorel. *Une heuristique d'ordonnancement et de distribution tolérante aux pannes pour systèmes temp réel embarqués*. Modélisation des systèmes réactifs, MSR'03, Metz, France. Hermes, pages. 145-160, Octobre 2003.

[22] L. Zaffalon. *Programmation concurrente et temps réel*. Presses Polytechniques et Universitaires Romandes (PPUR), 2007.

[23] J. Xu, D. L. Parnas. *On satisfying timing constraints in hard real-time systems*. IEEE transactions on software engineering. Vol. 19. pages : 70-84, 1993.

[24] M. Richard. *Contribution à la validation des systèmes temps réel distribués : ordonnancement à priorités fixe et placement*. Thèse de doctorat, 2002.

[25] G. Le Lann. *A methodology for designing and dimensioning critical complex computing systems*. IEEE Symposium and Workshop on engineering of computer based systems, pp. 332-339, March 1996.

[26] A. Shaw. *Reasoning about time in higher-level language software*. IEEE Transactionson on Software Engineering. 15(7):875-889, December 1989.

[27] P. Pushner and C. Koza. *Calculating the maximum execution time of real-time programs*. Real-time Systems, 1(2) : 159-176, December 1980.

[28] C. Park and A. Shaw. *Experiment with a program timing tool based on source-level timing schema*. IEEE Computer society press, 24(5) :48-57, 1991.

[29] P. Puschner and R. Nossal. *Testing the results of static worst-case execution time analysis*. In proceedings of IEEE Real-time Systems Symposium, Madrid, Spain, 1 :134-143, December 1998.

[30] J. Engblom and A. Ermedahl. *Modeling complex flows for worst case execution time analysis*. In proceedings of the 21st IEEE Real-time Systems Symposium (RTSS 2000), Orlando, Florida, USA, December 2000.

[31] S. Baruah, R. Howel, L. Rosier. *Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor*. Real-time Systems, Vol. 2 PP.301-324, 1990.

[32] D. Decotigny. *Une insfrastructure de simulation modulaire pour l'évaluation de performances de systèmes temps réel*. Thèse de doctorat, Institut de Recherche en informatique et systèmes aléatoires (IRISA)/INRIA Rennes, Université RENNES 1, 07 avril 2003.

[33] J. Lehoczky, L. Sha, Y. Ding. *The rate monotonic scheduling algorithm : exact characterization and average case behaviour*. In proceedings of Real-time Systems Symposium, 1989.

[34] S. Baruah. *Dynamic and static priority scheduling of recurring real-time tasks*. Journal of real-time Systems 24(1), pages 93-128, 2003.

[35] C. L. Lui and J. W. Layland. *Scheduling algorithms for multiprogramming in real-time environnment*. Journal of ACM 20(1): 46-61, January 1973.

[36] J. T. Leung and J. Whitehead. *On the complexity of fixed-priority scheduling of periodic real-time tasks*. Performance Evaluation, 2, pages 237-250, 1982.

[37] N. C. Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. Technical report, YCS164, Dept. Computer science, University of York, UK.

[38] M. Dertouzos. *Control Robotics : the procedural control of physical processors*. Proceedings of IFIP congress, pp. 807-813, 1974.

[39] K. Jeffay, D. F. Stanat, and C. U. Martel. *On non-preemptive scheduling of periodic and sporadic tasks*. IEEE Real-time Systems Symposium, pp.36-45, 1997.

[40] A. K. Mok and M. L. Dertouzos. *Multiprocessor scheduling in hard real-time environments*. Proc. Seventh Texas Conference on computing Systems, November 1978.

[41] A. K. Mok. *Fundamental design problems for hard real-time environments*. MIT PhD. Dissertation, May 1983.

[42] J. Beauquier and B. Berard. *Sytème d'exploitation – concepts et algorithmes*. McGraw-Hill, 1990.

[43] L. Sha, R. Rajkumar, J. Lehoczky. *Priority inheritance protocols : an approach to real-time synchronization*. IEEE Transactions on computers, 39(9), pages 1175-1185, 1990.

[44] F. Cottet and E. Grolleau. *Systèmes temps réel de contrôle de commande :conception et implémentation*. Dunod, Paris, 2005.

[45] M. Chen, K. Lin. *Dynamic priority ceiling : a concurrency control for real-time systems*. Real-time Systems, 2(4) pages:625-346, 1990.

[46] T. Baker. *Stack-based scheduling of real-time processes.*Journal of real-time systems, 3:67-99, 1999.

[47] J. Lehoczky. *Fixed priority scheduling of periodic task sets with arbitrary deadlines*. In proceedings 11th IEEE Real-time Systems Symposium, pp 201-209 Dec. 1990.

[48] M. Joseph, P. Pandya. *Finding response time in a real-time system*. Computer journal, 29(5):390-395, October 1986.

[49] K. Tindell, *Adding Time-Offsets to schedulability Analysis, Technical Report YCS 221, Dept of computer science, University of york, England, January 1994.*

[50] M. Spuri. *Analysis of deadline scheduled real-time systems*. INRIA Research Report. No. 2772, jan. 1996.

[51] A. Rahni. *Contributions à la validation d'ordonnancement temps réel en présence de transactions sous priorités fixes et EDF*. Thèse de doctorat, Décembre 2008.

[52] K. Traore. *Analyse et validation des applications temps réel en présence de transactions : Applications au pilotage d'un drône miniature*. Thèse de doctorat, Février 2007.

[53] J. Lehoczky, L. Sha, J. Strosnider, H. Tokuda. *Fixed priority scheduling theory for hard real-time systems*. Foundations of Real-time Computing : scheduling and resource management, Kluwer Academic Publishers, pages 1-30, 1991.

[54] S. Baruah, A. Mok, and L. Rosier. *The preemptive scheduling of sporadic real-time tasks on one processor*. Proceedings of 11th Real-time Systems Symposium, pages 182-190, 1990.

[55] S. Baruah. *The uniprocessor scheduling of sporadic real-time tasks*. PhD Thesis, Department of Computer Science. The University of Texas at Austin, 1993.

[56] J. Palencia, M. G. Harbour. *Schedulability analysis for tasks with static and dynamic Offsets*. In proceedings of IEEE Real-time Systems Symposium ,1998, IEEE computer society press.

[57] J. Maki-Turja, M. Nolin. *Faster response time analysis of tasks with offsets*. In proceedings of the 10th Real-Time Technology and Application Symposuim, Toronto, Canada, May 2004.

[58] J. Maki-Turja, M. Nolin. *Tighter response time analysis of tasks with offsets*. In proceedings of the 10th International conference on real-time computing and applications, August 2004.

[59] J. Maki-Turja, M. Nolin. *Fast and Tight response-times for tasks with offsets*. In proceedings of the 17th Euromicro Conference on Real-time Systems, IEEE Computer society press, Palma de Mallorca Spain.

[60] A. K. Mok, D. Chen. *A multiframe model for real-time tasks*. In proceedings of the 17th real-time systems symposium, Washington D.C, USA, 1996. IEEE Computer Society press.

[61] Y. Lee, Y. Altenbasak, R. M. Mersereau. *Multiframe error concealment for MPEG-coded video delivery over error-prone*. IEEE transactions image processing, pages 1314-1331, vol 11, 2002.

[62] S. K. Baruah, D. Chen, A. Mok. *Static-priority scheduling of multiframe tasks*. In proceedings of the 11th Euromicro Conference on real-time systems, York, England, 1999. IEEE Computer Society press.

[63] A. Zuhily. *Exact response time analysis for multiframe tasks*. Technical report YCS-2007-410, Dept of computer science, University of York, England, 2007.

[64] A. Zuhily, A. Burns. *Exact Scheduling analysis of non-accumulatively monotonic multiframe tasks*. Real-Time systems journal, vol. 43, pages 119-146, 2009.

[65] S. Baruah, D. Chen, S. Gorinsky, A. Mok. *Generalized multiframe tasks*. Real-time Systems: the international journal of time-critical Computing, 17(1):5-22, july 1999.

[66] H. Takada, K. Sakamura. *Schedulability of generalized multiframe task sets under static priority assignment*. In proceedings of the 4th international workshop on real-time computing systems and applications, 1997.

[67] F. Singhoff, J. Legrand, L. Nana, L. Marcé. *Cheddar: a Flexible real-time scheduling framework*. ACM SIGAda Ada Letters, vol. 24 pages 1-8, ACM press, New York, USA, December 2004.

[68] M. G. Harbour, J.J. G. Garcia, J.M. Drake Moyano. *MAST : Modeling and Analysis Suite for real-time applications*. In proceedings of the 13th Euromicro Conference On Real-Time Systems, Delft, Netherlands, june 2001. IEEE Computer Society Press.

[69] Rapid RMA Tutorial, Tri Pacific software inc.

[70] Gregory Gailliard, *Towards a common component and middleware approach for reconfigurable distributed real-time embedded systems*. Thèse de doctorat, Décembre 2009.

[71] N. C. Audsley, K. Tindell, A. Burns. *The end of the road for static cyclic scheduling*. In proceedings of 5th Euromicro Workshop on Real-time Systems. 1993, pages 36-41, oulu, Finland.

[72] J. C. Palencia, M. G. Harbour. *Offset-based response time analysis of distibuted systems scheduled under EDF*. In proceedings of the 15th Euromicro Conference on Real-Time Systems, 2003, IEEE Computer Society Press.

[73] R. Pellizzoni, G. Lipari. *Improved schedulability analysis of real-time transactions with earliest deadline scheduling*. In proceedings of the 11th IEEE Real-time on Embedded Technology and Applications Symposium, 2005

[74] K. Traore, E. Golleau, A, A. Rahni, M. Richard. *Response time analysis of tasks with offsets. 11th IEEE International Conference on emerging Technologies and factory Automation (ETFA 2006), Prague Czech Republic, September 2006.*

[75] S. Baruah. *Preemptive uniprocessor scheduling of non-cyclic GMF task systems.* In proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Macau, China. August 2010. IEEE Computer Society Press.

[76] S. Baruah. *The non-cyclic recurring real-time task model.* In proceedings of the IEEE Real-time Systems Symposium (RTSS), San Diego, CA, December 2010. IEEE Computer Society Press.

[77] A. Nijenhius. H. S. Wilf. *Combinatorial algorithms for computers and calculators*. Second edition, Academic Press, 1978.

[78] M. Lawley, J. Steel. *Pratical Declarative Model Transformation with Tefkat*. In Lecture Notes in Computer Science, 2006, Volume 3844/2006, 139-150.

[79] M. Gries. *Methods for evaluating an covering the design space during early design development*. Integration, the VLSI Journal, vol.38, no. 2, pp. 131-183, 2004

[80] J. Bickle. *Waveform portability and reuse across operating environments: an experience report*. In proceedings of SDR Forum Technical conference, Washington D.C, October 2008.

[81] F. Bordeleau, T. McClean. *Model Driven testing framework for SCA applications*. In proceedings of SDR Forum Technical conference, Washington D.C, October 2008.

[82] C. Haubelt, J. Keinert, T. Schlichter, M. Streubuhr, A. Deyhle, A. Hadert, J. Teich. *A SystemC-Based Design methodology for digital signal processins systems.* Eurasip Journal on Embedded Systems, 2007.

[83] M. Saksena, P. Karvelas. *Designing for schedulability: Integrating schedulability analysis with object-oriented design*. In proceedings of the 12th Euromicro Conference on Real-time Systems, Stockholm, Sweden, June 2000. IEEE Computer Society Press.

[84] C. Bartolini, G. Lipari. *An algorithm for process portioning and deadline assignment of dataflow application*. (Unpublished manuscript, available from http://pico.sssup.it/files/allegati/).

[85] S. Baruah. *A general model for recurring real-time tasks*. In proceedings of the real-time systems symposium. Madrid, Spain, 1998.

[86] A. Burns, A. J. Wellings. HRT-HOOD: A design method for hard real-time. Real-time Systems, 6(1):73-114, 1994.

[87] H. Gomaa. Software design methods for concurrent and real-time systems. Addison-Wesley, 1993

[88] L. Kabous, W. Nebel. Modeling hard real-time systems with uml the ooharts approach. In proceedings of international conference on unified modelling languages (UML'99), 1999.

[89] Z. Drey, C. Faucher, F. Fleury, V. Mahe, D. Vojtisek. *Kermeta Language Reference Manual*. (Manuscript available online http://www.kermeta.org)

[90] J. Mitola. *The Software Radio*. IEEE National Telesystems Conference, 1992.

[91] JPEO JTRS. *Software Communications Architecture specification*. May 2006 (manuscript available online http://sca.jpeojtrs.mil)

[92] SDR Forum. *The Transceiver Facility Specification*. January 2009. (manuscript available online http://www.sdrforum.org)

[93] C. Moy, M. Raulet. *High-Level Design for Ultra-Fast Software Defined Radio Prototyping on Multi-Processors Heterogeneous Platforms*. Journal on advances in electronics and telecommunications – Radio communication series vol. 1, n°1, pp. 67-85, april 2010.

[94] J-P Delahaye, C. Moy, P. Leray, J. Palicot. *Managing Dynamic Partial Reconfiguration on Heterogenous SDR platforms*. In the proceedings of the SDR Forum Technical conference, Anaheim (USA) november 2005.

[95] J. Mitola III, G. Q. Maguire, Jr. *Cognitive Radio : Making Software Radios more personal*. IEEE Personal Communications, vol. 6, pp. 13-18, august 1999.

[96] C. Moy. *High-Level Design Approach for the specification of cognitive radio equipments management APIs*. Journal of Network and systems Management, Springer Editions, vol. 18 number 1. pp. 64-96 March 2010.

[97] S. Rouxel, G. Gogniat, J-P. Diguet, J-L. Philippe, C. Moy. *From MDD concepts to experiments and illustrations*. Chapter 7 of book *Schedulability Analysis and MDD*. International Scientific and Technical Encyclopedia, September 2006, pp. 111-130.

[98] L. Godard, C. Moy, J. Palicot. *An executable Meta-Model of a hierarchical and distributed architecture management for the design of cognitive radio equipments*. Annals of Telecommunications, Special Issue on Cognitive Radio, Vol. 64 pp. 463-482, number 7-8, Aug. 2009.

[99] A. Koudri, D. Vojtisek, P. Soulard, C. Moy, J. Champeau, J. Vidal, J-C. Le Lann. *Using MARTE in a co-design methodology*. MARTE Workshop at DATE Conference, Munich, 14 March 2008.

[100] *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific Requirements, Part 11: Wireless LAN MAC and PHY specifications*,802.11, June 2007.

[101] *H.222.0: information technology – generic coding of moving pictures and associated audio information: systems*, (manuscript available online http://www.itu.int/rec/T-REC-H.222.0)

[102] *N. Tchidjo Moyo, "Développement d'un bus logiciel et d'un microframework pour DSP, portage et déploiement d'une forme d'onde couche physique générique sur DSP C5510, évolution de l'application sur DSP C6416", rapport de stage de fin d'études, Université Louis Pasteur de Strasbourg, septembre 2007*

[103] *JPEO SCA Next, http://sca.jpeojtrs.mil/scanext.asp*

VU :

Le Directeur de Thèse
Christophe MOY

VU :

Le Responsable de l'École Doctorale
Jean-Marie Lion

VU pour autorisation de soutenance

Rennes, le

Le Président de l'Université de Rennes 1

Guy CATHELINEAU

VU après soutenance pour autorisation de publication :

Le Président de Jury,
(Nom et Prénom)