l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Takieddine MAJDOUB

Mémoire présenté en vue de l'obtention
du grade de Docteur de l'Université de Nantes
Sous le label de l'Université Nantes Angers Le Mans

Discipline : Electronique
Spécialité : Systèmes embarqués
Laboratoire : IETR UMR 6164

Soutenue le 30 octobre 2012

École doctorale Sciences et Technologies de l'Information et Mathématiques (STIM)
Thèse N° ED503-177

Technique de modélisation transactionnelle en vue de l'amélioration de la simulation des modèles de performances des architectures électroniques dans le domaine automobile

JURY

Président :	**M. Bertrand GRANADO**, Professeur, ENSEA, Cergy
Rapporteurs :	**M. Guy GOGNIAT**, Professeur, Université de Bretagne Sud, Lorient
	M. Dominique HOUZET, Professeur, INP Grenoble
Examinateurs :	**M. Sébastien PILLEMENT**, Professeur, Ecole polytechnique de l'université de Nantes
Directeur de Thèse :	**Mme Fabienne NOUVEL**, Maître de Conférences/HDR, INSA Rennes
Co-encadrant :	**M. Sébastien LE NOURS**, Maître de Conférences, Ecole polytechnique de l'université de Nantes

Table des matières

Liste des figures

Liste des tableaux

Introduction

Dans le domaine automobile, les services rendus au sein des véhicules évoluent progressivement afin de répondre à l'émergence continue de nouveaux usages. Des fonctionnalités de plus en plus avancées et portant sur les différents domaines d'application présents au sein des véhicules sont progressivement intégrées. Ces nouvelles fonctionnalités sont rendues possibles par l'évolution des systèmes électroniques embarqués dans les véhicules. En effet, les architectures électroniques des véhicules se révèlent de plus en plus sophistiquées tout en satisfaisant à des exigences fortes de coût et de performance. Actuellement, l'architecture électronique embarquée au sein des véhicules repose sur un ensemble de cartes électroniques interconnectées selon différents réseaux de communication. Ces cartes, communément appelées ECUs (*Electronic Controller Unit*), intègrent différentes ressources matérielles et logicielles permettant à l'architecture de subvenir aux différents besoins applicatifs. Les ressources matérielles des ECUs s'organisent autour de plusieurs microcontrôleurs, de ressources de mémorisation, de périphériques d'entrées et de sorties et d'interfaces de communication. Les ressources logicielles correspondent à un ensemble de modules logiciels permettant l'exploitation et la gestion des ressources matérielles afin d'assurer l'exécution de l'application. Jusqu'à une époque récente, il était possible de concevoir indépendamment les ressources matérielles et logicielles de chacun des ECUs. Compte tenu des évolutions applicatives continues au sein des véhicules, ces architectures deviennent de plus en plus distribuées et les calculateurs deviennent fortement interdépendants. Différents protocoles de communication existent pour le domaine automobile afin de permettre le partage d'informations entre les différents organes du véhicule. La conception de telles architectures tend alors à devenir de plus en plus complexe et suppose de pouvoir appréhender les architectures dans leur ensemble. Dès lors, il s'avère indispensable de pouvoir prendre en compte au plus tôt lors des phases de conception l'influence des échanges entre calculateurs. Dans ce processus, l'architecte de systèmes est amené à dimensionner les architectures en définissant les ressources logicielles et les ressources de calcul, de mémorisation et de communication nécessaires afin de supporter les besoins applicatifs auxquels doivent répondre le système.

Afin de favoriser le travail des architectes de systèmes, différentes approches existent afin d'évaluer les performances des architectures possibles et d'explorer l'espace de conception. Ces approches peuvent être catégorisées selon la méthode d'évaluation des performances. Parmi les approches existantes, l'emploi de modèles d'architectures simulables offre la possibilité d'évaluer le comportement et les performances de l'architecture selon différents scénarios de fonctionnement possibles. La création de tels modèles résulte de l'association des vues exprimant respectivement l'application à supporter et les ressources de la plate-forme. Différentes approches basées sur l'emploi de modèles d'architectures simulables ont été proposées. Ces modèles sont généralement définis à un haut niveau d'abstraction afin de permettre une exploration rapide et efficace des différentes solutions de réalisation envisageables. De façon générale, ces modèles reposent le plus

souvent sur un ensemble concurrent de processus séquentiels. La simulation de ces modèles permet alors une analyse du comportement dynamique des architectures et une évaluation des performances associées. Pour ce faire, les modèles d'architectures doivent être décrits dans des langages exécutables. Dans ce contexte, le concept de modélisation transactionnelle (*TLM*) présente une solution intéressante pour la création de modèles d'architectures afin de faciliter le dimensionnement de ces systèmes. Les modèles d'architectures sont alors décrits dans des langages exécutables tels que SystemC afin de rendre la simulation possible. Les modèles peuvent être définis à différents niveaux d'abstraction conduisant à un compromis entre les temps de simulation et le niveau de précision des résultats obtenus par ces modèles. Différents travaux de recherche ont été menés afin de quantifier ce compromis et afin de proposer des techniques permettant de l'améliorer.

Les travaux de recherche présentés dans cette thèse visent à tirer parti des possibilités offertes par le concept de modélisation transactionnelle afin d'améliorer la simulation des modèles des architectures électroniques dans le domaine automobile. Notre travail a porté sur la définition d'une technique de modélisation permettant d'améliorer le compromis entre la rapidité d'exécution et la précision des résultats des modèles d'architectures en vue du dimensionnement des ressources matérielles et logicielles. Notre contribution porte sur la proposition d'une technique de modélisation visant à favoriser la création de modèles transactionnels des architectures distribuées. Cette technique permet notamment d'accélérer de manière significative la simulation de ces modèles tout en conservant un niveau de précision donné, favorisant ainsi l'exploration de l'espace de conception. Cette technique utilise le principe de fonctionnement du moteur de simulation de modèles transactionnels afin de limiter l'influence des échanges entre processus au sein des modèles et donc d'accélérer la simulation. La précision des modèles est alors conservée en effectuant des exécutions locales de l'évolution des propriétés à observer.

L'intérêt de nos travaux a été illustré à travers une étude de cas inspirée du domaine automobile. Ces travaux sont rentrés dans le cadre du projet ANR CIFAER. Dans le cadre de ce projet, un démonstrateur supportant une application de diffusion vidéo et utilisant des réseaux de communication, notamment basés sur les courants porteurs en ligne, a été défini. L'étude menée a porté sur la modélisation de l'architecture de ce démonstrateur afin de dimensionner les ressources matérielles et logicielles. Elle porte également sur l'application de la technique de modélisation proposée sur ce cas d'étude. Les résultats de simulation obtenus montrent un facteur d'accélération de l'ordre de cent tout en conservant le même niveau de précision. Les travaux menés ouvrent la voie à la création de modèles efficaces pour l'évaluation des performances des architectures électroniques du domaine automobile.

Ce document est organisé selon quatre chapitres.

Le premier chapitre présente les tendances actuelles dans le domaine des architectures électroniques embarquées au sein des véhicules automobiles et les méthodes employées pour leur conception. Parmi les problématiques identifiées de conception des architectures, nous insistons plus particulièrement sur la nécessité, pour les architectes de systèmes, de pouvoir disposer au plus tôt dans le processus de conception de modèles permettant

d'analyser l'impact des choix effectués. Les critères à considérer portent alors sur l'efficacité de ces modèles aussi bien en termes de rapidité de simulation que de précision des résultats.

Le deuxième chapitre positionne nos travaux parmi les activités existantes dans le domaine du dimensionnement des architectures de systèmes embarqués. Les différentes catégories d'approches existantes sont présentées. Dans le cadre de cette thèse, nous nous intéressons plus particulièrement aux approches basées sur la simulation de modèles et nous présentons différentes approches de création de modèles exécutables. Nous présentons également la notion de modélisation transactionnelle et les principes associés à la simulation de tels modèles. Enfin, nous présentons différents travaux menés en vue de l'amélioration des performances de ces modèles.

Le troisième chapitre présente la contribution de la thèse. Comme évoqué précédemment, cette contribution porte sur la définition d'une technique de modélisation visant à améliorer les temps de simulation des modèles tout en permettant une bonne précision des estimations délivrées. Dans ce chapitre, une qualification de l'apport de la technique proposée en termes de temps de simulation et de précision est étudiée. Un exemple didactique de l'application de la technique de modélisation est ensuite proposé.

Le quatrième chapitre explique, à travers une étude de cas inspirée du domaine automobile, l'application de la technique proposée pour la modélisation d'une architecture distribuée en vue du dimensionnement des ressources matérielles et logicielles. L'étude menée porte sur la modélisation des ressources associées au protocole de communication HomePlugAV. Il est montré de quelle manière l'application de la technique conduit à une amélioration significative des temps de simulation.

Finalement, une conclusion et des perspectives de nos travaux sont détaillées dans la dernière section.

Chapitre 1
Contexte applicatif : conception des architectures électroniques pour le domaine automobile

Sommaire

Ce chapitre présente les tendances actuelles pour les architectures électroniques embarquées au sein des véhicules automobiles. Ce chapitre présente également les méthodes employées pour leur conception. Compte tenu des évolutions observées, des problématiques nouvelles émergent vis-à-vis des phases de conception et de dimensionnement des architectures électroniques des véhicules.

Au sein de ce chapitre, la première section présente l'évolution des fonctionnalités supportées dans le domaine automobile et l'évolution des architectures des systèmes électroniques associés. La deuxième section explique les contraintes appliquées à ces architectures et décrit les tendances actuelles pour la conception des ressources matérielles et logicielles associées. La dernière section précise les problématiques de conception du domaine automobile et notamment celles portant sur la définition des ressources matérielles et logicielles.

1.1 Evolution des architectures électroniques dans le domaine automobile

Depuis les années 1920, on observe une augmentation des services rendus au sein des véhicules. Cette augmentation répond à l'émergence de nouveaux usages et se traduit par la mise en œuvre de nouvelles fonctionnalités. Ces usages nouveaux sont rendus possibles par l'évolution des systèmes électroniques embarqués dans les véhicules.

1.1.1 Evolution des fonctionnalités et de l'utilisation des systèmes électroniques

Des fonctionnalités de plus en plus nombreuses et de plus en plus avancées sont progressivement intégrées au sein des véhicules. Ces fonctionnalités sont catégorisées selon leurs domaines d'application. Classiquement, on distingue cinq domaines typiques dans l'automobile. Ces domaines correspondent à des parties spécifiques du véhicule comme le montre la figure 1.1.

Figure 1.1 – Les différents domaines d'applications présents au sein du véhicule.

Le domaine d'application du groupe motopropulseur concerne le moteur et la transmission. Les fonctionnalités liées à ce domaine d'application permettent de contrôler et d'optimiser la combustion dans le moteur et le transfert d'énergie vers les roues. Actuellement, on observe dans ce domaine que des dispositifs électriques sont de plus en plus employés pour remplacer des dispositifs mécaniques. Par exemple, l'arbre à came est remplacé par des actionneurs électromécaniques dotés de capteurs. Les fonctionnalités concernant le domaine du châssis visent à améliorer la stabilité du véhicule et la qualité de sa conduite. Par exemple, on peut citer la fonction de correction de trajectoire. De nouvelles fonctions avancées comme les « X-by-Wire » émergent également dans ce domaine. « X-by-Wire » est un terme générique qui signifie le remplacement de systèmes mécaniques ou hydrauliques par un système totalement électronique. Le domaine de la sécurité est un domaine particulièrement important. Les fonctionnalités de ce domaine assurent aussi bien la sécurité des passagers que celle des piétons. L'airbag est une des fonctions les plus répandues de ce domaine. Actuellement, des fonctions plus avancées comme la détection de collision sont rajoutées. La partie occupée par les passagers et le coffre forment l'habitacle. Les fonctionnalités de ce domaine d'application concernent le confort et l'interface homme machine. On peut citer l'exemple de la fonction de GPS (*Global Positioning System*). Le domaine multimédia concerne les nouvelles technologies de l'information et de la communication. Différentes fonctionnalités de ce domaine comme la télévision TNT (*Télévision Numérique Terrestre*) ou l'accès à internet sont disponibles au sein de véhicules.

L'ensemble de ces fonctionnalités avancées est rendu possible compte tenu de l'évolution de la part de l'électronique embarquée au sein des véhicules.

L'utilisation de l'électricité dans les véhicules remonte aux années 1920. Depuis cette époque, le nombre de fonctions confiées à l'électronique n'a cessé d'augmenter. La nature de ces fonctions évolue conjointement avec les progrès technologiques réalisés à différentes époques. Progressivement, de 1920 à 1960, différents dispositifs électriques de base sont employés dans l'automobile comme le démarreur, les lampes, la dynamo et l'autoradio. Depuis 1960, des dispositifs électriques sont intégrés dans les moteurs afin d'améliorer l'allumage et l'injection. Avec l'intégration des bus de communication dans les véhicules en 1990, il est devenu possible pour plusieurs cartes électroniques de partager des informations. En exemple, la fonction ABS (*Anti Blocker System*) utilise les réseaux de communication afin d'exploiter différentes informations comme la vitesse des roues, la force de freinage et la résistance au freinage. Vers les années 2000, l'électronique embarquée dans les véhicules offre des fonctions télématiques. Depuis, des fonctions avancées permettant ainsi la détection et l'interprétation des panneaux, la détection de collision et l'auto conduite sont progressivement intégrées dans les véhicules. La figure 1.2 récapitule l'évolution de la nature des éléments électroniques intégrés dans les automobiles ainsi que le coût associé à l'utilisation de ces éléments [**1**].

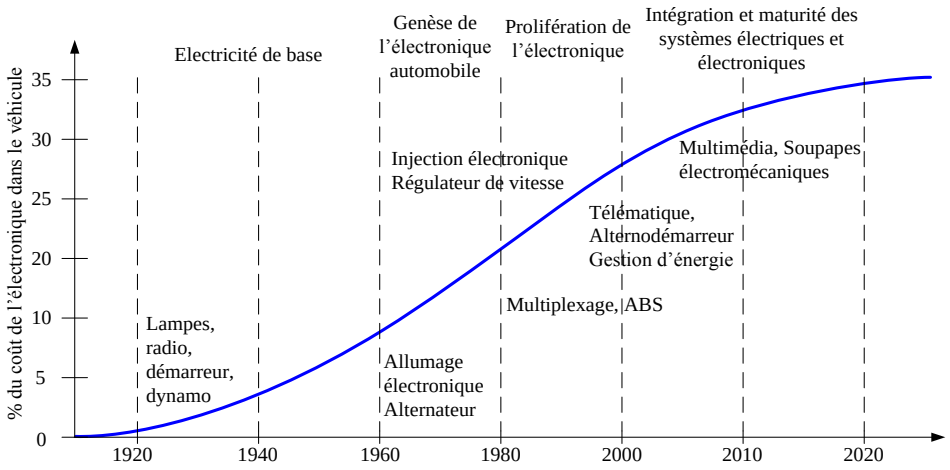


Figure 1.2 – Pourcentage du coût de l'électronique dans l'automobile.

La figure 1.2 montre l'évolution, en fonction des années, du coût de l'électronique dans l'automobile par rapport au coût global de la voiture. Elle positionne également la nature des dispositifs électroniques intégrés. On constate que le pourcentage du coût de l'électronique intégrée par rapport au coût global d'un véhicule augmente considérablement au cours du temps.

L'accroissement du nombre de ressources électroniques employées implique ainsi une croissance importante de la complexité des systèmes électroniques embarqués dans l'automobile. En 2004, on comptait 70 cartes électroniques dans les voitures haut de gamme [**2**]. De nos jours, ce nombre dépasse la centaine. Par conséquent, la complexité des architectures électroniques embarquées au sein des véhicules conduit à une densification des infrastructures nécessaires à mettre en œuvre afin de lier les différents calculateurs utilisés.

Ces derniers sont connectés à des réseaux de communication permettant l'échange d'informations entre eux. Plusieurs types de réseaux, choisis selon les exigences de débit et de fiabilité, existent et représentent des solutions standards pour différentes catégories d'application. Afin de répondre aux besoins croissants des fonctionnalités mises en œuvre, les tendances actuelles suggèrent l'utilisation d'autres supports de communication dans les véhicules du futur. Les critères importants portent sur l'amélioration des débits d'information disponibles, de la robustesse ainsi que la maîtrise des coûts d'infrastructure. Dans ce contexte, le projet CIFAER (Communication Intra-véhicule Flexible et Architecture Embarquée Reconfigurable) [**3**] a été mené afin d'étudier la faisabilité de l'application des technologies radiofréquences et Courant Porteur en Ligne (CPL) dans les véhicules.

Les différentes observations faites illustrent donc l'accroissement de la part de l'électronique au sein des véhicules ainsi que la densification des architectures embarquées.

1.1.2 Analyse et évolution des architectures électroniques des véhicules

L'architecture électronique des véhicules peut être perçue comme l'architecture d'un système embarqué à part entière. Un système embarqué est un système, associant électronique et informatique, autonome et intégré au sein d'un environnement avec lequel il interagit. L'architecture d'un tel système embarqué est distribuée et est édifiée sur trois éléments essentiels. Ces éléments sont le logiciel embarqué, les ressources matérielles de traitement et les supports de communication servant à connecter les ressources.

1.1.2.1 Protocoles de communication dans le domaine automobile

Jusqu'en 1990, les cartes électroniques embarquées dans les véhicules étaient indépendantes et chacune remplissait ses fonctions séparément. La longueur totale des câbles électriques dans une voiture dépassait alors le kilomètre. Pour réduire le coût et le poids du cuivre, les constructeurs automobiles ont commencé à intégrer des bus de communication au sein des véhicules, tel que le bus CAN (*Controller Area Network*) [**4**]. Typiquement les réseaux employés dans le domaine automobile respectent l'organisation hiérarchique en couche du modèle OSI (*Open Systems Interconnection*). Les trois couches physique, liaison et réseau caractérisent le débit, la fiabilité (tolérance aux pannes et taux d'erreur), les topologies possibles, la stratégie de communication ainsi que le nombre maximal de nœuds supportés.

Dans le contexte automobile, une classification des réseaux a été faite selon le débit et les fonctions supportées [**2**]. La classe, dénommée A, regroupe les réseaux de débit inférieur à 10 Kbit/s. Ces réseaux sont utilisés pour l'acheminement de simples informations de commande en employant des technologies à bas coût. Ils sont principalement utilisés dans l'habitacle pour la commande de verrouillage des portes. La classe B de réseau supporte un débit entre 10 et 125 Kbit/s. Ces réseaux permettent l'échange de données entre les unités de calcul et permettent le partage d'informations issues des capteurs. La classe C offrant des débits de 125 Kbit/s à 1 Mbit/s est généralement utilisée pour les domaines du groupe motopropulseur et du châssis. La dernière classe D, supportant un débit supérieur à 1Mbit/s, est employée pour le domaine multimédia.

Différentes topologies existent pour relier les différents éléments d'une architecture électronique. La topologie d'un réseau influe sur le débit dédié à chaque organe. Ainsi, la topologie en bus, actuellement la plus utilisée dans les réseaux au sein des automobiles, implique de devoir partager le support de communication entre les éléments, impliquant le partage de la bande passante du réseau. La topologie en étoile peut également être utilisée dans les véhicules. Selon les topologies on peut considérer différentes stratégies de communication.

Il existe deux paradigmes pour décrire les stratégies d'accès des réseaux de communication dans le domaine automobile. Le paradigme dit *Time-triggered* est basé sur une répartition cyclique des accès au support de communication. Un intervalle de temps est dédié à chaque nœud existant dans le réseau. Dans cet intervalle, un nœud transmet ses données ou bien une balise de rappel de sa présence. Le paradigme dit *Event-triggered* permet la transmission de données selon la production d'événements. La détection d'un événement significatif, comme l'ouverture d'une porte, engendre la transmission de cette information immédiatement. L'arbitrage par priorité est souvent utilisé si plusieurs événements se produisent. Il existe des réseaux qui adoptent ces deux paradigmes.

Parmi les stratégies de communication, l'accès multiple à répartition dans le temps (TDMA pour *Time division multiple access*) consiste à partager l'accès au support de communication dans le temps. Cet accès est cyclique. La durée d'un cycle est divisée en intervalles de temps dédiés pour chaque nœud connecté à ce réseau. Cette stratégie correspond au paradigme *Time-triggered*. L'arbitrage par priorité est basé sur le paradigme *Event-triggered*. L'accès au support de communication est accordé au nœud ou au message le plus important. Un identifiant au début du message à transmettre est considéré afin de résoudre la priorité associée. Un mécanisme de détection de priorité est nécessaire pour détecter la transmission d'un message plus prioritaire. Une des techniques de partage de support de communication qui permet la détection de priorité est le CSMA/CR (*Carrier Sense Multiple Access with Contention Resolution*). La stratégie maître-esclave impose l'existence d'un seul maître sur le réseau. L'échange de données est effectué suivant les requêtes du maître. Ce dernier contrôle le bus et scrute les esclaves afin qu'ils partagent leurs données sur le bus. Les esclaves ne fournissent des données que lorsqu'ils sont sollicités.

On cite par la suite certaines caractéristiques associées aux standards de communication les plus utilisés dans le domaine automobile.

- **CAN (*Controller Area Network*) [4]**

Le protocole CAN a été développé par la société Bosch afin de réduire la quantité de câbles dans les véhicules. Il permet de relier sur un seul bus l'ensemble des unités de calcul, des capteurs et des actionneurs. Ce protocole s'appuie sur le multiplexage des communications sur un bus formé par une paire de fils torsadés. Il a été standardisé en 1994 sous la référence ISO11898–1. Actuellement, il existe différentes déclinaisons de ce protocole dont les principales différences concernent la couche liaison et la couche physique. La première norme, CAN 2.0 A, alloue 11 bits pour le champ d'identification de trame. Dans la deuxième version de la norme, CAN 2.0 B, le champ d'identification est

étendu sur 29 bits. L'accès au bus précisé par ces deux normes est basé sur le principe du CSMA/CR. Le débit supporté par ce protocole dépend de la longueur de câble utilisée. Il peut varier de 40 Kbit/s à 1 Mbit/s pour le « High Speed CAN » et de 40 Kbit/s à 125 Kbit/s pour le *Low Speed*/*Fault Tolerant CAN*, selon la longueur de câble utilisée. D'autres déclinaisons du protocole CAN combinent deux versions d'accès au bus. TT-CAN (*Time-Triggered CAN*) et FTT-CAN (*Flexible Time-Triggered CAN*) adoptent les deux paradigmes *Event-triggered* et *Time-triggered*.

- **LIN (*Local Interconnect Network*) [**5**]**

Le protocole LIN est un protocole série à bas coût créé par le consortium LIN en 1999. Ce protocole est utilisé pour connecter des parties extrêmes et non critiques d'un réseau hiérarchique dans la voiture, comme par exemple la commande de fermeture de vitre. Ce protocole communique selon la stratégie maître-esclave et impose la présence d'un seul maître. Différentes versions se sont succédées. La version actuelle propose un débit de 1Kbit/s à 20 Kbit/s et jusqu'à 63 nœuds esclaves. Un de ses avantages est de ne compter qu'un seul câble par lequel transitent les données, auquel se rajoutent les câbles pour l'alimentation, ce qui en fait un réseau plus léger que le CAN.

- **MOST (*Media Oriented Systems Transport*) [**6**]**

Le protocole MOST a été développé par « MOST Corporation » en 1998. Ce protocole est destiné aux applications multimédias dans les véhicules qui nécessitent un haut débit de communication. Ce protocole utilise des fibres optiques qui sont mieux immunisées contre les interférences électromagnétiques que les câbles en cuivre. Le protocole MOST est un protocole point à point. Physiquement, il ne peut relier que deux équipements, mais logiquement, il peut relier jusqu'à 64 équipements en utilisant une topologie en anneau. La stratégie de communication de ce protocole est le CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*). MOST permet une transmission de données à trois débits possibles : 25 Mbit/s (MOST25) ou 50 Mbit/s (MOST50) ou 150 Mbit/s (MOST150). Actuellement, cette technologie, très coûteuse, est essentiellement utilisée dans les véhicules haut de gamme pour relier les différents systèmes de télématique.

- **FlexRay [**7**]**

Ce protocole a été standardisé en 2004 par un consortium regroupant des constructeurs automobiles, des équipementiers automobiles et des fondeurs de silicium. Ce standard de communication est défini pour des applications nécessitant des hauts débits d'informations. Il offre un débit maximal de 20 Mbit/s réparti sur deux canaux physiques répliqués. En effet, chaque canal admet un débit de 2.5 Mbit/s, 5 Mbit/s ou 10 Mbit/s, et peut supporter une topologie en bus, en étoile ou avec deux étoiles en cascade. Toutes les combinaisons de topologies formées par les deux canaux sont acceptables avec ce standard. Ce dernier peut également fonctionner avec un seul réseau vu qu'il est tolérant aux pannes. En effet, ce protocole implémente des mécanismes qui assurent une continuité de fonctionnement du système même dans le cas de défaillance de l'un des réseaux ou de l'un des deux canaux. La stratégie d'accès à un canal supporte les deux paradigmes *Event-triggered* et *Time-triggered* .

La diversité actuelle des réseaux de communication a contribué à l'évolution des infrastructures électroniques embarquées dans le domaine automobile. Progressivement, les infrastructures sont devenues capables de faire cohabiter différents protocoles de communication. L'organisation des infrastructures électroniques embarquées au sein des véhicules tend à hiérarchiser les ressources de calcul. Ces ressources sont organisées selon les différents domaines d'application. Un exemple possible d'infrastructure d'un système distribué embarqué dans les véhicules est détaillé dans la Figure 1.3.

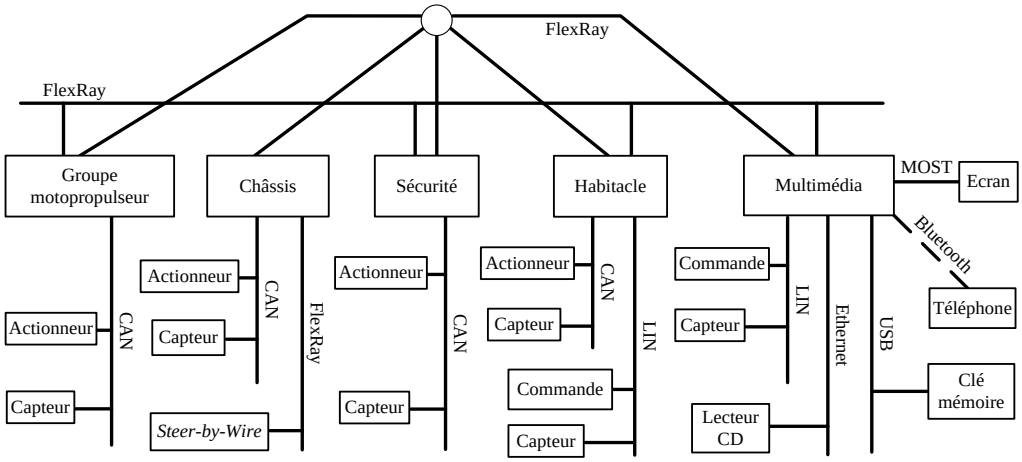


Figure 1.3 – Exemple d'architecture d'un système distribué embarqué dans les véhicules.

La Figure 1.3 présente l'organisation de ressources matérielles selon les cinq domaines d'application présents au sein d'un véhicule. Un réseau FlexRay, en bus et en étoile, relie les calculateurs dédiés à chaque domaine. Chaque calculateur peut se connecter aux différents capteurs ou actionneurs de son domaine par un bus CAN. Il peut également se connecter à des organes de commande via le réseau LIN. Des organes plus avancés, comme le *Steer-by-Wire*, sont connectés au calculateur associé par FlexRay. Le domaine multimédia peut éventuellement intégrer les réseaux Ethernet, Bluetooth et MOST.

Cette densification des réseaux embarqués au sein des véhicules va dès lors avoir une forte incidence sur les architectures matérielles et logicielles des calculateurs.

1.1.2.2 Architecture matérielle des ECUs

L'infrastructure matérielle des véhicules s'organise autour de différentes cartes électroniques, communément appelées ECUs (*Electronic Controller Unit*), communiquant via des réseaux spécifiques. Généralement, un ECU est composé d'un ou de plusieurs microcontrôleurs, de différentes ressources de mémorisation, de périphériques et d'interfaces de communication. Selon les domaines d'application identifiés dans la Figure 1.1, les ECUs diffèrent compte tenu des caractéristiques des ressources matérielles utilisées. Le choix des ressources est fait selon les contraintes de l'application à mettre en œuvre. Le microcontrôleur est le composant principal au sein d'un ECU. Les caractéristiques physiques des microcontrôleurs dédiés au domaine automobile sont particulières. Par exemple, leur robustesse est renforcée afin de résister à un froid ou à une chaleur importante, contre les perturbations électromagnétiques et les vibrations mécaniques. Par

ailleurs, leurs architectures internes intègrent des éléments spécifiques au domaine automobile. La Figure 1.4 présente un exemple d'architecture d'un microcontrôleur dédié aux applications multimédias du domaine automobile.

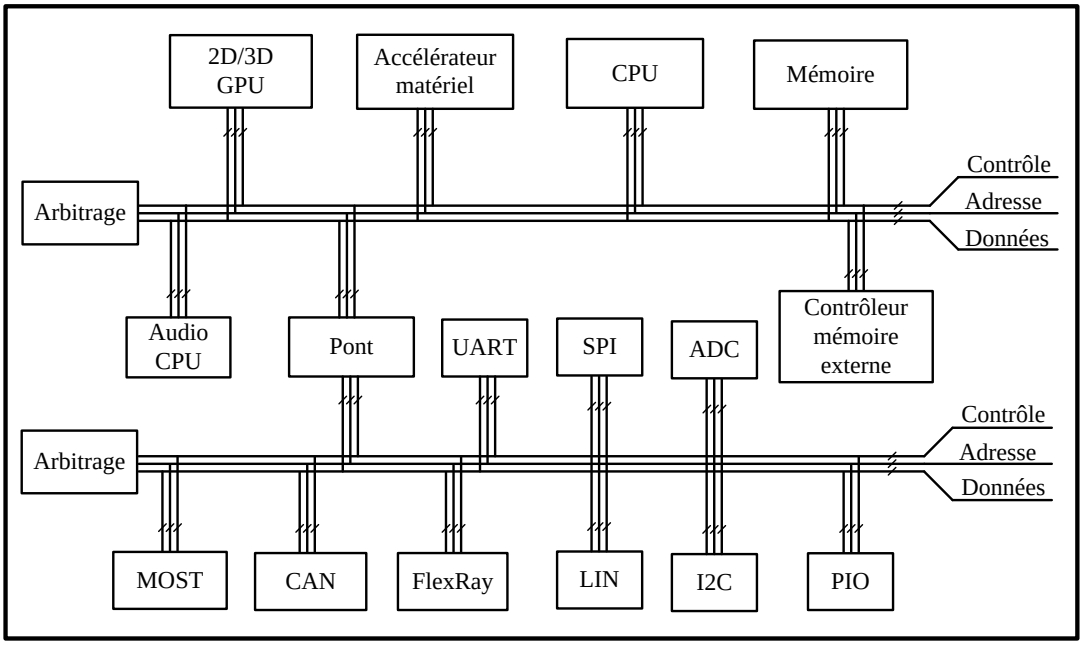


Figure 1.4 – Exemple d'architecture interne d'un microcontrôleur dédié au domaine automobile.

Un microcontrôleur dédié aux applications multimédias peut posséder plusieurs cœurs de processeur pour répondre aux besoins de l'application. Dans la Figure 1.4 on distingue différents cœurs de processeur : un processeur d'usage générique, un processeur dédié au traitement audio et un processeur spécialisé pour les traitements vidéo. Un microcontrôleur peut intégrer également des accélérateurs matériels pour épauler les cœurs de processeurs dans l'exécution de calculs gourmands en ressources. Un microcontrôleur possède différentes unités de mémorisation composées essentiellement d'une mémoire Flash et d'une mémoire RAM (*Random Access Memory*). La mémoire Flash permet de stocker le programme. La mémoire RAM permet, elle, de stocker les variables pendant l'exécution du programme. Un ensemble de bus, constitué d'un bus d'adresse, d'un bus de données et d'un bus de contrôle, permet de connecter ces différents éléments.

Un microcontrôleur dispose de plusieurs périphériques tels que des entrées et des sorties parallèles PIO (*Parallels Input/Output*), des convertisseurs numériques analogiques et analogiques numériques, des PWM (*Pulse Width Modulation*) ou PDM (*Pulse-Duration Modulation*) et des interfaces génériques telles que USB (*Universal Serial Bus*), I2C (*Inter Integrated Circuit*) et SPI (*Serial Peripheral Interface*). Un microcontrôleur intègre également différentes interfaces de communication supportant des protocoles spécifiques au domaine automobile, comme le CAN, LIN et FlexRay. Ces périphériques peuvent être liés par un deuxième ensemble de bus afin de ne pas surcharger les échanges entre CPU et mémoire.

Un microcontrôleur est caractérisé par différents aspects. La capacité de calcul, exprimée en MIPS (*Mega Instructions Per Second*), d'un microcontrôleur indique le

nombre d'instructions élémentaires qu'un processeur est capable d'exécuter en une seconde. Les microcontrôleurs se basent sur différentes catégories de cœurs de processeurs selon la nature des applications à supporter et des contraintes de coût. Généralement, on catégorise les cœurs de processeurs selon la taille de bus de données. Vue l'augmentation du nombre de fonctions qui sont intégrées dans les systèmes embarqués et l'augmentation de leur complexité, des microcontrôleurs mettant en œuvre plusieurs cœurs de processeurs émergent. En effet, les architectures multi-cœur sont des solutions alternatives aux architectures mono-cœur afin de limiter les fréquences de fonctionnement nécessaires et donc la consommation induite. Afin de subvenir aux besoins de capacités de calcul importantes, de nouvelles tendances visent à employer des ressources matérielles reconfigurables comme les FPGAs (*Field-programmable gate array*). Ces ressources reconfigurables offrent une flexibilité afin de mettre en œuvre des fonctions spécifiques gourmandes en calcul et aussi des modes dégradés de ces fonctions pour assurer une continuité de service [**8**].

Le Tableau 1.1 présente différentes configurations de microcontrôleurs rencontrés dans le domaine automobile. Leurs caractéristiques diffèrent compte tenu du domaine d'application dans lequel ils sont utilisés.

Nom	CPU		mémoire		Interface de communication				Périphériques					
	Type	Fréquence	Flash	RAM	CAN	LIN	Flex Ray	MOST	ADC	PWM	SPI	I2C	PIO	UART
AT90CAN128 (Atmel)	8 bits	16 MHz	128 Ko	4 Ko	oui	non	non	non	oui	oui	oui	oui	oui	oui
Atmega64C1 (Atmel)	8 bits	16 MHz	64 Ko	4 Ko	oui	oui	non	non	oui	oui	oui	non	oui	oui
ST10F276E (ST)	16 bits	64 MHz	832 Ko	68Ko	oui	non	non	non	oui	oui	oui	oui	oui	oui
SPC560P50L5 (ST)	32 bits	64 MHz	576 Ko	40 Ko	oui	oui	oui	non	oui	oui	oui	non	oui	oui
SPC564A80L7 (ST)	32 bits	150 MHz	4 Mo	192 Ko	oui	oui	oui	non	oui	non	oui	non	oui	non
TMS570 (texas instruments)	Duo-Cœurs 32 bits	160 MHz	2 Mo	160 Ko	oui	oui	oui	non	oui	non	oui	non	oui	oui
TMS470 (texas instruments)	32 bits	80 MHz	512 Ko	64 Ko	oui	oui	non	non	oui	non	oui	oui	oui	non

Tableau 1.1 – Caractéristiques des microcontrôleurs selon le domaine de fonctionnement

A la densification des ressources matérielles disponibles au sein des ECUs composant l'architecture électronique des véhicules s'ajoute l'accroissement de la complexité des ressources logicielles embarquées.

1.1.2.3 Architecture logicielle des ECUs

Un nombre croissant d'applications sont intégrées dans les véhicules. Ces applications sont mises en œuvre par un ensemble de ressources logicielles embarquées sur des plates-formes distribuées. Ces ressources logicielles correspondent, par exemple, aux pilotes des interfaces de communication, aux piles protocolaires utilisées ou au système d'exploitation. Classiquement, les ressources logicielles embarquées dépendent fortement des ressources matérielles disponibles au sein de l'ECU. Par conséquent, des efforts importants peuvent

être engendrés afin d'adapter le logiciel pour une configuration d'ECU donnée. Afin de maîtriser les coûts et les temps de développement, il s'est donc avéré nécessaire d'améliorer l'échange et la réutilisabilité des logiciels développés dans le domaine automobile. Dans ce contexte, des consortiums tels qu'OSEK/VDX (*Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen / Vehicle Distributed eXecutive*) [**9**] et AUTOSAR (*AUTomotive Open System Architecture*) [**10**] ont été mis successivement en place afin de proposer aux concepteurs des solutions d'architectures logicielles standardisées pour les calculateurs. Ces solutions permettent de simplifier et d'accélérer la phase de conception et ainsi de minimiser le risque d'erreur de développement. Par la suite, afin de rendre compte de l'organisation des architectures logicielles des ECUs, nous nous appuyons sur l'organisation préconisée par le consortium AUTOSAR.

Le consortium AUTOSAR utilise et étend la spécification d'OSEK/VDX. Initialement, OSEK/VDX spécifiait une organisation en couche des ressources logicielles assurant la gestion des communications entre calculateurs. Il définissait également un ensemble de services requis au niveau du système d'exploitation. Plus largement, AUTOSAR vise à faciliter la distribution des applications sur les différents ECUs. Il a pour objectif de définir un cadre méthodologique pour la spécification des différents constituants d'un système électronique. Ainsi, l'environnement défini par AUTOSAR doit conduire à automatiser certaines phases du développement et à améliorer la réutilisation des ressources logicielles composant un système embarqué. L'architecture logicielle proposée par AUTOSAR vise à favoriser la modularité des développements en définissant différentes couches d'abstraction entre les ressources matérielles et la description de l'application. L'organisation entre ces différentes couches est illustrée dans la Figure 1.5.

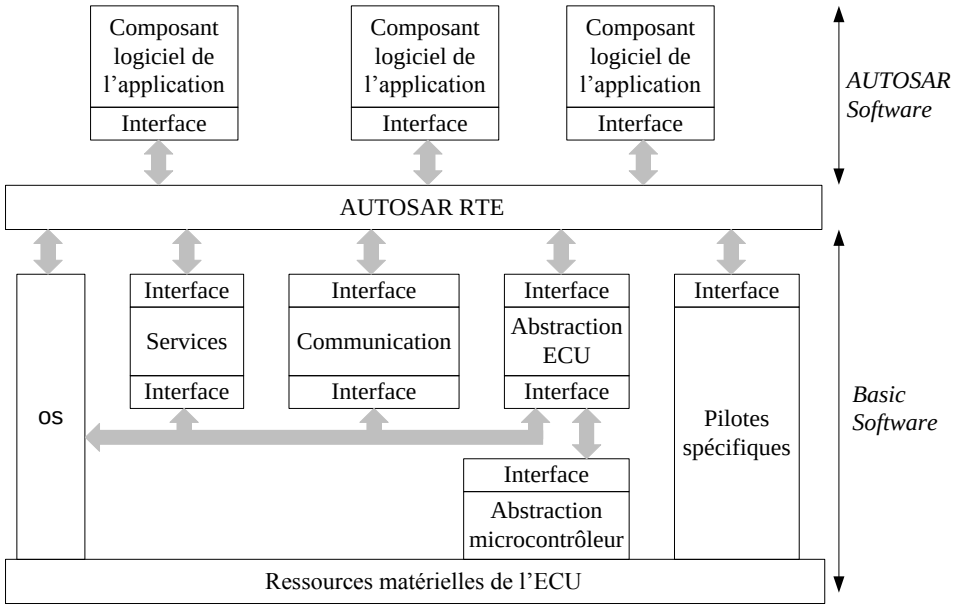


Figure 1.5 – Organisation des couches d'abstraction retenues dans le projet AUTOSAR.

Trois principales couches logicielles sont mises en place par le consortium AUTOSAR. Ces couches sont notées AUTOSAR *Software*, AUTOSAR *Runtime Environment* (RTE) et *Basic Software* (BSW), comme indiqué par la Figure 1.5. La couche AUTOSAR *Software*

est constituée d'un ensemble de composants logiciels qui forment l'application. Ces composants sont basés sur des interfaces. Ces interfaces assurent la connectivité entre l'application et la couche AUTOSAR *Runtime Environment*. La couche *Basic Software* englobe les couches logicielles nécessaires pour l'abstraction des ressources matérielles. La couche BSW contient l'OS (*Operating System*) et des gestionnaires des ressources matérielles de chaque ECU, tels que les pilotes de périphériques et des bus de communication. Il contient également des fonctions gérant les communications entre ECUs, telles que les piles protocolaires du bus CAN, ainsi qu'un ensemble de fonctions de services, comme la gestion de la défaillance de réseaux. Les fonctions logicielles de cette couche sont dotées d'une interface permettant la communication avec la couche RTE. La couche RTE est un lien standard entre les deux couches AUTOSAR *Software* et BSW. Elle permet l'acheminement de toutes les interactions entre les composants de la couche logicielle AUTOSAR et les composants logiciels unitaires de la couche BSW. La couche RTE agit comme un centre de communication pour l'échange d'informations inter et intra ECUs. En effet, pour la communication inter-ECUs, elle fournit une abstraction des communications aux composants logiciels AUTOSAR en leur procurant la même interface et les même services.

Le concept traité par AUTOSAR permet de rendre possible le développement d'applications en utilisant des interfaces et des services standards indépendamment du matériel hôte. Une fois le pilote d'un matériel donné ou d'un standard de communication développé, il sera utilisé pour tout développement d'application qui utilise ce matériel ou ce standard. Ce concept participe à la simplification de la phase de développement logiciel dans la phase de conception des architectures distribuées.

En résumé, cette sous-section illustre la complexité des architectures distribuées embarquées dans le domaine automobile, due à l'augmentation continue du nombre de services rendus au sein des véhicules. Cette complexité réside dans le nombre important des ECUs employés, dans l'usage de différents supports de communication et dans la richesse des ressources logicielles nécessaires. Cette complexité influe directement sur les phases de conception.

1.2 Conception des architectures électroniques dans le domaine automobile

Après nous être intéressés aux caractéristiques des architectures embarquées au sein des véhicules nous abordons dans cette section les démarches adoptées pour leur conception. La conception des architectures s'inscrit dans un processus organisé partant de spécifications initiales pour aboutir à une architecture satisfaisant à un ensemble de contraintes. Des outils et des méthodes existent pour appréhender la conception aussi bien d'un ECU que de l'ensemble des ECUs en réseau.

1.2.1 Processus global de conception et contraintes associées

De façon générale, différents processus existent pour organiser la conception et le développement de produits. Dans le domaine automobile, le cycle en V décrit le cycle de vie d'un système électronique en passant par les différentes phases de conception et de

développement. La Figure 1.6 détaille les différentes étapes identifiées dans le cycle de vie d'un système [**11**].

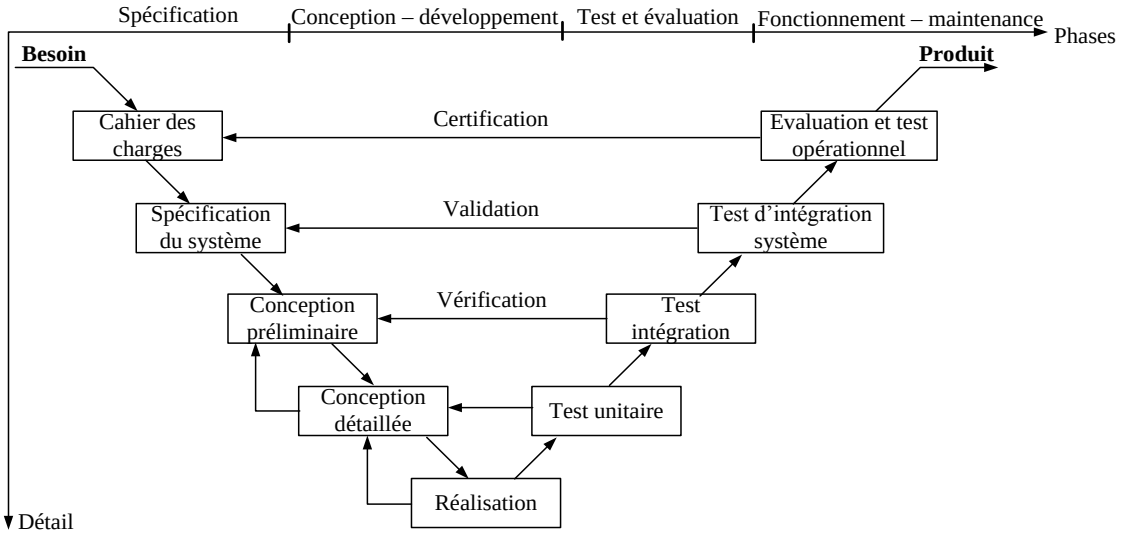


Figure 1.6 – Cycle de vie en cycle en V.

Dans la Figure 1.6, l'axe horizontal présente les différentes phases d'un projet. L'axe vertical présente le niveau de détail dans la description du système. Les phases de spécification et de conception définissent des niveaux de description de plus en plus détaillés. A partir d'une description du besoin utilisateur, un cahier des charges est dressé pour décrire un ensemble d'exigences fonctionnelles attendues par le système à réaliser. Une spécification du système permet de préciser avec exactitude le besoin utilisateur en décrivant les exigences de façon exhaustive, chiffrée et sans ambiguïté. Une première définition des constituants du système est effectuée dans la phase de conception préliminaire. Généralement, cette définition est liée aux aspects fonctionnels. Ensuite, une conception détaillée prend en compte les aspects technologiques pour définir les ressources matérielles et logicielles nécessaires pour le bon fonctionnement du système. La réalisation est l'étape qui consiste au développement matériel et / ou logiciel de la solution. La partie ascendante du cycle en V permet de vérifier la conformité de la réalisation pour chaque niveau de la partie descendante. En effet, à chaque étape de la conception et de la spécification correspond un test de conformité qui est effectué en commençant par les parties les plus élémentaires puis en remontant jusqu'au produit complet.

Parmi les différents acteurs du domaine automobile intervenant dans ce processus on trouve :

- les constructeurs automobiles, comme BMW, Ford, Renault et Toyota, qui fournissent les véhicules et représentent les donneurs d'ordre définissant les spécifications des différents constituants des véhicules ;
- les équipementiers de rang 1, comme Bosch et Siemens, qui fournissent des sous-ensembles des véhicules tels que la gestion du groupe motopropulseur ou la suspension et son contrôle ;

- les équipementiers de rang 2, comme STMicroelectronics et WindRiver qui interviennent sur des constituants élémentaires, matériels et logiciels, tels que les microcontrôleurs et les systèmes d'exploitation temps réel ;
- les fabricants, comme Flextronics et TSMC, qui réalisent certains sous-ensembles via la fabrication des composants et des cartes électroniques.

Ces intervenants sont confrontés à diverses exigences pour concevoir les systèmes électroniques embarqués. Classiquement, les contraintes de conception portent sur les propriétés fonctionnelles et non fonctionnelles des systèmes à concevoir. Les contraintes liées à la fonctionnalité précisent le comportement et les fonctions assurées par le système à concevoir. Les contraintes non fonctionnelles portent sur la façon de réaliser les fonctionnalités attendues et sont, par exemple, le temps de réponse du système, la consommation d'énergie, l'encombrement, la robustesse, la fiabilité et le coût. Un point important du domaine automobile porte sur la fiabilité et la tolérance aux pannes en vue d'assurer la sécurité. Il s'agit d'accorder au système de fonctionner selon plusieurs modes dégradés en prévoyant les scénarios de défaillances possibles qu'il pourra subir. Le système garantit alors un minimum de fonctions vitales pour protéger la sécurité des passagers.

Afin de réduire les efforts et les coûts d'adaptation des solutions développées, différentes contraintes sont appliquées sur le concepteur et portent sur :

- la réutilisation : il s'agit de permettre l'amélioration de la portabilité des développements selon les différentes configurations de ressources ciblées.
- La flexibilité : il s'agit de favoriser la possibilité d'ajout ou de remplacement de ressources matérielles ou logicielles au sein d'une solution existante.
- L'évolutivité : consiste à rendre le système capable d'autoriser des nouvelles versions ou des mises à jour pour la correction de défauts éventuels ou pour l'optimisation de l'exécution.
- L'extensibilité : le système doit être conçu de façon à être capable, en fonctionnement, d'acquérir de nouvelles fonctionnalités ou d'accepter de nouveaux éléments matériels sans remplacer la totalité du logiciel et sans modifier l'infrastructure matérielle.
- La maitrise du coût et du temps de développement et de conception : il s'agit de veiller à réduire le coût de fabrication du système et le coût de la conception et du développement.

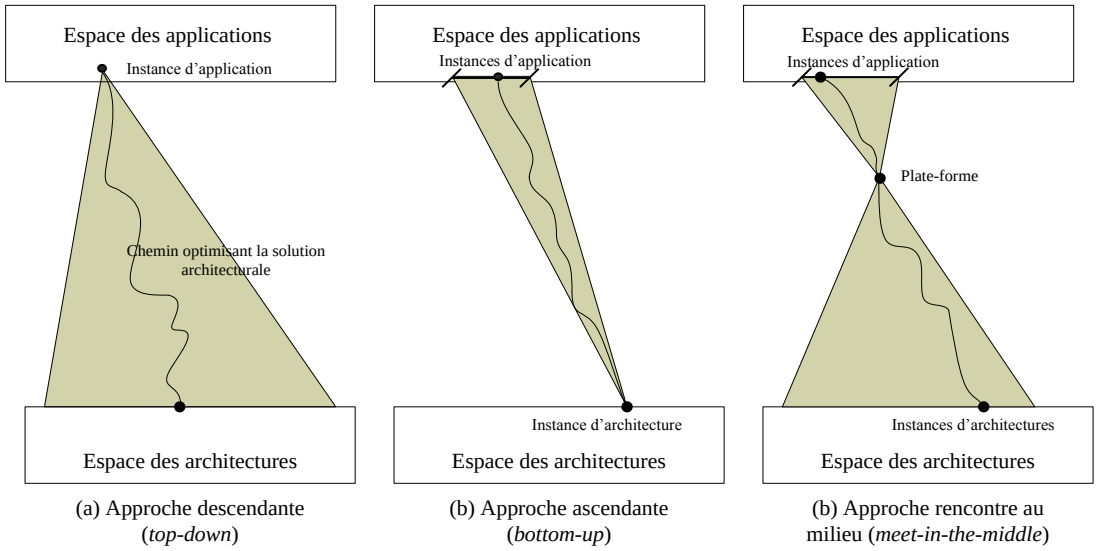
Compte tenu de l'accroissement de la complexité des architectures utilisées dans le domaine automobile et des contraintes appliquées aux concepteurs il est nécessaire de faire évoluer les démarches de conception utilisées.

1.2.2 Approche orientée plate-forme pour la conception des architectures des ECUs

La notion de plate-forme est fréquemment utilisée dans le domaine automobile pour désigner des sous-systèmes pouvant être partagés par plusieurs gammes de véhicules, et ce

afin d'offrir des solutions réutilisables conduisant à une meilleure maîtrise des coûts et des temps de production.

Dans le domaine de la conception de systèmes électroniques, une des solutions apportées afin d'assurer la maîtrise de la complexité de conception réside dans la définition de solutions standardisées. Une plate-forme désigne alors un ensemble de ressources, matérielles ou logicielles, pré-définies à un niveau de précision donné [**12**] [**13**]. Ces ressources peuvent alors être combinées afin de former des ensembles plus complexes et ce en un temps restreint. Une approche orientée plate-forme (*Platform Based Design*) désigne un ensemble de méthodes et d'outils utilisés afin d'associer efficacement les éléments disponibles au sein de la plate-forme. Il est possible de positionner l'approche orientée plate-forme parmi les différentes catégories d'approches de conception possibles. Ces catégories sont illustrées sur la Figure 1.7.



(a) Approche descendante (*top-down*) (b) Approche ascendante (*bottom-up*) (b) Approche rencontre au milieu (*meet-in-the-middle*)

Figure 1.7 – Approches de conception descendante, ascendante, rencontre au milieu.

La première catégorie illustrée correspond à une approche dite descendante (*top-down*). Dans ce cas de figure le concepteur est amené à raffiner progressivement les spécifications initiales jusqu'à l'obtention d'une solution satisfaisant aux contraintes imposées. Dans le cas d'une démarche ascendante (*bottom-up*) l'architecture est progressivement formée par l'association de constituants élémentaires. Enfin, l'approche de rencontre au milieu (*meet-in-the-middle*) correspond au fait de partir d'une spécification initiale pour arriver à un niveau de description de l'architecture réutilisant des ressources définies au préalable et ce pour un niveau d'abstraction donné. Une approche orientée plate-forme est donc typiquement une approche de rencontre au milieu. En favorisant la réutilisation de ressources pré-définies elle conduit à maîtriser les temps de développement. Une telle approche se retrouve déclinée aussi bien pour la conception des ressources matérielles des ECUs que pour la conception des ressources logicielles associées.

La conception des ressources matérielles des ECUs regroupe l'ensemble des activités liées à la conception de la carte électronique et des ressources alors utilisées. La conception des microcontrôleurs consiste en la définition des ressources de calcul, de communication et

de mémorisation nécessaires. Cette conception correspond alors à la réalisation d'un système sur puce (SoC, *System on Chip*) spécifique. La notion de plate-forme s'exprime ici par le fait que les microcontrôleurs utilisés reposent sur des structures génériques qui peuvent ensuite être adaptées en fonction des besoins des applications visées. Comme évoqué précédemment ces structures consistent en l'association de ressources prédéfinies IP (*Intellectual Property*) correspondant à des cœurs de processeurs, des unités de traitement spécifiques, des périphériques dédiés… L'association de ces éléments est d'autant plus facilitée qu'ils se conforment généralement à des protocoles de bus standardisés.

En ce qui concerne la conception des ressources logicielles la notion de plate-forme s'exprime ici par l'existence d'architectures standardisées, favorisant la réutilisation de modules logiciels. Dans le cas de l'architecture AUTOSAR la définition d'une structure standardisée a permis la définition d'outils favorisant la génération automatique de solutions. Un tel processus est illustré sur la Figure 1.8.

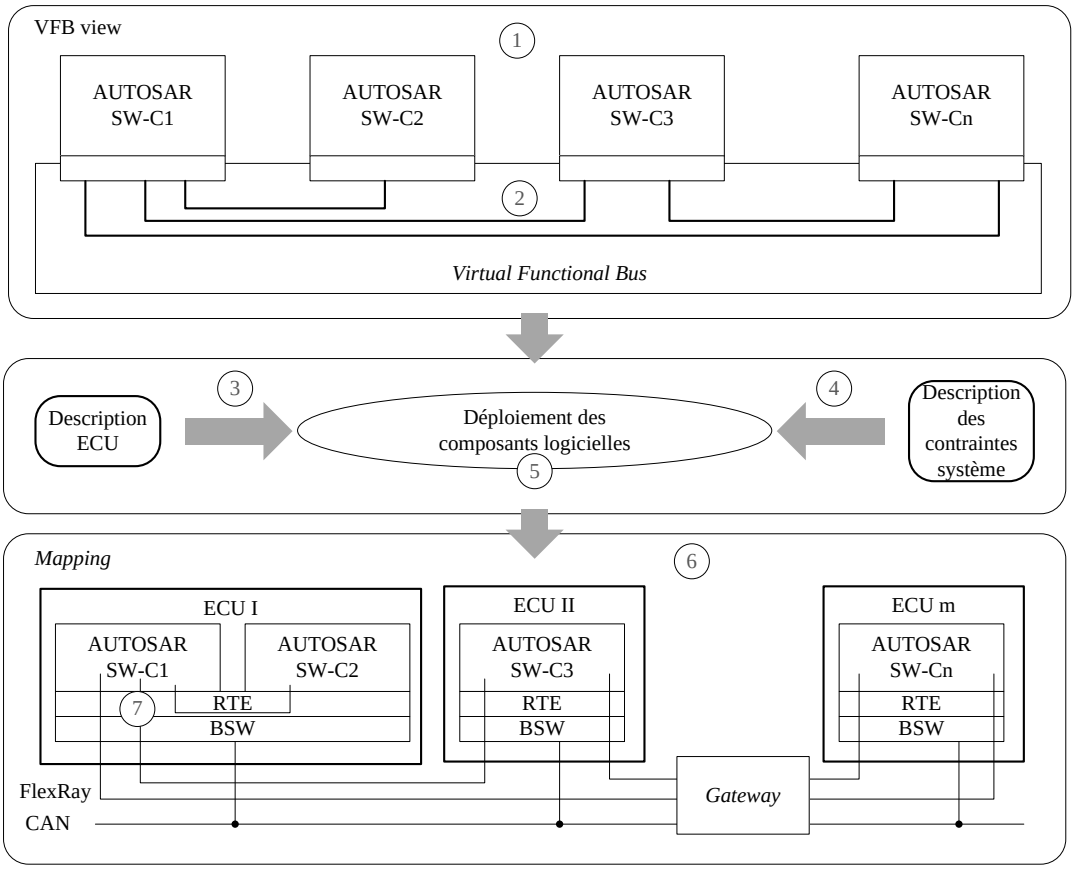


Figure 1.8 –Démarche de conception retenue par le projet AUTOSAR [**10**].

La Figure 1.8 illustre le processus de développement prévu selon l'approche préconisée par AUTOSAR. Cette approche débute par une description de l'application sans prise en compte des considérations technologiques et notamment en ne prenant pas en compte les contraintes de distribution de l'application (1 et 2). L'élément noté *Virtual Functional Bus* (VFB) désigne les relations fonctionnelles abstraites identifiées entre les différents éléments de l'application. L'étape suivante consiste à définir les caractéristiques de chaque ECU (3) ainsi que les contraintes de distribution (4). Enfin, la phase de génération conduit à

l'obtention des ressources logicielles nécessaires et correctement configurées pour chaque ECU (5) et (6). L'étape (7) consiste à traduire les relations entre constituants de l'application compte tenu des mécanismes d'échange supportés au sein d'un même calculateur ou compte tenu des protocoles d'échange supportés entre différents calculateurs. Une telle approche représente un gain en temps de développement particulièrement important.

Un aspect particulier de la conception porte sur la prise en compte et la mise en œuvre des communications entre calculateurs.

1.2.3 Conception des architectures distribuées

La conception et le dimensionnement des réseaux de communication au sein des véhicules consistent à prévoir les ressources nécessaires au sein de l'architecture distribuée. Dans ce processus, il est indispensable d'identifier les besoins de communication de chaque élément. Ces besoins sont identifiés selon différents critères, notamment les valeurs maximales, minimales et moyennes du débit et de la latence, et la nature dure ou souple des contraintes de temps portant sur les communications. Différents paramètres entrent en jeu dans le dimensionnement. Dans un premier temps, le choix du type et du nombre de réseaux, de leurs topologies, de leurs étendus et du nombre d'interfaces de communication est une étape primordiale dans ce processus. Ensuite, en se basant sur l'infrastructure définie, la configuration des protocoles pour chaque réseau définit les débits, les latences, les taux d'erreur, les modes d'accès, les niveaux de priorité des communications et des nœuds, etc.

Actuellement, il existe différents outils qui peuvent être utilisés pour la conception des infrastructures électroniques des véhicules. On peut citer l'outil SymTA/S (*Symbolic Timing Analysis for Systems*) [**14**], outil de chez Symtavision basé sur la simulation de modèles pour concevoir des systèmes temps réel. Il permet l'analyse, l'optimisation et la vérification des performances temporelles. Les analyses effectuées portent sur le respect des contraintes de temps maximales compte tenu de stratégies d'ordonnancement données. Les analyses portent également sur l'influence des communications entre calculateurs sur le respect des contraintes de temps. Pour le domaine automobile un modèle du protocole CAN peut être utilisé. L'outil VNA (*Volcano Network Architect*) [**15**] est un outil de chez *Mentor Graphics* qui permet la conception et le développement d'architectures basées sur les protocoles CAN et LIN. Il permet de définir l'organisation des trames communiquées et l'ordonnancement des communications entre les différents éléments de l'architecture. Il permet de vérifier le respect des exigences sur les délais de transmission de messages et leurs éventuelles pertes. On peut également citer certains environnements moins spécifiques au domaine automobile et qui peuvent être utilisés afin de considérer des couches protocolaires de plus haut niveau. On peut ainsi citer l'outil OPNET [**16**] qui est un outil permettant de décrire, de simuler et de dimensionner des réseaux de communication locaux et étendus. Il est possible de créer en plus des librairies existantes de nouveaux protocoles et de les personnaliser. NS2 (*Network Simulator version 2*) [**17**] est un outil de simulation au niveau paquet de données pour des réseaux filaires ou sans fil supportant des protocoles comme le TCP (*Transmission Control*

Protocol), UDP (*User Datagram Protocol*), FTP (*File Transfer Protocol*) et HTTP (*Hypertext Transfer Protocol*).

La connexion des ECUs à un ou plusieurs réseaux impose l'intégration de ressources logicielles spécifiques afin de gérer ces communications. L'aspect distribué des architectures électroniques impose une structure logicielle spécifique qui prend en compte les échanges d'informations entre les ECUs. Des protocoles de plus en plus avancés, comme TCP et IP, sont employés. Différentes couches logicielles assurent la communication entre les ECUs. Ces couches sont normalisées afin de faciliter le développement et d'abstraire la gestion de la communication. Par exemple, le rôle de ces couches logicielles est de choisir le réseau de communication qui va être utilisé, ou de piloter le contrôleur de communication gérant l'accès au réseau considéré. AUTOSAR a permis de standardiser ces couches dans une structure logicielle spécifique comme le montre la Figure 1.9.

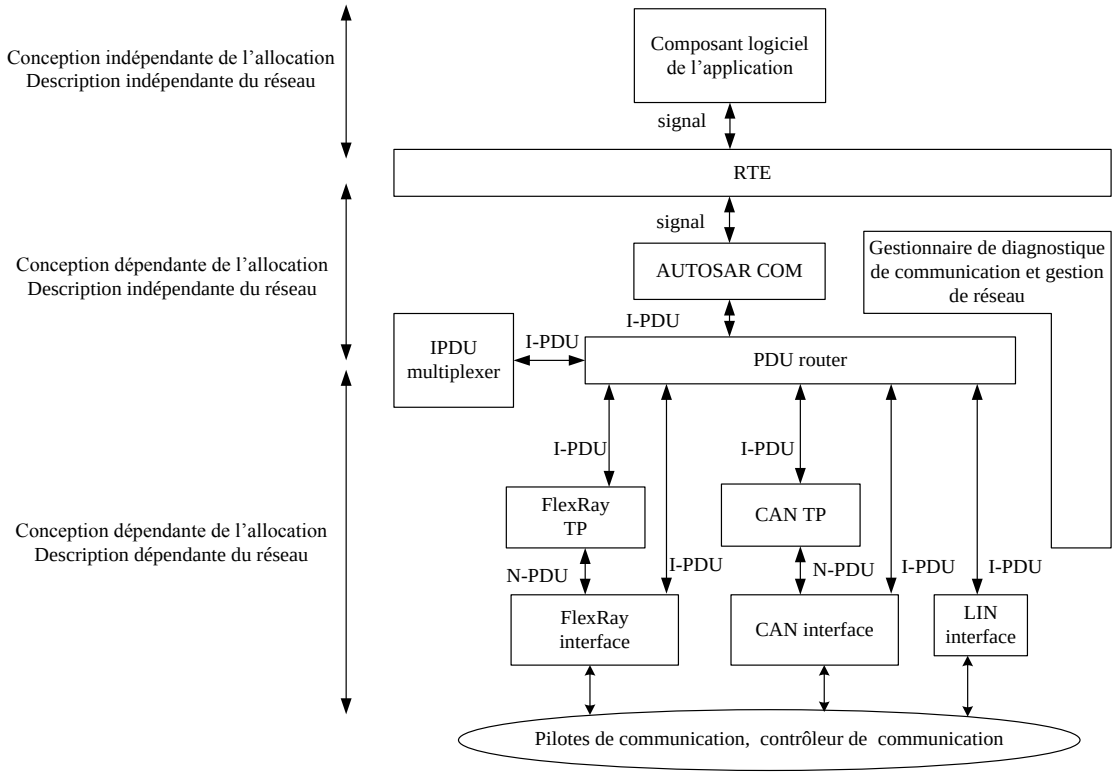


Figure 1.9 – Organisation des couches logicielles pour la communication entre calculateurs.

AUTOSAR identifie trois types de communication au sein d'un ECU. La communication entre les composants logiciels au niveau application est assurée par des signaux (*signal*). Les I-PDUs (*Interaction Layer Protocol Data Unit*) forment un ensemble de signaux. Les N-PDUs (*Data Link Layer Protocol Data Unit*) représentent les données à transmettre via le réseau. L'objectif des différentes couches présentées dans la Figure 1.9 est de gérer l'émission et la réception d'informations selon un protocole spécifique de communication. La couche AUTOSAR Com est le module logiciel qui gère le comportement global de la pile de communication. Cette couche supporte différents mécanismes d'émission/réception de messages. La couche PDU Router assure l'orientation des informations en provenance de la couche AUTOSAR Com sur les différents bus de

communication disponibles. Elle assure également un rôle de passerelle permettant d'effectuer des transferts d'un bus à un autre sans intervention des couches de plus haut niveau. D'autres couches logicielles sont dépendantes des bus de communication considérés. Les couches de protocole TP (*Transport Protocol*) supportent des mécanismes de segmentation de données et de contrôle de flux. Ils sont nécessaires pour les protocoles CAN et FlexRay. L'interface de bus met en œuvre la couche d'abstraction des contrôleurs de bus. Les couches pilotes et contrôleur de communication assurent l'accès au bus de communication. Ces différentes couches sont mises en œuvre pour les protocoles LIN, CAN et FlexRay.

Après avoir abordé différents aspects du processus de conception des architectures embarquées au sein des véhicules nous décrivons dans la section suivante les problématiques considérées dans le cadre de nos travaux.

1.3 Problématiques liées à la conception et au dimensionnement des architectures distribuées dans le domaine automobile

L'accroissement de la complexité des architectures embarquées dans le domaine automobile se traduit par la densification des ressources matérielles et logicielles mises en œuvre afin de supporter des fonctionnalités de plus en plus avancées. La nature distribuée de ces architectures implique une interdépendance forte qu'il est nécessaire de pouvoir prendre en considération tout au long du processus de conception. Cet aspect complexifie chacune des différentes activités de conception et de test. Ces différentes activités peuvent être représentées par la Figure 1.10.

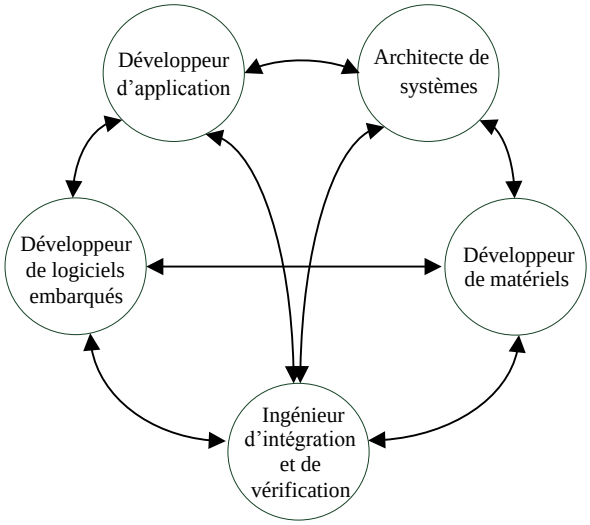


Figure 1.10 – Positionnement des différentes activités de conception.

La Figure 1.10 positionne les différentes catégories de métiers intervenant dans le processus de conception des architectures embarquées. Jusqu'à présent nous avons expliqué l'évolution des ressources matérielles et logicielles au sein des architectures. Cette évolution conduit à complexifier le travail des développeurs de ces ressources. Comme évoqué précédemment, la complexité du processus de mise au point des ressources matérielles et logicielles justifie l'emploi de plates-formes standardisées afin de favoriser la réutilisation et

le partage entre acteurs du développement. La définition de plates-formes conduit également à la définition d'outils permettant d'automatiser certaines étapes du développement. Le développeur d'application a, lui, pour rôle de mettre au point les applications nécessaires au sein des différents domaines composant les véhicules. Ces développements sont le plus souvent réalisés dans des environnements tels que Matlab/Simulink, adaptés pour la mise au point d'algorithmes avancés. Comme évoqué précédemment, ces développements sont faits de manières indépendantes de toutes considérations technologiques, notamment liées à la distribution de l'application sur différents ECUs. Le rôle de l'architecte de systèmes consiste à dimensionner les caractéristiques des ressources matérielles et logicielles nécessaires compte tenu des besoins applicatifs et des différentes contraintes imposées. Dans le cas d'une architecture distribuée, le dimensionnement d'une architecture formée de plusieurs calculateurs conduit à devoir prendre en compte la topologie des réseaux, la configuration d'un protocole de communication sur le fonctionnement de l'architecture et la sélection des ressources nécessaires. Le développeur de matériel a pour rôle de concevoir les éléments constituant les plates-formes matérielles requises. Le développement des ressources logicielles du système consiste à déployer l'application sur les plates-formes définies. Enfin, le rôle de l'intégrateur est d'effectuer l'association des différents constituants d'une architecture. Jusqu'à une époque récente, il était possible de concevoir indépendamment chacun des ECUs constituant une architecture. L'intégration au sein d'une architecture commune était alors considérée lors des phases finales de mise au point et de test. Dès lors que ces ECUs sont devenus fortement interdépendants, il s'avère indispensable de pouvoir prendre en compte au plus tôt lors des phases de conception les contraintes induites par les échanges inter-ECUs.

Ainsi, la maîtrise de la complexité des architectures du domaine automobile réside dans la capacité des concepteurs à pouvoir prendre en considération l'ensemble des caractéristiques de ces architectures et ce au plus tôt dans le processus de conception. Dès lors, il s'avère indispensable de disposer de moyens permettant de valider au plus tôt les choix et les développements effectués. Un moyen de favoriser ce travail consiste à disposer d'un ou de plusieurs modèles de référence permettant aux différentes activités de conception d'interagir. Ce principe est illustré sur la Figure 1.11.

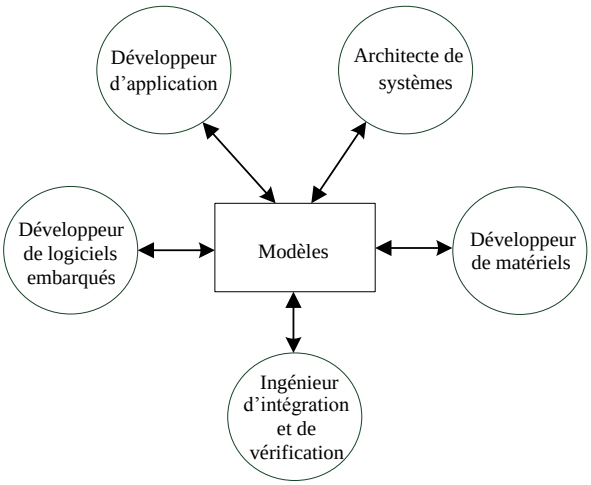


Figure 1.11 – Utilisation de modèles de référence entre les différentes activités de conception.

Le fait de disposer de modèles de référence exprimant, avec différents niveaux de détail, les caractéristiques de l'architecture à concevoir favorise le travail et les échanges entre les différents acteurs du processus de conception. Par exemple, la notion de prototype virtuel désigne une description abstraite des ressources matérielles d'une architecture. Il est ainsi possible au développeur de ressources logicielles d'analyser l'efficacité des solutions développées sur la base d'un tel prototype et donc sans nécessairement devoir attendre la disponibilité d'un prototype réel.

Dans ce contexte, l'activité d'architecte de systèmes joue un rôle essentiel en définissant les ressources nécessaires compte tenu des allocations possibles de l'application sur les différents calculateurs. L'enjeu consiste alors à pouvoir définir au plus tôt dans le processus de conception les caractéristiques des architectures compte tenu des différentes alternatives possibles. A cette fin, les modèles utilisés se doivent de capturer les caractéristiques de chaque calculateur, mais également l'influence des différents protocoles de communication considérés au sein d'une architecture distribuée.

Ainsi, les problématiques associées portent sur la création de modèles d'architectures qui soient suffisamment efficaces pour favoriser le travail des architectes. Il s'agit alors de pouvoir évaluer les performances de différentes architectures en un temps restreint. Compte tenu de la complexité des architectures considérées, l'efficacité porte tout d'abord sur le respect d'un bon compromis entre la rapidité d'obtention de résultats à partir de l'exécution des modèles utilisés et de la précision de ces résultats. L'efficacité porte également sur le temps de création de tels modèles d'architectures. Il s'agit alors de favoriser l'obtention de tels modèles de l'architecture et d'en réduire le temps de création.

Les différentes contributions apportées dans le cadre de cette thèse visent à répondre à ces problématiques. Ces contributions portent sur la définition de méthodes visant à favoriser la création de modèles efficaces d'architectures en vue de leur dimensionnement sous contraintes. Ces contributions sont développées dans la suite de ce document. Elles ont été appliquées au cas des architectures considérées pour le projet CIFAER.

Chapitre 2
Etat de l'art sur les approches de dimensionnement des architectures de systèmes embarqués

Sommaire

Ce chapitre vise à positionner nos travaux parmi les activités existantes dans le domaine du dimensionnement des architectures de systèmes embarqués.

La première section de ce chapitre présente les différentes catégories d'approches de dimensionnement ainsi que les modèles couramment utilisés pour décrire et étudier les architectures des systèmes embarqués. La seconde section présente plus spécifiquement différentes approches existantes de dimensionnement, vis-à-vis desquelles nous nous positionnons. La troisième section détaille plus particulièrement la notion de modélisation transactionnelle, servant de base à la création de modèles exécutables d'architectures. Cette partie évoque différents travaux menés en vue de l'amélioration du compromis entre rapidité d'exécution des modèles et précision des résultats. Enfin, la quatrième section positionne les contributions apportées dans le cadre de nos travaux.

2.1 Catégories de méthodes de dimensionnement des systèmes embarqués

2.1.1 Principes et définitions

Le dimensionnement de systèmes embarqués consiste à définir l'organisation et les propriétés des différents constituants d'une architecture. On entend ici par architecture l'association faite des ressources de la plate-forme et des éléments constituants l'application. Ces éléments correspondent aux fonctions composant l'application et aux relations entre ces fonctions. Le dimensionnement consiste donc à définir les ressources matérielles et logicielles nécessaires à mettre en œuvre compte tenu de l'association considérée. La définition des ressources matérielles consiste à caractériser les propriétés des ressources de calcul, de mémorisation et de communication de la plate-forme vis à vis des contraintes fonctionnelles et non-fonctionnelles imposées. On désigne par ressources de calcul des éléments tels que des cœurs de processeurs d'usage générique ou spécifique (GPP (G*eneral-Purpose Processor*), DSP (*Digital Signal Processor*)) ou des blocs matériels dédiés correspondant à des accélérateurs de calcul. Les ressources de mémorisation correspondent à l'ensemble des ressources permettant de stocker les données et les programmes nécessaires. Les ressources de communication désignent ici les éléments assurant les échanges entre les ressources de calcul. La définition des ressources logicielles consiste à caractériser les éléments nécessaires à l'exécution des fonctions de l'application sur les ressources matérielles de la plate-forme. Cette définition dépend notamment de la distribution des fonctions sur les ressources de calcul disponibles. Il s'agit également de déterminer l'usage ou non de systèmes d'exploitation temps réel et de définir l'ordonnancement et les priorités relatives des tâches qui seront exécutées par les processeurs.

Face à l'augmentation de la complexité de conception, le dimensionnement de l'architecture est de plus en plus effectué au plus tôt dans le processus de conception, dès la phase de spécification. L'apport du dimensionnement suffisamment tôt dans le processus de développement consiste à pouvoir mesurer l'incidence des différents choix de conception de l'architecture, des choix inappropriés pouvant nuire au bon fonctionnement de l'architecture. Le dimensionnement comprend généralement deux étapes. La première consiste à l'évaluation des performances d'une architecture donnée, pour des conditions de fonctionnement données. On entend par performances l'ensemble des critères permettant de juger de l'efficacité d'une architecture vis-à-vis de contraintes spécifiques à satisfaire. Par exemple, ces contraintes peuvent porter sur le respect de contraintes de temps, le coût de la solution choisie, l'encombrement ou la consommation. La deuxième étape du processus de dimensionnement consiste à explorer l'espace de conception. Cette exploration permet d'identifier un ensemble de solutions possibles respectant les contraintes considérées. Ces solutions sont progressivement évaluées et comparées afin de déterminer une solution satisfaisant au mieux aux critères considérés.

L'évaluation des performances et l'exploration de l'espace de conception nécessitent de disposer d'une représentation fidèle de l'architecture. Classiquement, la représentation de

l'architecture est obtenue selon une approche en Y [**18**]. Le principe de cette approche est basé sur la combinaison de la description de l'application et de la description de la plate-forme afin de former un modèle de l'architecture. Le principe de cette approche est présenté dans la Figure 2.1.

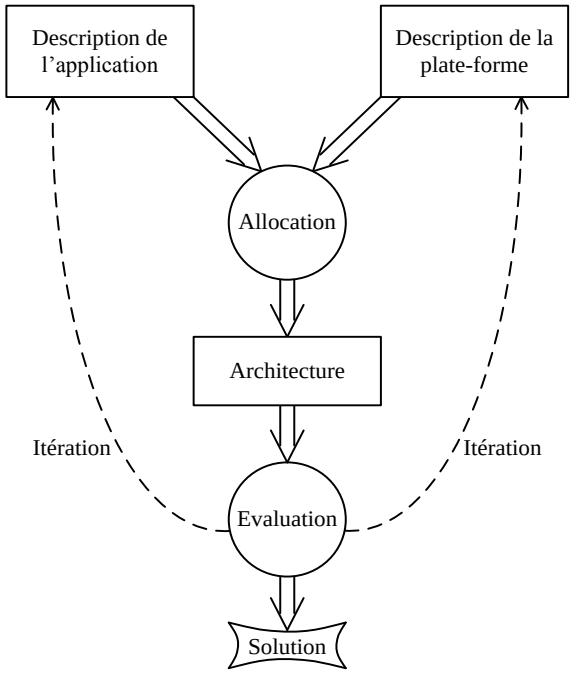


Figure 2.1 – Utilisation du flot en Y pour le dimensionnement des architectures.

Dans cette approche, la description de l'application et la description de la plate-forme sont effectuées séparément. Ensuite, la phase d'allocation définit le lien entre les éléments composant l'application (fonctions et relations entre fonctions) et les ressources composant la plate-forme. La description obtenue alors modélise l'architecture et peut être analysée afin de vérifier s'il s'agit d'une solution satisfaisant aux contraintes considérées. L'exploration de l'espace de conception demande plusieurs itérations en considérant différentes allocations et différentes descriptions de l'application ou de la plate-forme. Ce processus permet d'obtenir un ensemble de solutions potentielles, parmi lesquelles le concepteur pourra identifier la solution la plus optimisée compte tenu des différents critères à respecter.

Il existe différentes méthodes permettant d'effectuer l'évaluation des performances des architectures et de mener l'exploration de l'espace de conception.

2.1.2 Méthodes d'évaluation et d'exploration

L'évaluation des performances des architectures vise à déterminer l'utilisation faite des ressources de calcul de communication et de mémorisation lors de l'exécution de l'application. Cette évaluation est faite compte tenu d'un scénario de fonctionnement donné de l'environnement au sein duquel opèreraient les architectures étudiées. Nous reprenons ici la classification faite dans [**19**] afin de présenter les différentes catégories d'approches existantes pour évaluer les performances des architectures de systèmes embarqués.

Une première catégorie d'approches consiste à utiliser des références existantes (souvent qualifiées de *benchmark*) servant à qualifier une plate-forme particulière pour une ou différentes applications possibles. Ces références peuvent être fournies par le constructeur de la plate-forme. Elles peuvent également être obtenues par des mesures obtenues par des exécutions spécifiques. De telles références peuvent être obtenues pour différents domaines d'application tels que l'embarqué [**20**], le multimédia [**21**] ou les ordinateurs haute performance [**22**]. Ces références comportent généralement une description de l'application pour une plate-forme donnée et qualifient l'utilisation faite des ressources de la plate-forme pour des conditions spécifiques de fonctionnement. Différents critères peuvent être considérés : le temps d'exécution, le nombre d'opérations, l'occupation mémoire, la consommation … Ces références servent le plus souvent de base afin de qualifier certains des constituants de base d'une architecture. Il s'agit alors de compléter ces références par une analyse des interactions entre constituants au sein de l'architecture.

Une autre catégorie d'approches, qualifiée d'analytique dans [**19**], consiste à analyser une description existante d'une application afin d'identifier les caractéristiques requises des ressources de la plate-forme d'exécution, et ce compte tenu de contraintes données. Un premier aspect porte sur l'analyse statique de codes existants (*static profiling*) visant à extraire des informations sur la complexité d'un algorithme donné ou à estimer des temps d'exécution. Le temps maximal d'exécution, ou exécution au pire cas, est obtenu en analysant les différentes séquences possibles d'instructions au sein de l'algorithme. Un second aspect porte sur l'analyse de l'organisation des fonctions de l'application afin d'estimer les caractéristiques temporelles associées. De nombreux travaux ont ainsi été menés afin de permettre de déterminer les temps d'exécution au pire cas d'applications sur des plates-formes monoprocesseurs ou multiprocesseurs, et ce selon différentes stratégies d'ordonnancement de tâches. Ces différentes approches supposent de définir une représentation formelle du système étudié. Ceci fait, ces approches permettent une comparaison rapide de différentes solutions mais peuvent s'avérer pessimistes par rapport à la réalité.

Les approches basées sur la simulation consistent à exécuter une description de l'architecture pour un jeu donné de stimuli. Ces simulations permettent une observation de l'utilisation faite des ressources pour différents modes possibles de fonctionnement de l'architecture. La description des architectures peut être faite à différents niveaux de précision, conduisant à différents compromis entre la précision des estimations et la rapidité de simulation. De telles approches sont particulièrement adaptées afin d'analyser des phénomènes dits émergents. Ces phénomènes proviennent de l'association des différents constituants de l'architecture et de la description faite de l'environnement de fonctionnement. Ces approches supposent une description exécutable de l'architecture. La création d'une description complète de l'architecture peut alors se révéler fastidieuse, car cela suppose de définir la description de l'ensemble de l'application et de la plate-forme.

Afin de faciliter l'obtention de modèles exécutables, il est commun de combiner les différentes catégories d'approches. De telles approches sont dites hybrides dans [**23**]. Le principe de ces approches consiste à identifier des éléments de l'architecture étudiée pour lesquels les propriétés peuvent être caractérisées statiquement. Ces caractéristiques sont

alors incorporées au sein de modèles simulables. Il est ainsi possible de simuler les architectures sans avoir à disposer de la description complète de l'ensemble des constituants. De telles approches hybrides conduisent à limiter les temps de création des modèles et à réduire les temps de simulation.

Ces différentes catégories d'approches peuvent être comparées selon les critères de précision des estimations et de vitesse d'obtention des résultats. Cette comparaison est illustrée sur la figure suivante, issue de [**19**].

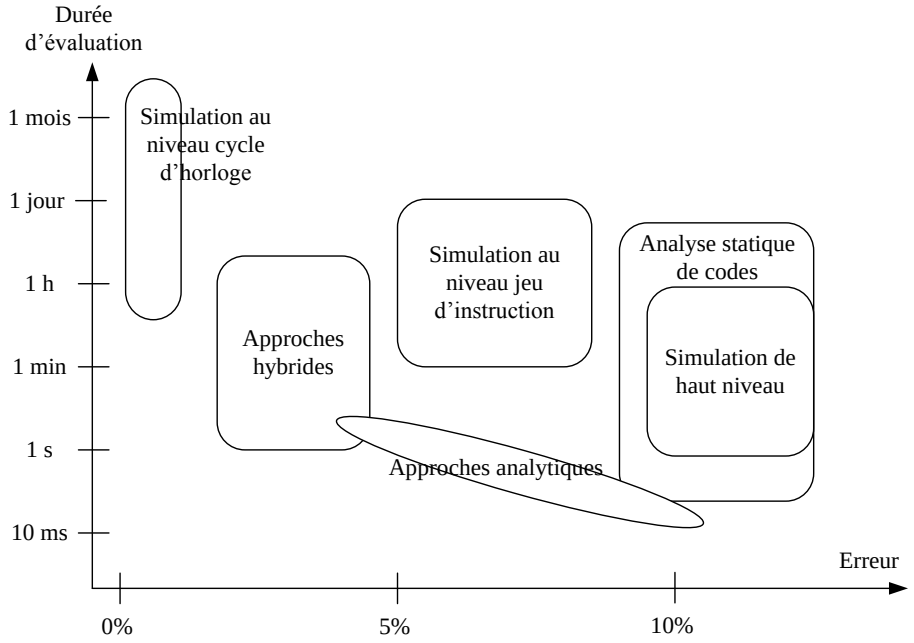


Figure 2.2– Compromis entre durée d'évaluation et précision des résultats de différentes méthodes.

La Figure 2.2 positionne les différentes catégories de méthodes selon les critères de durée d'évaluation et de précision des résultats. La durée d'évaluation s'entend par le temps nécessaire pour évaluer une architecture. Cette durée intègre également les temps mis pour établir les spécifications de l'architecture étudiée et les mettre à jour afin de considérer différentes architectures. Les approches basées sur la simulation se positionnent selon le niveau de précision considéré pour décrire l'architecture. Trois niveaux sont ici identifiés : un niveau précis au cycle d'horloge près, un niveau précis au niveau des instructions exécutées par l'architecture et enfin un niveau supplémentaire qualifié de haut niveau ou *system-level* dans [**19**]. Par la suite, dans la section 2.3, nous précisons les différents niveaux de représentation qui peuvent être considérés pour décrire les architectures de systèmes embarqués. Comme évoqué précédemment, les approches hybrides intègrent des détails supplémentaires par rapport aux approches de simulation de haut niveau. Ces détails sont obtenus par analyse de certains aspects de l'architecture ou par simulation au niveau cycle d'horloge ou au niveau jeu d'instruction. Les méthodes d'analyse statique de codes permettent d'estimer rapidement les temps d'exécution des algorithmes, mais la définition des modèles associés peut s'avérer longue. Les méthodes d'analyse statique de codes sont moins précises que des méthodes basées sur la simulation au niveau du jeu d'instruction car elles font abstraction de différents détails tels que, par exemple, la gestion de la mémoire

cache du processeur. Une telle classification illustre le compromis pouvant exister entre temps de simulation et précision des résultats obtenus.

L'exploration de l'espace de conception consiste à identifier les architectures potentielles pouvant satisfaire à un ensemble de contraintes données. Cette identification est un processus itératif, conduisant à comparer les performances de l'ensemble des solutions possibles. L'exemple de la Figure 2.3 illustre le principe d'une telle identification pour 6 architectures.

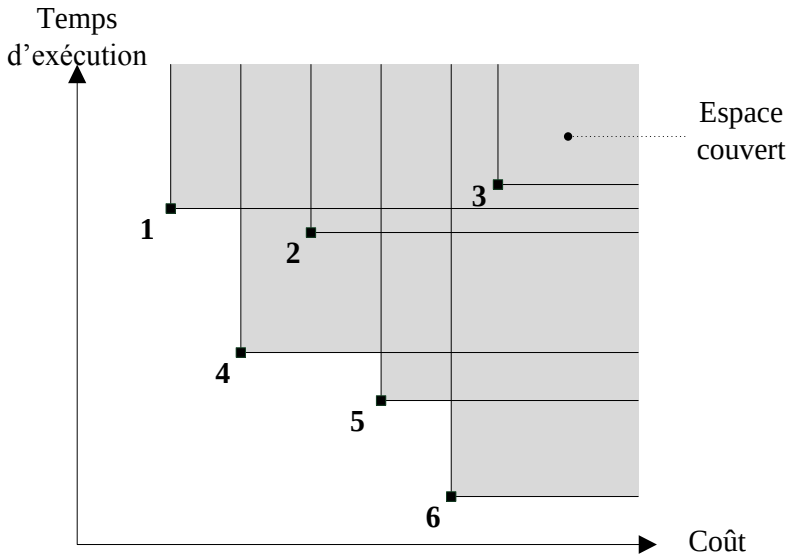
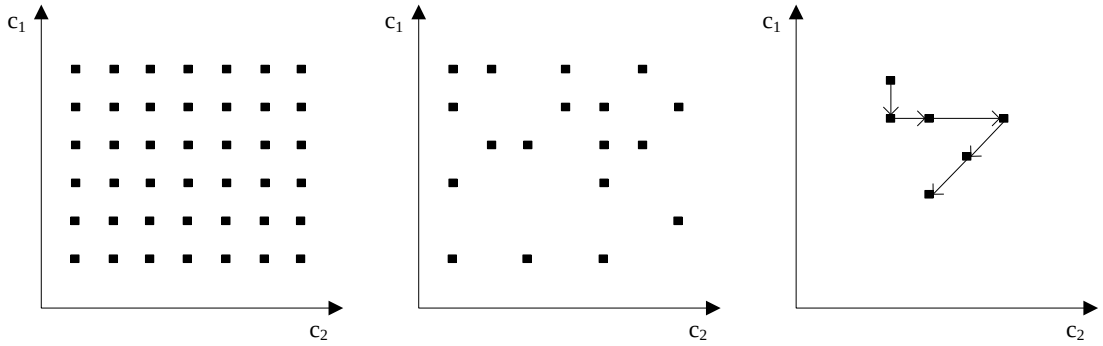


Figure 2.3 – Positionnement de six architectures au sein d'un espace de conception organisé selon les critères de coût et de temps d'exécution.

Sur la Figure 2.3, l'espace des architectures possibles est organisé selon deux critères. L'axe horizontal organise les architectures selon le critère du coût de développement. L'axe vertical exprime le temps d'exécution de l'application sur les ressources de la plate-forme. Chacune des six architectures identifiées délimite un espace de solutions possibles. L'exploration consiste donc à identifier les architectures permettant de délimiter ces espaces et à les comparer entre elles. Dans l'exemple de la Figure 2.3 les solutions 1, 4, 5 et 6 surpassent les solutions 2 et 3 sur les critères retenus.

Différents critères peuvent être retenus afin d'organiser l'espace de conception. On peut notamment citer : le coût de la solution, la consommation, la surface de silicium utilisée, la vitesse d'exécution… Différentes stratégies de recherche peuvent alors être considérées afin de parcourir l'espace des solutions possibles. Le plus souvent le concepteur est amené à ne retenir et à ne comparer que certaines solutions possibles. Afin de converger rapidement vers une solution optimisée, différentes techniques de recherche peuvent être utilisées afin de limiter l'espace des solutions. La Figure 2.4 illustre le principe de ce processus d'exploration.



(a) recherche exhaustive au sein de l'espace de conception

(b) échantillonnage de l'espace de conception

(c) exploration guidée de l'espace de conception

Figure 2.4 – Stratégies d'exploration de l'espace de conception.

Sur la Figure 2.4, l'espace représenté comprend deux critères notés c_1 et c_2 ainsi qu'un ensemble de solutions possibles. Une première approche, (a), consiste à identifier l'ensemble des solutions possibles et à les comparer systématiquement. Une seconde approche, (b), consiste à ne considérer que certaines solutions retenues de manière aléatoire parmi les solutions possibles. Enfin, une troisième catégorie, (c), consiste à converger vers une solution optimisée en évaluant l'effet des modifications successives de l'architecture sur le respect des critères c_1 et c_2. Différentes stratégies de recherche peuvent alors être adoptées afin de converger le plus rapidement possible vers une solution pertinente. Différentes stratégies sont évoquées dans [**19**].

L'évaluation des performances et l'exploration d'espace de conception suppose la création de modèles afin de capturer les propriétés des architectures étudiées. La sous-section suivante évoque les modèles couramment utilisés afin de décrire les architectures des systèmes embarqués.

2.1.3 Modèles pour la description des architectures des systèmes embarqués

Comme évoqué précédemment, un système embarqué peut être perçu selon trois vues distinctes : application, plate-forme et architecture. La modélisation associée à chaque vue consiste à créer une représentation intermédiaire et partielle permettant d'exprimer certaines caractéristiques portant sur le système étudié. Ces caractéristiques portent sur différents aspects parmi lesquels on peut identifier :

- les propriétés temporelles : elles expriment les contraintes temporelles associées à l'exécution du système étudié.
- Le comportement : il exprime les états significatifs du système ainsi que les conditions de transitions entre états.
- Le calcul : il exprime l'ensemble des traitements sur des données assurées par le système.
- La concurrence : elle exprime le niveau de parallélisme assuré par le système.

- Les communications et synchronisations : elles définissent les relations de dépendances et les méthodes d'échange entre les éléments structurant le système.

De nombreuses catégories de modèles existent afin de capturer les diverses propriétés des systèmes embarqués. Selon la nature des propriétés à exprimer et à étudier une ou plusieurs catégories de modèles peuvent être considérées. Par la suite nous évoquons certaines de ces catégories. Une description plus détaillée peut être trouvée dans [**24**] [**25**] [**26**].

Parmi les propriétés essentielles dans la modélisation des systèmes embarqués, la prise en compte des propriétés temporelles se révèle particulièrement importante. Cette description peut se faire essentiellement selon quatre niveaux : une représentation dite causale, à temps continu, à temps discret ou à événement discret. La représentation causale exprime uniquement les relations d'ordre lors de l'exécution du système. La représentation à temps continu considère une évolution permanente au cours du temps du système considéré. La représentation discrète du temps peut se faire en considérant une discrétisation régulière ou irrégulière. Dans le cas d'une représentation irrégulière, on parlera alors de représentation événementielle pour indiquer le fait que la représentation se focalise sur les événements strictement nécessaires à l'évolution du système à décrire. Dans la suite de ce chapitre nous évoquerons plus particulièrement l'emploi de simulateurs à événement discret pour la simulation des architectures.

Une autre propriété essentielle réside dans la description des mécanismes d'échange identifiés. Plusieurs modèles de communication peuvent être identifiés : par passage de messages, par variable partagée, par poignée de main. Ces différents modèles peuvent être précisés pour identifier des modes spécifiques : communication par FIFO, par rendez-vous, communications synchronisées ou non … Le plus souvent, dans la modélisation des architectures, on cherchera à dissocier la modélisation des mécanismes de communication de la modélisation des calculs liés aux transformations de données échangées.

Les systèmes embarqués sont des dispositifs dits réactifs, assurant un ensemble de fonctionnalités en interaction avec leur environnement. Leurs architectures associent des ressources pour permettre le contrôle et le traitement des informations échangées. Parmi les formalismes existants, les modèles à base d'automates à états finis sont particulièrement utilisés afin d'exprimer la séquence d'états au sein d'un système. L'évolution est alors exprimée selon les conditions de transitions entre états, qui portent le plus souvent sur les relations d'entrée. Dans le cas d'automates avec chemin de données, il est également possible d'exprimer l'évolution des relations de sorties. Parmi les différents formalismes d'automates existants, Statechart [**27**] correspond à une notation particulièrement répandue d'automates hiérarchiques, comportant une sémantique adaptée à la représentation de systèmes embarqués (hiérarchie, concurrence, communication). Une extension de ce formalisme correspond aux CFSMs (*Codesign Finite State Machine*) utilisés dans l'approche Metro [**28**].

Parmi les formalismes utilisés afin de rendre compte de la concurrence et des synchronisations au sein d'un système embarqué les réseaux de Pétri offrent un cadre

particulièrement approprié. Ces modèles permettent d'exprimer l'évolution de différents processus concurrents tout en tenant compte des relations de synchronisation existantes. Par ailleurs, il existe des extensions de ces modèles permettant l'expression des propriétés temporelles.

Enfin, afin de rendre compte des flots de données importants présents dans certaines catégories de systèmes embarqués, les modèles dits flots de données sont utilisés. Ces modèles permettent de rendre compte de la structure d'un système et des couplages entre éléments. Ces modèles supportent une sémantique précise quant au comportement des différents éléments structurels. Ainsi, ces modèles consistent en un ensemble de nœuds (ou acteurs, *actors*), reliés par des arcs. Chaque nœud exprime la consommation et la production d'un ensemble prédéfini de jetons échangés via les arcs. Parmi ces modèles on trouve les réseaux de processus Kahn [**29**], ou les réseaux flots de données synchrones (SDF, *Synchronous Data Flow*) [**30**]. De tels modèles sont particulièrement adaptés pour décrire des applications du domaine du traitement numérique du signal.

Ces différentes catégories de modèles peuvent être utilisées pour décrire des architectures en vue de leur dimensionnement. La section suivante présente différentes approches récemment proposées afin de mener le processus d'évaluation des performances des architectures de systèmes embarqués.

2.2 Approches de modélisation et de dimensionnement existantes

Pour chaque approche présentée, les modèles d'architectures utilisés sont analysés ainsi que les méthodes de création de ces modèles. Les différentes approches présentées se distinguent selon la nature des modèles utilisés afin de capturer les propriétés des architectures étudiées.

L'approche SPADE (*System Level Performance Analysis and Design Space Exploration*), présentée dans [**31**], permet l'exploration de l'espace de conception d'architectures pour des applications du domaine du traitement du signal. L'approche SPADE s'appuie sur l'approche en Y afin de créer un modèle d'architecture. Le modèle alors créé est exécutable et permet ainsi d'évaluer par simulation les performances obtenues. Cette évaluation s'appuie sur un modèle de charge de travail (*workload*) associé à l'application. Le modèle de charge décrit la charge de traitement et de communication imposée par l'application sur les ressources de la plate-forme. Ce modèle de charge s'exprime par un ensemble de traces de l'application. Ces traces comportent des informations sur les opérations de communication et de calcul induites par l'exécution de l'application. Dans l'approche SPADE ces informations sont obtenues à partir d'une exécution préalable de l'application. Les traces sont ensuite utilisées pour la simulation selon une technique dite « *trace-driven* ». Cette technique consiste à ce que, lors de la simulation, les traces guident l'utilisation des ressources de la plate-forme lors de l'exécution de l'application sur celle-ci. Ainsi, les caractéristiques temporelles de l'architecture et ses performances peuvent être estimées. Dans l'approche SPADE, afin d'exhiber le parallélisme des traitements et des communications, l'application est décrite sous la forme de réseaux de processus Kahn (KPN). Selon ce formalisme, des processus parallèles communiquent via des canaux de type FIFO de capacité infinie. L'opération de

lecture sur ces canaux est bloquante tandis que l'écriture est non bloquante. Chaque processus est exécuté séquentiellement. Le modèle KPN ne contient aucune information temporelle. Il considère uniquement l'ordre d'exécution et des communications entre processus. L'application peut être décrite selon ce formalisme dans un langage tel que C/C++. Chaque processus est alors décrit sur la base de primitives *Read*, *Write* et *Execute*. Cette description peut alors être exécutée afin de fournir les modèles de charge de communication et de calcul liés à l'application. Au sein du modèle de la plate-forme, la description associée à chaque ressource de calcul correspond à la définition du nombre de cycles nécessaires par instruction. La Figure 2.5 illustre le principe d'application de la méthodologie SPADE sur un exemple.

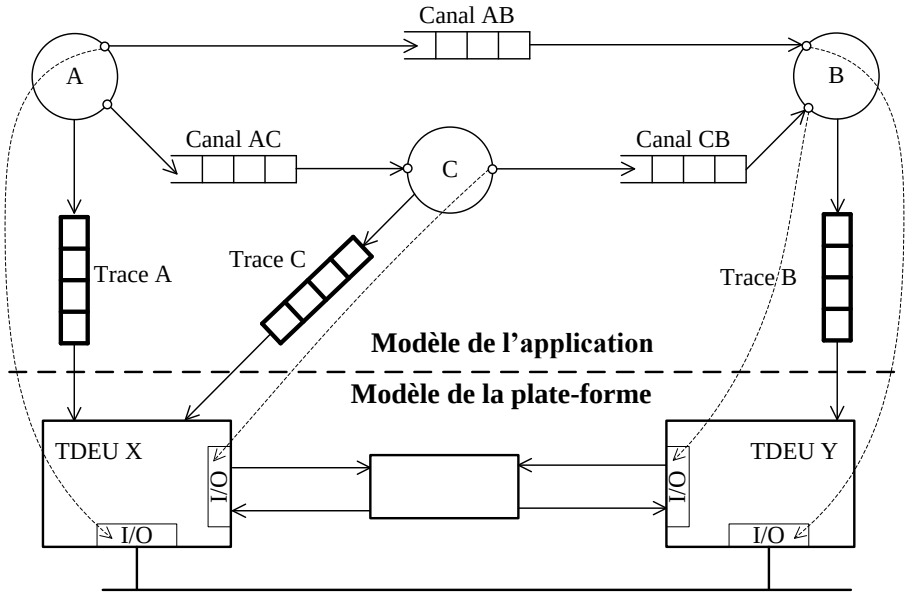


Figure 2.5 – Représentation d'un exemple d'architecture selon la méthodologie SPADE.

La partie haute de la Figure 2.5 montre le modèle de l'application sous forme de réseaux de processus Kahn. Trois processus A, B et C sont présenté par des cercles. Les interfaces de communication utilisées par ces processus sont présentées par des petits cercles collés à ceux des processus. Des canaux de type FIFO relient ces interfaces. La description associée aux interfaces correspond aux différents protocoles d'échange. La partie basse de la Figure 2.5 montre le modèle de la plate-forme. Les ressources de calcul sont notées TDEU sur la Figure 2.5 (*Trace Driven Execution Unit*). Les interfaces présentes au sein de la plate-forme définissent les échanges entre ressources de calcul selon différents modes possibles (bus, point-à-point). Les ressources de communication sont présentées, sur la Figure 2.5, par des interfaces d'entrées-sorties (I/O). Sur la Figure 2.5, une allocation des ressources de calculs et de communications de la plate-forme est effectuée pour différents processus et interfaces de communication de l'application. Ainsi, chaque trace est associée à différentes ressources de la plate-forme. Les traces produites lors de l'exécution de l'application guident la façon dont les ressources de la plate-forme sont utilisées. Les TDEU interprètent ces traces sur la base d'un modèle définissant les latences en nombre de cycles associés à chaque instruction à exécuter. Les observations alors obtenues portent sur l'utilisation des ressources de calcul et de communication.

Les travaux présentés dans [**32**], [**33**] et [**34**] détaillent l'approche intitulée SESAME (*Simulation of Embedded System Architectures for Multilevel Exploration*). Cette approche correspond à une extension de l'approche SPADE. Comme pour l'approche SPADE, le processus de modélisation repose sur le flot en Y. Le modèle de l'application repose sur l'utilisation de réseaux de processus Kahn. Par ailleurs, la simulation des architectures repose sur la technique « *trace-driven* ». Par rapport à l'approche SPADE, l'approche SESAME propose une exploration de l'espace de conception sur la base d'un algorithme de recherche SPEA2 (*Strength Pareto Evolutionary Algorithm 2*). Cet algorithme vise à identifier les allocations de l'application sur les ressources de la plate-forme minimisant le temps d'exécution maximal, de consommation totale et de coût total. Dans un second temps, la simulation des solutions retenues est effectuée selon le principe de simulation « *trace-driven* ». Par rapport à l'approche SPADE, cette simulation utilise un niveau intermédiaire qualifié de « *Mapping layer* ». Ce niveau correspondant à l'intersection entre le modèle d'application et le modèle de plate-forme. A ce niveau, la notion de processeur virtuel (*virtual processor*) est introduite pour désigner l'élément assurant l'ordonnancement des différents processus de l'application sur les ressources de calcul de la plate-forme. La description de l'architecture est traduite dans les langages Pearl ou SystemC afin de permettre la simulation. Les résultats de simulation obtenus portent sur l'observation de l'utilisation des différents éléments de la plate-forme. Le principe de la simulation repose donc sur l'analyse des traces résultant des charges induites par l'application sur les ressources de la plate-forme. L'une des originalités de l'approche porte également sur le fait de proposer une démarche de raffinement progressif du modèle d'architecture afin de couvrir différents niveaux d'abstraction.

L'approche présentée dans [**35**] porte sur la conception d'applications du domaine du traitement du signal pour des plates-formes hétérogènes sur FPGA. Cette approche est supportée par l'environnement SystemCoDesigner. Cette approche est très similaire à l'approche SESAME. L'approche présentée supporte une phase d'exploration automatisée de l'espace de conception ainsi qu'une phase de simulation des solutions retenues. L'exploration porte sur l'identification de solutions minimisant le nombre de ressources matérielles utilisées au sein de FPGA (LUT, Bloc RAM, bascules Flip-Flop). Cette phase suppose une description préalable de l'application selon un modèle particulier appelé SysteMoc. Ce modèle est défini afin de pouvoir décrire plusieurs modèles classiquement utilisés pour la description d'applications du domaine du traitement du signal, par exemple les modèles KPN et SDF. Les solutions retenues sont ensuite simulées selon une approche similaire à celle utilisée dans l'approche SESAME. Pour ce faire, les ressources de la plate-forme sont décrites sous la forme d'éléments, qualifiés de *virtual processing components* (VPC), exprimant les retards liés à l'exécution des éléments de l'application. Un élément appelé *Director* est utilisé afin d'effectuer le lien entre les traces induites par le modèle d'application et les VPC. Les observations alors obtenues pour la simulation sont utilisées pour évaluer les performances temporelles de l'architecture en termes de latence moyenne et de débit moyen. Les solutions retenues peuvent ensuite être générées sous forme de ressources matérielles et logicielles ciblées pour à une plate-forme FPGA Xilinx.

Dans [**36**] l'approche intitulée TAPES (*Trace-based Architecture Performance Evaluation with SystemC*) est présentée. Cette approche repose sur le principe qu'il n'est pas nécessaire d'intégrer la description complète des fonctionnalités du système pour effectuer l'évaluation des performances de l'architecture. Dès lors, ces fonctionnalités sont décrites sous la forme de traces. Ces traces désignent ici l'influence que représente l'exécution des fonctionnalités du système sur les ressources de calcul, de communication et de mémorisation. Chaque ressource est décrite sous la forme d'une succession de délais exprimant la séquence d'exécution des fonctions de l'application. Les échanges entre les ressources sont également décrits. La Figure 2.6 donne un exemple possible de traces telles que définies selon cette approche.

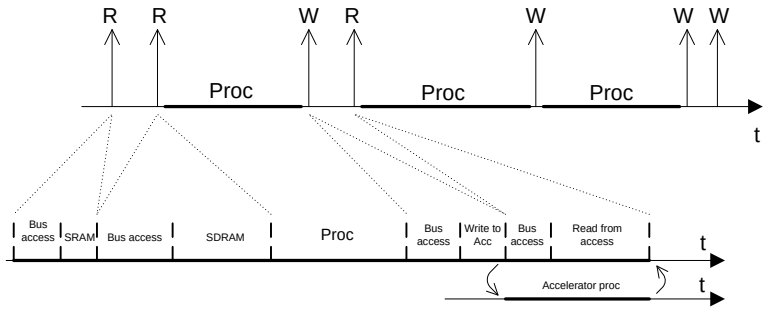


Figure 2.6 – Exemple de traces obtenues selon l'approche TAPES.

La Figure 2.6 représente la trace obtenue pour décrire l'exécution d'un traitement, noté Proc sur la Figure 2.6, sur un processeur donné d'une plate-forme. Sur l'exemple illustré deux opérations de lecture (notées R) sont effectuées avant traitement. Une fois le traitement effectué une opération d'écriture (notée W) est menée, par exemple vers une autre ressource de calcul. Chaque opération peut être affinée pour exprimer les accès aux ressources du type mémoire et bus. Ainsi, une observation plus détaillée de l'exécution est décrite sur la partie basse de la Figure 2.6. L'analyse de telles traces permet d'obtenir une estimation de l'utilisation des ressources de la plate-forme. Selon l'allocation considérée, chaque ressource est décrite par un ensemble de traces indiquant la séquence de traitements. La spécification des traces est faite à partir de fichiers selon une syntaxe particulière définissant les traces à utiliser. Ces fichiers sont associés à chaque ressource de la plate-forme pour être interprétés. La description de l'architecture est effectuée à l'aide du langage SystemC.

Dans [**37**], une approche pour la création de modèles en vue de l'évaluation des performances de systèmes embarqués est présentée, avec une orientation particulière vers les dispositifs mobiles multimédias. Cette approche repose sur la description de l'application supportée par la plate-forme sous la forme d'un modèle de charge de travail. La description de l'application est faite de manière hiérarchique afin d'exhiber le comportement de l'application et les différentes fonctions induisant une charge de calcul et de mémorisation. Cette description est faite sur la base du langage UML2, à partir des diagrammes de classes et d'automates. Les modèles de charge sont décrits sur la base d'automates auxquels sont associés des primitives abstraites du type *read*, *write* et *execute*. Une telle description n'inclut pas le détail des fonctionnalités supportées. La plate-forme est également décrite de façon hiérarchique afin de définir les services associés à chaque niveau

de représentation de l'application. Au niveau *component*, le plus détaillé, les éléments de la plate-forme sont décrits de façon à exprimer le nombre de cycles nécessaires par instruction. Les effets des mémoires cache sur les temps d'exécution sont décrits de manière statistique. Le niveau *subsystem* décrit la structuration des éléments au niveau *component*. Enfin, le niveau *platform architecture* inclut les services définis aux niveaux précédents ainsi que les ressources logicielles associées (les services de l'OS notamment). La description de l'application est ensuite allouée sur les éléments de la plate-forme. Le résultat de l'allocation est ensuite traduit en une description simulable dans le langage SystemC. Les résultats de simulation obtenus portent sur l'utilisation des ressources de calcul et de communication au cours du temps. Comme pour l'approche TAPES et contrairement aux l'approches SPADE et SESAME, les fonctionnalités de l'application ne sont pas décrites d'une façon détaillée afin de disposer de simulations plus rapides. En lien à cette approche, trois techniques sont proposées afin d'obtenir des modèles de charge selon différents niveaux de précision. Une première technique purement analytique consiste à déduire à partir de la description des algorithmes les instructions nécessaires de traitement et le coût mémoire associé. Une seconde technique consiste à qualifier chaque fonction sur la base de mesures effectuées préalablement sur des plates-formes de référence. Enfin, une technique basée sur l'analyse du code source [**38**] est présentée. Un outil spécifique est alors utilisé afin d'extraire les informations de charge à partir de l'analyse du code C.

Les travaux décrits dans [**39**] présentent une démarche pour la création de modèles en vue de l'évaluation des performances de systèmes embarqués, avec une orientation particulière vers les applications flots de données. Comme pour les autre approches précédemment présentées, l'évaluation des performances est effectuée par simulation afin d'estimer les ressources de calcul, de mémorisation et de communication requises ainsi que les performances temporelles atteintes. Un méta-modèle particulier est défini afin de guider la création des modèles de l'application et de la plate-forme. Les règles établies pour la description de l'application conduisent à définir l'application sous la forme d'un réseau de processus communiquant via des FIFO infinies. Chaque processus est caractérisé par un comportement propre au sein duquel sont exprimées la complexité de calcul et la mémoire requise. Les ressources de calcul et de communication de la plate-forme sont caractérisées par des critères de mérite permettant de définir les temps d'exécution de chaque fonction ou les temps de chaque transfert entre ressources. Sur la base des règles définies au sein de ce méta-modèle, le modèle de l'application est capturé, pour l'application, à l'aide du diagramme d'activité supporté par le langage UML2, et le modèle de la plate-forme est capturé par un diagramme de classe. Ce diagramme de classe définit les propriétés associées à chaque constituant de la plate-forme. Le profil spécifique MARTE est également utilisé [**40**]. Ce profil comporte notamment les éléments nécessaires à la définition de propriétés non-fonctionnelles et de stratégies d'allocation au sein des modèles. Le modèle d'architecture obtenu est ensuite traduit en une description SystemC afin d'être simulé et analysé. Le niveau d'abstraction considéré conduit à la création de modèles ne comportant pas la description précise des fonctionnalités mais uniquement les temps liés à l'exécution des fonctions de l'application sur les ressources de la plate-forme. L'exploration de l'espace de conception est menée de façon à analyser différentes allocations possibles. Différentes

allocations optimisées sont identifiées en comparant de manière automatisée les performances obtenues.

Les travaux présentés dans [**41**] décrivent un processus de conception de systèmes sur puce multiprocesseur, appelé Koski, basé sur l'utilisation du langage UML2. Cette approche repose sur un processus de modélisation en Y pour lequel différents profils spécifiques ont été définis afin de tenir compte des particularités des architectures envisagées. L'application est décrite sur la base de réseaux KPN. La complexité associée à chaque processus est évaluée par des mesures sur une plate-forme de référence à base de processeurs NIOSII. Les ressources de calcul de la plate-forme sont caractérisées par leur puissance de calcul, leur surface, leur consommation et la mémoire disponible. Les ressources de communication sont elles définies par la bande passante disponible, la surface associée et leur consommation. L'exploration de l'espace de conception est menée en deux étapes. La première, dite statique, consiste à identifier des allocations et des ordonnancements optimisés de l'application sur les ressources de la plate-forme. La seconde étape, dite dynamique, consiste à simuler de manière itérative l'architecture en faisant varier successivement différents paramètres de conception. Cette évaluation permet d'obtenir différents compromis en termes de coûts, de temps d'exécution et de surface, et ce pour différents ensembles de stimuli appliqués à l'architecture.

L'approche présentée dans [**42**] porte sur une démarche pour la prise en compte des propriétés non-fonctionnelles des architectures que sont la température et la consommation. Cette prise en compte repose sur une mesure préalable de ces propriétés compte tenu de la plate-forme considérée. Cette caractérisation sert à la construction d'un modèle de la plate-forme sous la forme d'un automate exprimant les différents états de consommation de la plate-forme (*power-state machine*). Le comportement temporel associé à l'application est représenté sous la forme d'un graphe de dépendance nommé *communication dependency graph*. Cette représentation exhibe les états significatifs au sein des activités composant l'application ainsi que les dépendances de données. Cette représentation comporte également les annotations temporelles exprimant les meilleurs et pires cas d'exécution pour chaque état. Le modèle d'architecture résulte de l'association des deux types de description, c'est-à-dire les activités du système et les modèles de consommation et de température. Le modèle résultant est exprimé dans le langage SystemC et permet une observation de l'évolution de la consommation et de la température lors de l'exécution de l'application. Il est ainsi possible d'analyser différentes stratégies de gestion de la consommation et l'effet sur les performances temporelles.

De façon générale, les travaux présentés dans [**43**] portent sur l'environnement SCE (*System-on-Chip Environment*). Cet environnement repose sur le langage SpecC et la méthodologie associée [**44**]. L'environnement SCE supporte différentes étapes visant à l'exploration et au raffinement de systèmes pour une réalisation sur des plates-formes hétérogènes. Chacune des étapes comporte un ensemble d'outils afin de spécifier un modèle de représentation, analyser les performances associées selon les paramètres de dimensionnement et raffiner la description vers le niveau de représentation suivant. Les modèles capturés sont exprimés selon le formalisme SpecChart et traduits dans le langage SpecC. La phase d'exploration d'architecture est organisée en quatre phases :

l'identification et l'allocation des ressources de calcul, l'ordonnancement des tâches sur les ressources de calcul, l'identification et l'allocation des ressources de communication, et la conception des ressources de communication.

Parmi les autres travaux de références, l'approche METROPOLIS décrite dans [**45**] et [**46**] a été pensée afin de supporter les différents principes associés à une approche de conception orientée plate-forme. Cette approche repose sur un modèle de référence, ou méta-modèle, utilisé pour la représentation du modèle dit fonctionnel, c'est-à-dire de l'application, et du modèle dit architectural (la plate-forme). Le modèle fonctionnel repose sur un ensemble de services appelés afin d'assurer les calculs et les communications nécessaires. Ces services sont mis en œuvre par les ressources identifiées au sein du modèle architectural. La phase d'allocation (*mapping*) consiste à définir les dépendances entre les deux modèles décrits. Les contraintes imposées alors fixent le degré de concurrence applicable sur le modèle fonctionnel et définissent la séquence d'opérations imposées sur le modèle de plate-forme. Ces contraintes peuvent être utilisées pour vérifier le respect de certaines exigences telles que les contraintes de temps ou de coût. Ces différentes descriptions sont ensuite traduites en un code SystemC exécutable afin d'être simulé. La portée de l'approche réside dans le méta-modèle utilisé qui permet de décrire de nombreux systèmes et pouvant être traduit selon différents modèles de représentations. Ces travaux sont actuellement poursuivis dans le cadre de l'environnement MetroII [**28**].

Cette présentation de différentes approches de dimensionnement des architectures de systèmes embarqués permet de retirer différentes observations. Tout d'abord, en ce qui concerne la démarche de création des modèles d'architectures, la plupart des travaux reposent sur une démarche en Y. Le modèle d'architecture résulte alors de la combinaison des modèles d'application et de plate-forme. Parmi les approches présentées, seule l'approche TAPES ne repose pas sur le processus de création en Y. Dans toutes ces approches, les modèles d'architectures créés expriment l'utilisation des ressources de calcul et de communication des plates-formes lors de l'exécution des applications. Les approches présentées diffèrent selon le niveau de description de l'application. Certaines approches (SPADE, SESAME, et [**35**]) considèrent une description complète de l'application afin d'en extraire le modèle de charge de travail associé. D'autre approches (TAPES, [**37**], [**39**], [**42**] et METROPOLIS) ne considèrent pas une description complète de l'application et se basent sur une connaissance a priori des charges souhaitées. L'intérêt réside alors dans le fait d'accélérer les temps de simulation des modèles. Au sein du chapitre 3 nous utiliserons une notation particulière afin de décrire de tels modèles d'architectures. Pour toutes les approches présentées jusqu'ici, l'obtention de modèles exécutables des architectures permet d'analyser l'influence des paramètres de conception sur les ressources utilisées. Ces modèles exécutables sont décrits dans les langages SystemC ou SpecC.

Parmi les approches présentées, différents niveaux de description des architectures sont considérés. La section suivante détaille les différents niveaux d'abstraction pouvant être considérés afin de décrire les ressources de calcul et de communication d'une architecture. Elle évoque également certaines techniques adoptées en vue de l'amélioration des temps de simulation des modèles d'architectures.

2.3 Modélisation transactionnelle des architectures

La notion de modélisation transactionnelle a récemment été introduite afin de désigner une représentation de l'architecture à un niveau de précision au dessus du niveau classique RTL (*Register Transfer Level*). Dès lors, différents compromis entre la précision des résultats et la rapidité de simulation peuvent être considérés.

2.3.1 Définition des niveaux de représentation des architectures

Comme expliqué précédemment, la description d'un système embarqué repose sur trois vues : application, plate-forme et architecture. Chaque vue peut être décrite selon différents niveaux de précision. En passant d'un niveau donné à un niveau plus abstrait, il est ainsi possible de masquer dans la description d'un système des détails non significatifs pour sa modélisation. Cette abstraction permet donc de maitriser la complexité d'un système afin de faciliter sa modélisation et d'accélérer l'obtention de résultats. De nombreux travaux ont été menés afin de classifier les différents niveaux d'abstraction à considérer pour la description des systèmes embarqués. La Figure 2.7, extraite de [**47**], positionne différents niveaux de précision pouvant être identifiés pour la description des différents points de vues d'un système embarqué. Cette figure complète la Figure 2.1 en introduisant les différents niveaux de représentation d'une architecture.

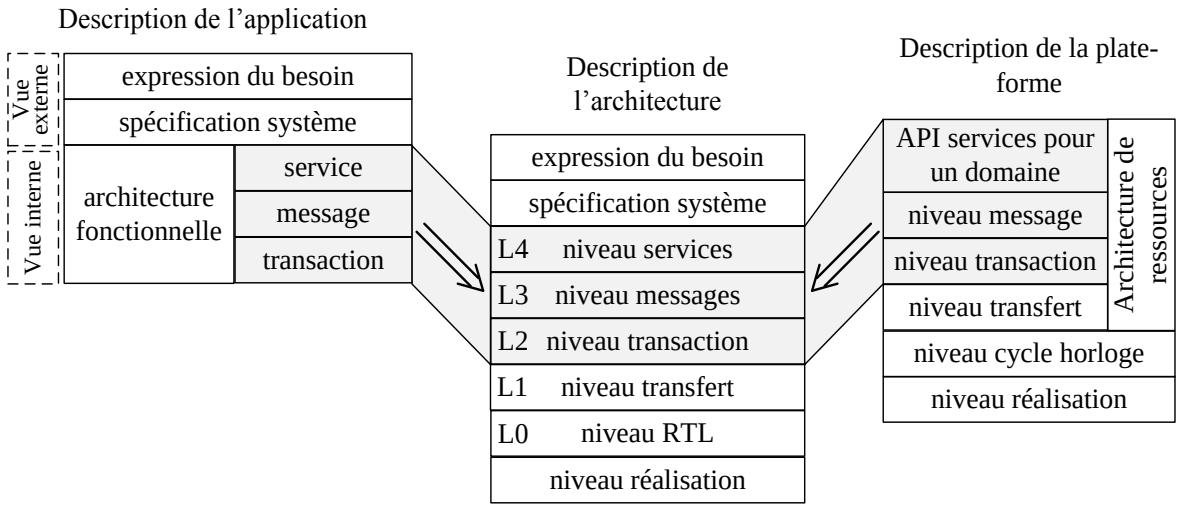


Figure 2.7– Les niveaux d'abstraction des vues application, plate-forme et du modèle d'architecture.

La description de l'application peut être perçue aussi bien d'un point de vue extérieur qu'intérieur, comme le montre la Figure 2.7. D'un point de vue extérieur, l'application peut être exprimée en tant que besoin sous forme d'un cahier des charges ou peut être spécifiée. Cette spécification décrit les contraintes fonctionnelles et non fonctionnelles, liées à des contraintes de conception ou de réalisation. Du point de vue interne, l'application est vue comme une solution fonctionnelle qui se focalise sur les objectifs fonctionnels du système en excluant toute information liée à la réalisation. La solution fonctionnelle exprime différents modules fonctionnels qui communiquent entre eux. Il existe trois niveaux d'abstraction qui permettent de représenter cette communication. Ces niveaux, du plus abstrait au plus détaillé, sont le niveau service, le niveau message et le niveau transaction.

Au niveau service, la communication est effectuée sous forme de services rendus en réponse à une requête, par exemple « décompresser (image) ». Au niveau message, les modules communiquent via des canaux qui peuvent disposer d'un comportement particulier. Ces canaux permettent l'envoi de messages de synchronisation ou de messages selon des protocoles donnés, par exemple « envoi (message, destination) ». Au niveau transaction, la communication est effectuée via des liens abstraits point à point qui supportent des échanges de données élémentaires et de synchronisations, par exemple « lire (données, adresse) ».

La description de la plate-forme exprime les ressources matérielles sans faire référence à l'application. Ces ressources peuvent être spécifiées avec trois niveaux d'abstraction, comme le montre la Figure 2.7. Ces niveaux, dans l'ordre du niveau du plus abstrait au plus détaillé, sont le niveau architecture de ressources, le niveau cycle d'horloge et le niveau réalisation. L'architecture de ressources est formée de composants actifs comme les processeurs, les microcontrôleurs, les DSP, les composants mémoires, les FPGA etc. Ces ressources communiquent via des réseaux d'interconnexion. La communication peut être décrite à quatre niveaux de détail. Il s'agit des trois niveaux d'abstraction des communications : le niveau service, le niveau message et le niveau transaction, auxquels on ajoute un niveau supplémentaire plus détaillé. Au niveau transfert, la communication repose sur des transferts de données élémentaires à une adresse explicite d'un composant, avec une échelle de temps correspondant au cycle d'horloge, par exemple « écrire (valeur, adresse composant) ». La description des ressources matérielles au niveau cycle d'horloge est une description RTL. Cette description précise les interconnexions entre les portes logiques et considère le cycle d'horloge comme unité de temps. La communication est réalisée directement sur des ports via des fils et des bus physiques.

Compte tenu de ces observations, la description d'une architecture peut être faite à différents niveaux d'abstraction : service, message, transaction, transfert ou RTL. Afin de favoriser la conception des architectures et de faciliter les interactions entre concepteurs à différents niveaux d'abstraction, la notion de la modélisation transactionnelle a émergé à partir de la fin des années 1990. La modélisation transactionnelle désigne une approche de modélisation d'architectures à un niveau de précision compris entre une description non temporisée et une description précise au niveau du cycle d'horloge. Cette approche, appelée aussi TLM (*Transaction Level Modeling*), considère séparément la description des ressources de calcul et des ressources de communication d'une architecture. La notion de transaction qualifie alors les échanges effectués au sein d'un modèle d'architecture. Ainsi, une transaction désigne un échange de données ou une synchronisation entre deux modules constituant l'architecture. Il existe différents niveaux de précision pour la description des ressources de calcul et de communication au sein d'une architecture. La Figure 2.8 reprend la classification faite dans [**48**].

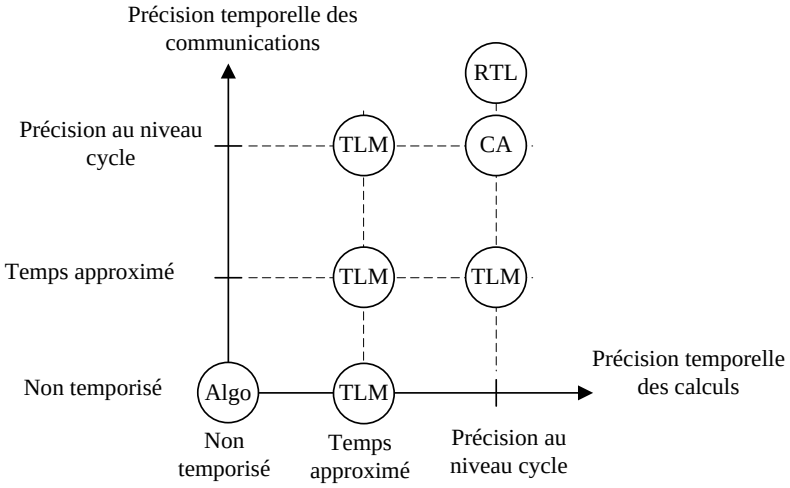


Figure 2.8 –Différents niveaux de précision temporelle des calculs et des communications [**48**].

La Figure 2.8 positionne différents niveaux de représentation possibles d'une architecture. Ces niveaux sont échelonnés selon la précision temporelle de la description des ressources de calculs et de communications au sein d'une architecture. Une description non temporisée ne prend pas en compte les caractéristiques temporelles. Par exemple, une représentation algorithmique seule permet de décrire un comportement mais n'inclus pas les caractéristiques temporelles des ressources de calcul et de communication. Une description à temps approximé décrit, d'une façon approximative, les caractéristiques temporelles des ressources de communication et de calcul. Il est ainsi possible de définir certaines durées caractéristiques de l'évolution de l'architecture considérée. Une description au niveau cycle CA (*Cycle Accurate*) considère le cycle d'horloge comme unité d'avancement de l'évolution du comportement. Ainsi, les ressources de calcul et de communication sont décrites de façons précises au niveau cycle d'horloge. Le niveau RTL correspond à un niveau de détail supplémentaire à celui du CA pour lequel sont précisés les signaux logiques utilisés. Afin d'illustrer ces notions, un exemple de représentation d'architecture est présenté dans la Figure 2.9.

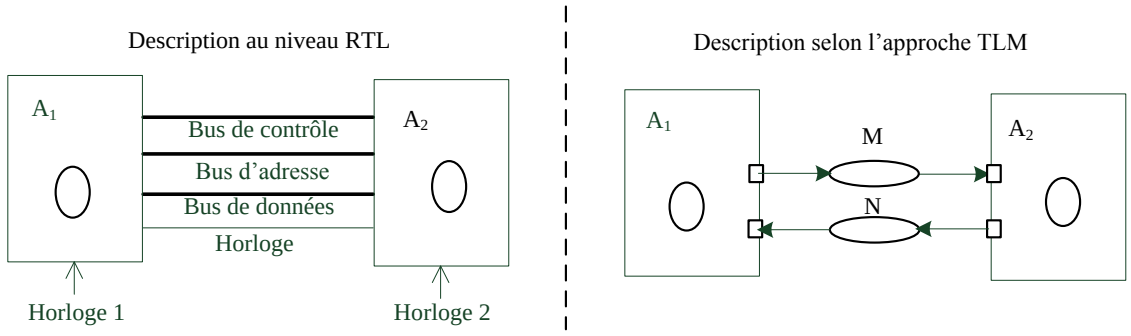


Figure 2.9 – Description d'une d'architecture au niveau RTL et selon l'approche TLM.

Sur l'exemple présenté sur la Figure 2.9, l'architecture représentée au niveau RTL se constitue des éléments A_1 et A_2. Pour une représentation au niveau RTL, ces éléments sont décrits sur la base de processus synchrones à des horloges et communiquent via un ensemble de signaux logiques. A ce niveau de représentation, les ressources logiques et les

signaux évoluent à chaque cycle d'horloge. La description de cette architecture selon l'approche TLM conduit à une représentation sous forme de modules (A_1 et A_2) et de relations entre modules (M et N). Les modules sont décrits sur la base de processus séquentiels pouvant intégrer des indications temporelles. Les relations sont accédées par les modules via un ensemble de méthodes. Ainsi, une transaction peut représenter l'évolution, sur un intervalle de temps déterminé, de différents signaux présentés au niveau RTL.

La modélisation transactionnelle des architectures conduit à pouvoir considérer différents niveaux de représentation. De ce fait, la simulation des modèles créés selon cette approche peut être rendue plus rapide que pour le niveau RTL. En contrepartie, les résultats obtenus par la simulation peuvent s'avérer moins précis. Dès lors, dans la phase de conception, le concepteur est amené à choisir un certain compromis entre la vitesse de simulation et la précision des résultats selon le contexte de son étude. Parmi les travaux menés afin de quantifier ce compromis, dans [**49**], les auteurs présentent une classification des niveaux de représentation des communications au sein d'une architecture. Les communications décrites expriment les échanges effectués entre ressources de calculs. Cette classification porte sur le niveau de représentation du temps, sur le niveau de granularité des données échangées ainsi que sur le niveau de représentation des arbitrages nécessaires lors des transferts entre modules. Les niveaux identifiés sont illustrés sur la Figure 2.10.

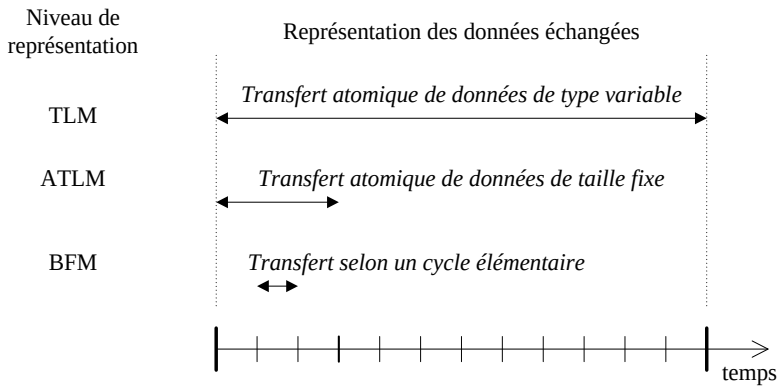


Figure 2.10 – Classification proposée dans [**49**] des niveaux de représentation des communications.

La classification proposée identifie les niveaux suivants :

- Niveau TLM (*Transaction Level Model*) : ce niveau désigne des transferts atomiques de données de tailles variables, en considérant une durée globale de transfert pour l'ensemble des données. A ce niveau, l'arbitrage entre plusieurs transferts n'est pas décrit, seul un accès exclusif du type sémaphore est considéré. Ce niveau est intermédiaire entre les niveaux Message et Transaction présenté sur la Figure 2.7.
- Niveau ATLM (*Arbitrated TLM*) : ce niveau désigne des transferts à un niveau de granularité de données faisant apparaître l'arbitrage nécessaire pour l'accès au ressource de communication, sans pour autant être précis au niveau du cycle élémentaire de transmission. Ce niveau est intermédiaire entre les niveaux Transaction et Transfert présenté sur la Figure 2.7.

- Niveau BFM (*Bus Functional Model*) : ce niveau désigne des transferts précis au cycle près, prenant en compte les différents mécanismes d'arbitrage nécessaires. Ce niveau est intermédiaire entre les niveaux Transfert et RTL présenté sur la Figure 2.7.

Ces différents niveaux sont considérés pour évaluer le compromis entre précision des estimations et rapidité de simulation. Pour ce faire trois protocoles de bus sont considérés : le protocole de bus de terrain CAN et les protocoles de bus pour systèmes à microprocesseur AMBA/AHB et ColdFire. Il est ainsi illustré le gain potentiel en termes de rapidité de simulation en augmentant le niveau de représentation des communications. Pour exemple, dans le cas du protocole CAN, les expérimentations menées montrent des facteurs d'accélération de l'ordre de 22000 entre un modèle défini au niveau TLM et un modèle BFM, et ce pour une précision des estimations de l'ordre de 42%.

Dans ce cadre, il est possible de définir des techniques permettant d'améliorer le compromis existant entre rapidité de simulation et précision des estimations. Avant d'évoquer ces techniques, il est important de comprendre le principe de simulation de modèles d'architectures créés selon l'approche de modélisation transactionnelle.

2.3.2 Simulation de modèles transactionnels

La modélisation transactionnelle est supportée par des langages tels que SystemC, SpecC et SystemVerilog. Cette approche de modélisation est actuellement utilisée dans les différentes phases de mise au point de SoC (développement algorithmes, exploration d'architectures, développements logiciels et matériels, vérification) au niveau industriel [**50**]. Par la suite, nous nous baserons plus particulièrement sur le langage SystemC [**51**].

La description d'une architecture dans le langage SystemC consiste à décrire un ensemble de modules accédant aux relations par un ensemble de méthodes. Les processus composant l'architecture peuvent contenir des indications temporelles. Le rôle du noyau de simulation SystemC est de permettre l'exécution des différents processus compte tenu des synchronisations entre processus et des indications temporelles. Ce noyau opère donc selon le principe d'un simulateur à événements discrets. Le principe de fonctionnement est illustré sur la Figure 2.11 extraite de [**51**].

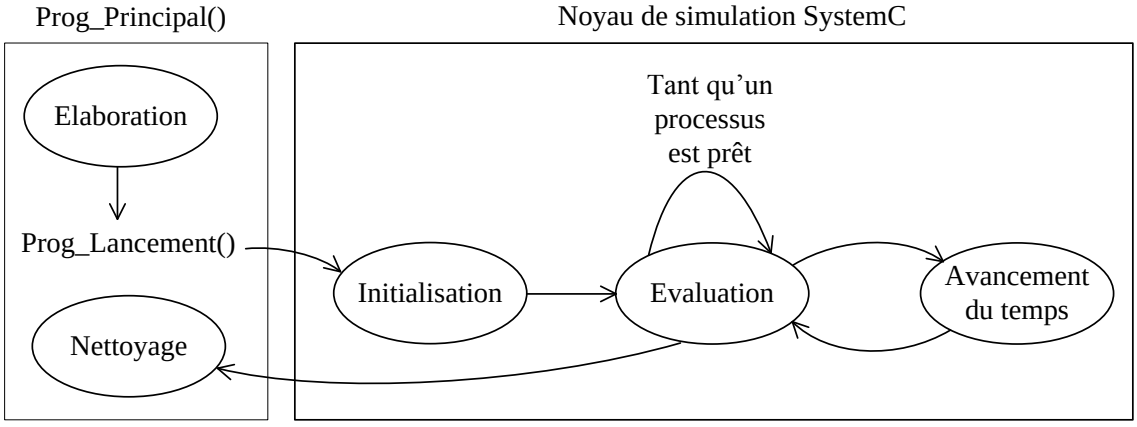


Figure 2.11 – Principe de fonctionnement du noyau de simulation SystemC.

Avant le lancement de la simulation, une phase d'élaboration est effectuée. Cette phase établit la hiérarchie entre processus et initialise les structures de données. Dans cette phase, les instances des horloges, des modules et des canaux d'interconnexion du modèle sont créées. Ensuite, la simulation débute par une phase d'initialisation. Dans cette phase tous les processus définis dans la phase de l'élaboration sont classés comme processus prêts. Les paramètres et valeurs des données sont également initialisés. Lors de la simulation, les différents processus prêts sont successivement exécutés dans la phase d'évaluation. A la fin de l'exécution d'un processus, ce dernier est classé en attente. Il peut s'agir d'une attente d'événement ou de condition temporelle. Quand tous les processus sont en attente, le simulateur avance le temps à l'échéance la plus proche et continue l'évaluation. A la fin de la simulation, une phase de nettoyage est effectuée pour détruire toutes les structures de données et libérer la mémoire de la machine exécutant la simulation. Nous expliquons dans la suite l'exécution d'un modèle et l'influence du simulateur sur son exécution.

La Figure 2.13 représente un modèle d'architecture formée par deux modules A_1 et A_2.

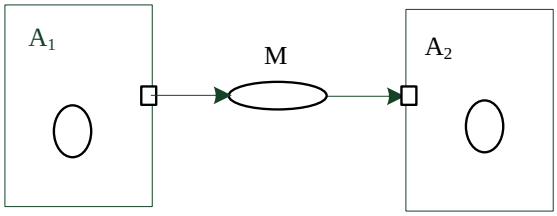


Figure 2.12 – Modèle d'architecture à deux activités.

La Figure 2.13 illustre un exemple d'exécution des processus en fonction du temps.

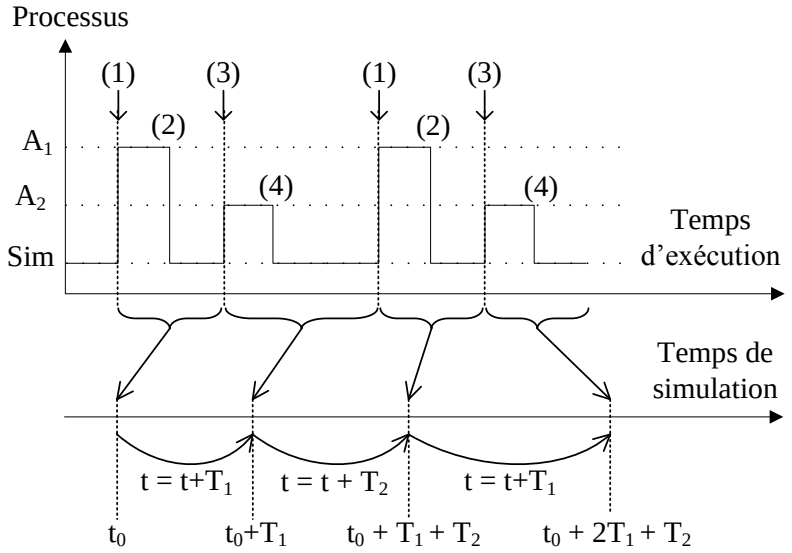


Figure 2.13 – Exécution des processus au cours d'une simulation.

On distingue deux échelles de temps. Le temps d'exécution est un temps « réel » nécessaire pour l'exécution des processus sur une machine. Le temps de simulation correspond au temps perçu par le modèle au cours de la simulation. A_1, A_2 et Sim correspondent respectivement aux processus des modules A_1, A_2 et du noyau du simulateur. On définit un changement de contexte comme l'arrêt de l'exécution d'un processus et le

déclenchement de l'exécution d'un autre. Sur la Figure 2.13, le premier changement de contexte est effectué lors de (1). En effet, le processus Sim est arrêté et A_1 est exécuté. La Figure 2.14 montre les états des processus avant et après la prise en compte de (1).

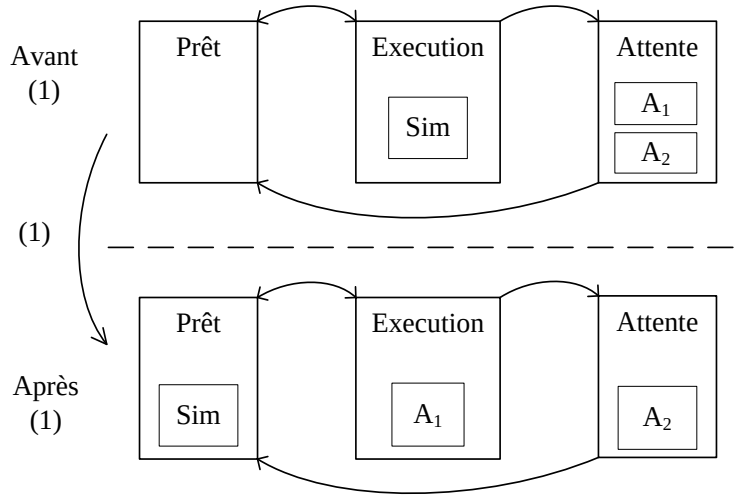


Figure 2.14 – Etats des processus lors de la prise en compte d'un événement.

Dans la partie haute de la Figure 2.14, avant (1), le processus Sim est en exécution. Il avance le temps de simulation à t_0 ce qui provoquera la production de (1). Suite à cet événement, Sim et A_1 passent momentanément à l'état prêt. Etant donné que Sim est le processus le moins prioritaire, A_1 passe à l'exécution et Sim reste à l'état prêt.

Sur la Figure 2.13, le deuxième changement de contexte, noté (2), est effectué lors de l'arrêt de A_1 et l'activation de Sim. L'exécution de A_1 s'effectue en temps nul vis-à-vis du temps de simulation. Elle s'effectue à l'instant noté t_0 sur la Figure 2.13. Après (2), le simulateur avance le temps de simulation à l'instant t_0+T_1. Ainsi, le troisième changement de contexte (3) est effectué en arrêtant Sim et en activant A_2. Finalement, à la fin de l'exécution de A_2, le quatrième changement de contexte (4) est effectué en arrêtant A_2 et en activant Sim. L'exécution successive de A_2 et Sim s'effectue en temps nul vis-à-vis au temps de simulation. Elle s'effectue à l'instant noté t_0+T_1 sur la Figure 2.13. Après (4) l'exécution de Sim permet d'avancer le temps de simulation à l'instant $t_0+T_1+T_2$.

Le principe retenu pour simuler l'exécution de processus concurrents est illustré sur la Figure 2.15 [**51**]. Le processus du simulateur n'est pas représenté.

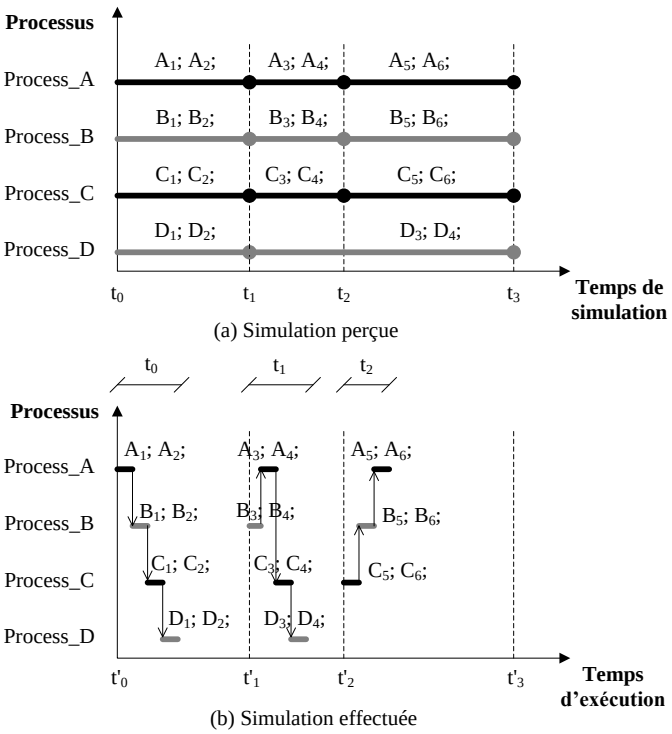


Figure 2.15 – Principe de simulation de processus concurrents à partir du noyau de simulation SystemC [**51**].

La partie (a) de la Figure 2.15 illustre la simulation perçue lorsque quatre processus concurrents s'exécutent. Chaque processus exécute successivement des instructions (par exemple A_1, A_2, pour Process_A), cette exécution étant observée par rapport au temps de simulation. Les instants t_1, t_2 et t_3 représentent les instants auxquels certaines instructions sont censées se terminer. Pour exemple le processus Process_D exécute les instructions D_1 et D_2 sur un intervalle compris entre t_0 et t_1 puis D_3 et D_4 entre t_1 et t_3. La partie (b) de la Figure 2.15 illustre le fonctionnement assuré par le noyau de simulation pour effectuer une telle simulation. Les instructions associées à chaque processus sont exécutées dans le même temps. Par exemple, les instructions C_5, C_6, B_5, B_6, A_5, A_6 sont exécutées successivement à l'instant t_2. L'ordre d'exécution des instructions est a priori défini aléatoirement par le noyau de simulation. Une fois les instructions exécutées, le temps de simulation est avancé jusqu'à l'instant suivant. Cette technique est utilisée pour donner l'illusion d'une exécution concurrente entre processus.

La simulation de modèles transactionnels repose donc sur l'emploi de simulateurs à événements discrets. De tels simulateurs gèrent les commutations entre processus formant le modèle de l'architecture considéré. Dès lors, la vitesse de simulation est limitée par les commutations de contexte entre processus. Ces contextes correspondent aux informations locales relatives à chaque processus. Les commutations induites par le simulateur impliquent la sauvegarde de ces informations. L'exécution répétitive de ce travail ralentit la simulation. Dans ce contexte, différentes études ont été menées afin d'améliorer les temps de simulation.

2.3.3 Techniques d'amélioration du compromis rapidité et précision de modèles transactionnels

De façon générale, différentes techniques permettent d'améliorer les temps de simulation des modèles transactionnels : la distribution de l'exécution des modèles sur des machines parallèles [**52**] ou la réorganisation de l'ordonnancement de l'exécution des processus afin de limiter le nombre de changement de contexte [**53**]. Pour autant ces différentes approches ne conduisent pas à considérer différents compromis possibles de précision. Ici le problème posé porte sur la définition de techniques conduisant à la création de modèles d'architectures offrant un bon compromis entre précision des estimations et rapidité de simulation. Les techniques abordées ici adressent l'application de ces techniques à la description des ressources de communication au sein d'une architecture. Dans ce cadre, les techniques proposées se basent sur une description de l'architecture sur la base de modules définissant les ressources de calcul et de communication nécessaires. Ainsi, le problème posé est illustré sur le cas de la Figure 2.16.

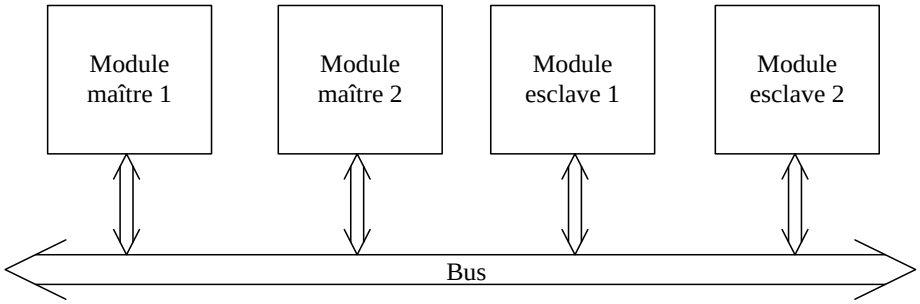
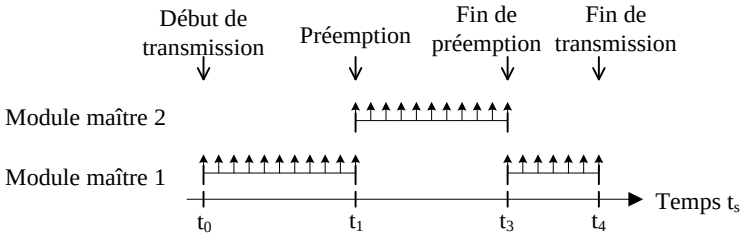


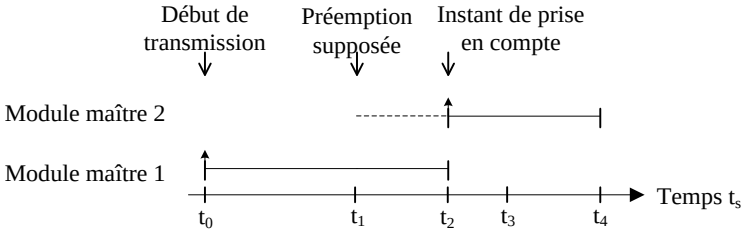
Figure 2.16 – Organisation d'une architecture composée de ressources de calcul et de communication.

L'architecture décrite correspond au cas d'une architecture composée de quatre éléments décrivant des ressources de calcul et des communications selon un bus partagé. Les deux modules notés maître 1 et maître 2 communiquent en direction respectivement des modules notés esclave 1 et esclave 2. La priorité du maître 2 est supposée supérieure à celle du maître1, la description faite de l'élément noté Bus devant permettre d'arbitrer les échanges effectués. Comme évoqué précédemment, il est possible de décrire avec différents niveaux de précision un tel élément. Les travaux évoqués ci-après visent à proposer un compromis efficace entre précision et rapidité de simulation.

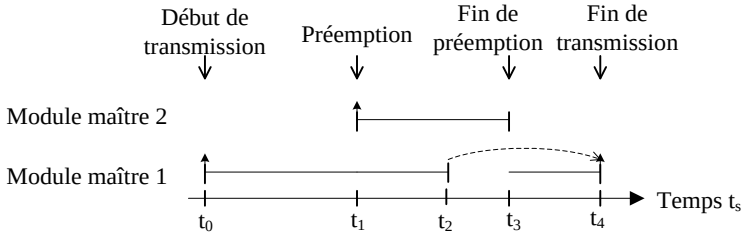
Parmi les travaux existants portant sur l'amélioration des temps de simulation de modèles transactionnels on peut citer les travaux de l'équipe de Rainer Dömer au sein du CECS à l'université de Californie. Afin d'offrir un compromis pertinent entre rapidité de simulation et précision des estimations une technique est présentée dans [**54**]. La classification retenue pour les différents niveaux de représentation est celle indiquée dans [**49**], et a été évoqué précédemment. Ainsi, cette technique, dite ROM (*Result Oriented Modeling*), est proposée afin d'améliorer la précision des résultats de simulation de modèles de niveau TLM. Le principe de cette technique s'explique pour la situation représentée précédemment et est illustré sur la Figure 2.17 extraite de [**54**].



(a) Modèle de niveau BFM



(b) Modèle de niveau TLM



(c) Modèle de niveau TLM+ROM

Figure 2.17 – Technique de simulation ROM en vue de l'amélioration des temps de simulation de modèles transactionnels.

Le cas d'étude considéré est celui de la communication entre deux modules pouvant à tout moment accéder au bus partagé. Dans le cas (a), la relation est décrite au niveau BFM, c'est-à-dire au niveau d'un cycle élémentaire et de manière à définir à tout instant l'arbitrage de l'accès au support de communication. Ce niveau correspond au cas le plus précis mais également le plus lent en terme de simulation. Le cas (b) correspond au niveau TLM pour lequel les échanges se font de manière atomique, un transfert occupant la ressource de communication pendant toute la durée du transfert des données englobées dans la transaction. La durée de transfert est déterminée compte tenu des données échangées et des caractéristiques associées à la relation. Cette durée s'exprime par l'emploi d'une primitive *wait* au sein de la description. Ce cas correspond à un modèle rapide en terme de simulation, car économisant de nombreux événements, mais potentiellement imprécis car ne permettant pas de prendre en compte la préemption éventuelle du support de communication par un autre nœud. Enfin, le cas (c) décrit la modélisation adoptée en utilisant la technique ROM. Dans ce cas, la description faite du module Bus permet de détecter les préemptions éventuelles. Dans ce cas, l'instant de fin de transfert, t_4 sur la Figure 2.17, est calculé dynamiquement en cours de simulation. La technique ROM repose donc sur le principe d'un calcul en cours de simulation des instants de transfert.

Dans le cas de plusieurs préemptions successives, ces corrections sont établies successivement, impliquant l'exécution de plusieurs primitives *wait* successives. Ce principe est illustré sur la Figure 2.18.

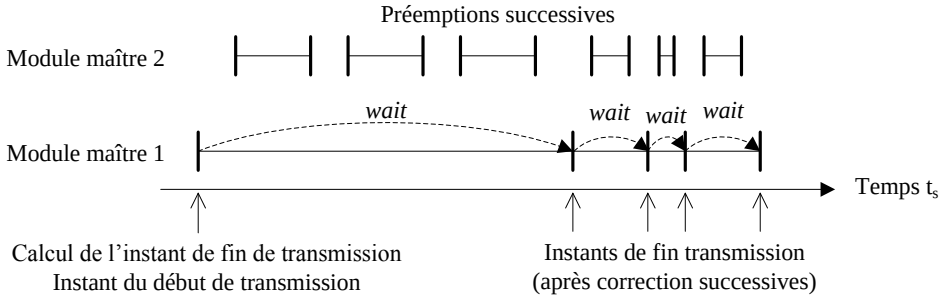


Figure 2.18 – Principe de calcul dynamique de l'instant de fin de transfert selon la technique ROM.

Dans le cas où aucune préemption n'intervient le même niveau de précision que pour le niveau TLM est atteint en utilisant la technique ROM. Cette technique est mise en œuvre par un ensemble de méthodes permettant de corriger en cours de simulation les instants d'évolution du modèle de bus. Une telle approche appliquée sur l'exemple du protocole de bus AHB montre une précision des résultats équivalente à un modèle de niveau cycle avec un gain en terme de temps de simulation d'un rapport 1000. Dans le cas du protocole de bus CAN le facteur d'accélération observé est de l'ordre de 10 000.

L'approche présentée dans [**55**] repose sur le principe d'adaptation dynamique, en cours de simulation, du niveau de précision des modèles de bus. Ce principe conduit à utiliser différents niveaux de précision lors de la simulation. Les niveaux de représentation associés sont les niveaux transactionnels et CA. Ainsi, lors des phases d'arbitrage du bus, le niveau de représentation est précis au niveau cycle tandis que pour les phases de transfert ne nécessitant pas d'arbitrage les transferts sont effectués en considérant un transfert atomique de données prenant une durée de donnée. Ce principe est illustré sur la Figure 2.19 pour le cas de la situation présentée précédemment sur la Figure 2.16.

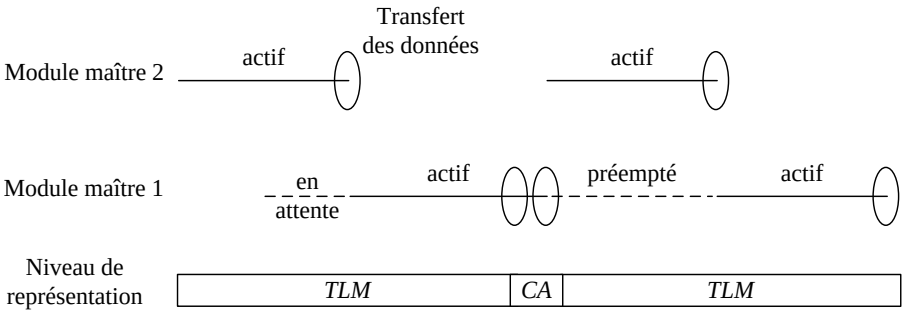


Figure 2.19 – Principe de simulation adaptant le niveau de représentation du bus de communication.

Lorsqu'un intervalle d'arbitrage approche et que le module utilisant le bus peut être préempté par un module de priorité plus élevée, le modèle de bus est passé au niveau CA, permettant une détection précise d'une demande d'accès éventuelle. Dans le cas où de nombreuses préemptions doivent être détectées le modèle se rapproche d'une description précise au niveau CA. La comparaison des temps de simulation relatifs pour différents taux

de préemption est présentée dans [**55**]. L'application de cette technique à la modélisation du bus AMBA AHB a été présentée dans [**56**].

Le principe de la co-simulation à différent niveaux d'abstraction a également été considéré dans [**57**] afin de simuler le protocole de communication FlexRay. Les travaux menés portent sur la définition d'une technique de co-simulation afin d'améliorer le passage entre des modèles abstraits, simulés sur la base de simulateurs à événements discrets, et des modèles plus précis. L'application de la technique proposée porte sur la co-simulation entre le langage SystemC et le langage VHDL-AMS. Le protocole FlexRay est considéré comme étude de cas mais peu de détail sont donnés sur le contenu des modèles créés. La comparaison établie un facteur d'accélération de l'ordre de 25 000 entre un modèle mixte et un modèle SystemC seul. Dans le cas d'une simulation uniquement basée sur VHDL-AMS ce rapport monte à un facteur de l'ordre de 338 000. L'évaluation de la précision des estimations n'est cependant pas considérée.

2.4 Contributions visées en vue d'améliorer les modèles de performances des architectures

Face à l'accroissement de la complexité de conception des architectures embarquées dans le domaine automobile, la maîtrise de cette complexité nécessite de disposer de modèles des architectures. Ces modèles favorisent l'échange entre les différents acteurs intervenant dans le processus de développement des architectures. Ces modèles rendent également possible la prise en considération, au plus tôt dans le processus de conception, de l'ensemble des ressources logicielles, de calcul, de mémorisation et de communication formant les architectures. Il est donc essentiel de pouvoir proposer des modèles permettant de capturer efficacement les caractéristiques des architectures afin de pouvoir dimensionner au plus tôt les ressources matérielles et logicielles requises.

Différentes approches et méthodes ont été proposées afin de faciliter la création de tels modèles. L'intérêt des approches basées sur la simulation réside dans le fait de permettre au concepteur d'analyser l'influence du comportement dynamique de l'architecture et d'observer l'émergence de phénomènes résultant de l'association des différents constituants de l'architecture. Dès lors, afin d'évaluer par simulation les caractéristiques des ressources matérielles et logicielles, les modèles d'architectures sont décrits dans des langages exécutables. Par la suite, nous adopterons une notation spécifique afin de représenter les modèles d'architectures. Pour autant notre contribution ne porte pas sur la définition d'une nouvelle approche de création de modèles. Elle porte sur l'amélioration possible de la description de ces modèles et de leur simulation.

Le paradigme de la modélisation transactionnelle des architectures a été introduit afin de rendre possible la création de modèles d'architectures décrits à différents niveaux de détail. Les travaux menés notamment dans [**54**] ont permis de quantifier le compromis pouvant exister entre la précision des modèles d'architectures et la rapidité de simulation.

Notre contribution développée dans le chapitre suivant vise donc à définir une technique de modélisation permettant d'améliorer le compromis entre la rapidité d'exécution et la précision des résultats des modèles d'architectures. Elle porte sur la proposition d'une technique de modélisation visant à faciliter pour l'architecte de systèmes la création de modèles transactionnels des architectures distribuées en vue de dimensionner les ressources matérielles et logicielles. Cette technique vise notamment à accélérer la simulation de ces modèles en conservant un niveau de précision donné, favorisant ainsi l'exploration de l'espace de conception.

Chapitre 3
Proposition d'une technique de modélisation en vue d'améliorer la simulation des modèles de performances des architectures distribuées

Sommaire

Ce chapitre présente la contribution principale de la thèse. Cette contribution porte sur la définition d'une technique de modélisation visant à améliorer le compromis entre le temps de simulation et la précision des modèles utilisés pour l'évaluation des performances des architectures. La mise en œuvre de cette technique est illustrée pour la modélisation des ressources de communication au sein des architectures.

Ce chapitre s'organise comme suit. La première partie expose l'approche et les notations adoptées pour représenter les architectures étudiées. La seconde section explique la technique proposée et son application pour la modélisation des ressources de communication. Dans la troisième section, nous cherchons à quantifier l'influence de la technique proposée sur l'accélération des temps de simulation et sur la précision des estimations fournies. Par la suite, nous illustrons, via un exemple didactique, l'application de la technique proposée sur la modélisation d'un protocole de bus de communication particulier. Finalement, nous dressons un bilan de la technique proposée.

3.1 Approche considérée pour la modélisation des architectures de systèmes embarqués

Dans cette section, nous exposons l'approche de modélisation adoptée afin de décrire les architectures étudiées. Une notation spécifique est utilisée afin de correctement capturer les propriétés des architectures en vue de l'évaluation de leurs performances. Dans nos travaux, les modèles créés en adoptant cette notation pourraient être saisis directement dans le langage SystemC afin d'être simulés. Afin de faciliter ce travail, nous utiliserons un outil spécifique de modélisation, à savoir CoFluent Studio [**58**].

3.1.1 Notations utilisées pour la modélisation des architectures

L'exemple représenté sur la Figure 3.1 rappelle l'organisation d'une architecture distribuée typique du domaine automobile et formée par plusieurs calculateurs. La Figure 3.1 fait la synthèse des figures 1.4 et 1.5 exposées au chapitre 1.

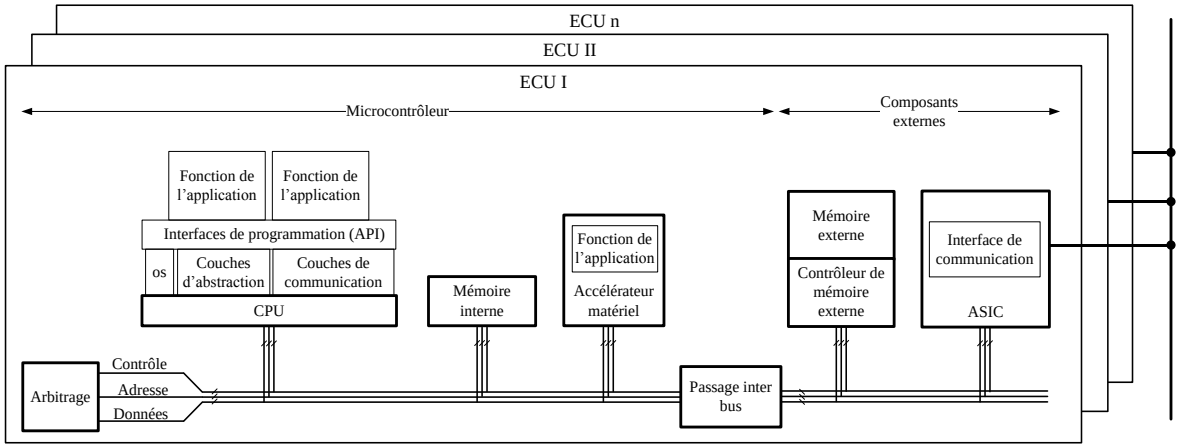


Figure 3.1 – Organisation caractéristique d'une architecture distribuée de calculateurs.

Dans la Figure 3.1 les ressources matérielles sont constituées de ressources de calcul, de mémorisation et de communication. Les ressources de calcul sont formées de ressources programmables (CPU) ou dédiées (accélérateur matériel, ASIC…). Les ressources de communication intra-ECU désignent les éléments assurant le transfert entre ressources de calcul et entre les ressources de calcul et les ressources de mémorisation (bus, arbitrage, passage inter-bus…). Les ressources logicielles désignent l'ensemble des ressources supportées par le processeur afin d'assurer l'exécution de l'application et d'utiliser les ressources matérielles disponibles. Les ressources de calcul supportent l'exécution des fonctions constituant l'application. Le terme fonction désigne ici un constituant de l'application et désigne également un constituant des ressources logicielles. Le terme relation désigne le couplage pouvant exister entre ces fonctions. Ainsi, les ressources de communication et de mémorisation servent respectivement à échanger et stocker les données associées à ces relations.

Les ECUs d'un système distribué implémentent des interfaces de communication inter-ECUs. Des ressources spécifiques sont nécessaires pour la mise en réseau. Ces ressources impliquent éventuellement la mise en place de ressources de calcul et de mémorisation

spécifiques et des ressources logicielles particulières. Dans le cas général, les fonctions associées sont organisées selon les couches de communication du modèle OSI. L'organisation de ces couches peut être définie sur un circuit dédié (ASIC) pour les couches proches de la couche physique. Les couches de niveaux supérieurs sont généralement mises en place en tant que fonctions exécutées par le processeur.

L'approche de modélisation considérée vise à la création de modèles utilisés pour l'évaluation des performances des architectures distribuées. Afin de pouvoir analyser le comportement dynamique des architectures étudiées cette évaluation s'effectue par simulation. Les performances analysées portent sur les ressources nécessaires de l'architecture pour respecter les contraintes de temps et de coût considérées. Comme évoqué au chapitre 2, la modélisation effectuée ne nécessite pas la description précise des fonctionnalités supportées par l'architecture. La modélisation consiste alors à exprimer l'influence de l'exécution des fonctions identifiées sur les ressources de calcul. Elle consiste également à exprimer l'utilisation des relations au sein des ressources de communication et de mémorisation de la plate-forme. Cette influence correspond aux notions de charges de travail (*workload*) et de traces utilisées dans d'autres travaux similaires cités au chapitre 2. La notation adoptée s'inspire de la notation définie dans [**59**]. Nous utilisons ici cette notation afin de décrire les modèles de performances des architectures étudiées. L'illustration de l'approche de modélisation est illustrée sur la Figure 3.2.

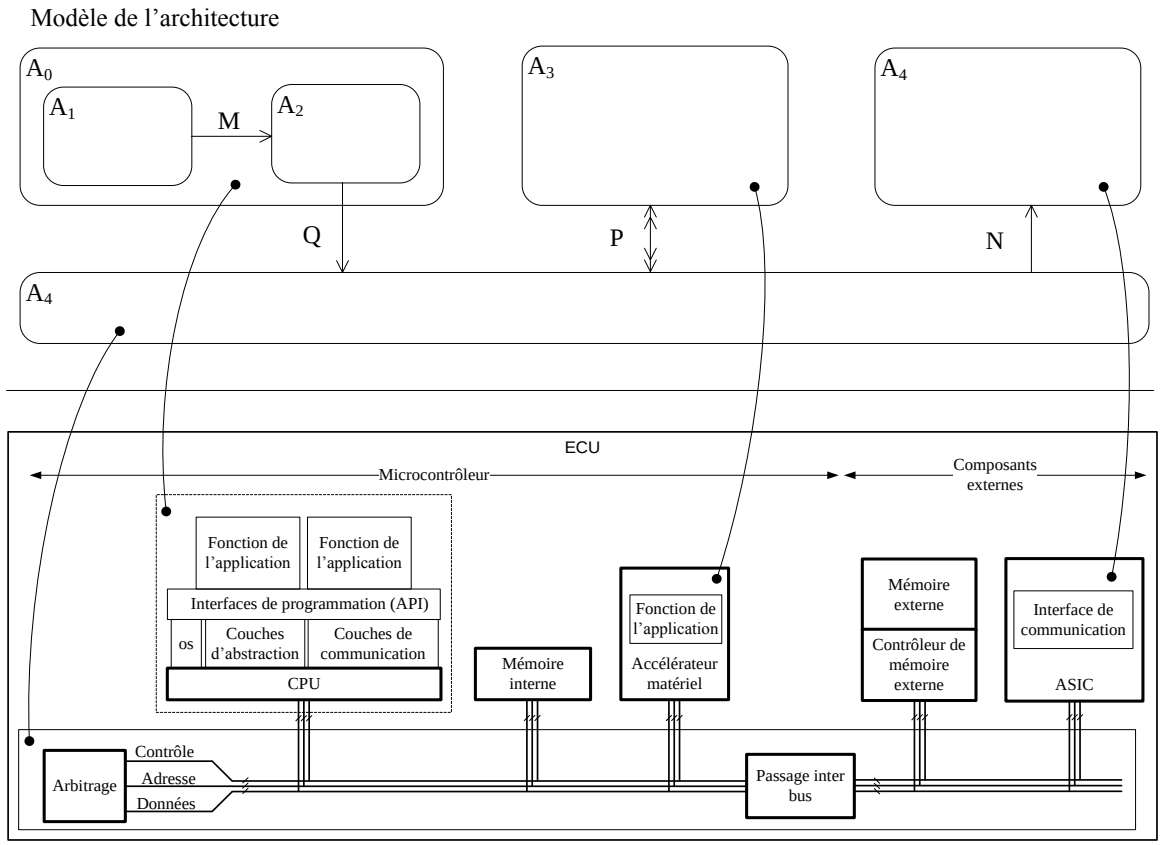


Figure 3.2 – Approche de modélisation en vue de l'évaluation des performances des architectures de systèmes embarqués communicants.

La partie haute de la Figure 3.2 donne une représentation possible comme modèle de l'architecture de l'élément ECU. La structure du modèle repose sur une notation sous forme de diagramme d'activité. Les activités identifiées représentent chacune une partie de l'architecture présentée sur la partie basse de la Figure 3.2. La représentation sous forme de diagramme d'activité fait apparaître les relations échangées entre activités. Une relation sert de support à l'échange de données entre activités. Selon la notation utilisée, une relation simple flèche correspond à une synchronisation ou à un échange synchronisé de données entre activités selon un protocole de type rendez-vous. Une relation double flèche correspond à une relation permanente, l'échange de données se fait alors sans synchronisation. Une activité décrit l'utilisation faite des ressources de la plate-forme, aussi bien les ressources de calcul que les ressources de communication ou de mémorisation. Une activité peut être décrite de manière séquentielle tel que présenté sur la Figure 3.3.

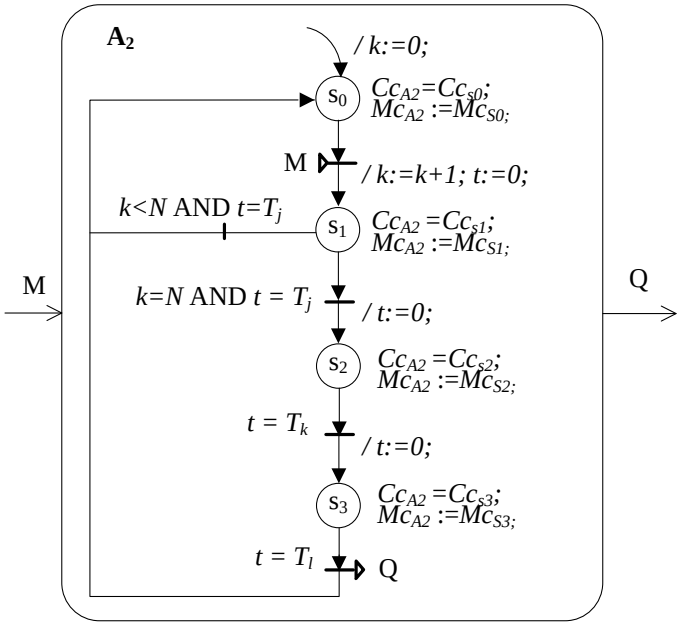


Figure 3.3 – Notation retenue pour l'expression du comportement des activités.

Le comportement séquentiel d'une activité exprime l'utilisation faite des ressources de la plate-forme. Ce comportement exprime la consommation et la production d'un événement de synchronisation avec ou sans données selon les relations d'entrée et de sortie de l'activité. Dans la Figure 3.3, le passage de s_1 à s_2 est effectué suite à la réception de N transactions M. Le comportement exhibe également les états représentatifs de l'utilisation des ressources de la plate-forme. Les transitions entre états se font selon des conditions d'attente d'événements de synchronisation avec ou sans données, des conditions logiques ou des conditions temporelles. Les informations temporelles utilisées correspondent à des indications estimées sur l'utilisation des ressources ou à des contraintes imposées sur l'exécution de la plate-forme. Le modèle comporte donc des indications temporelles approximatives. Dans la Figure 3.3, le passage de s_2 à s_3 est effectué suite à l'écoulement d'une durée T_k. Afin de disposer d'éléments pour la définition des caractéristiques de la plate-forme des propriétés sont ajoutées. Ces propriétés définissent les quantités de ressources utilisées pendant la durée d'un état de l'activité. Sur la Figure 3.3, ces propriétés

sont notées Cc_{A2} et Mc_{A2} et correspondent respectivement à la quantité de ressources de calcul utilisées et à la quantité de ressources de mémorisation utilisées. L'ensemble des variables utilisées pour la description du comportement peuvent être influencées par la valeur des données échangées via les relations identifiées. Deux types d'actions sont considérés dans l'automate. Les actions sur transition sont exécutées en temps nul au passage d'un état à un autre. Les actions sur états sont réalisées pendant toute la durée d'un état.

3.1.2 Moyens de simulation utilisés

L'analyse du modèle d'une architecture porte sur l'observation du comportement obtenu et de l'évolution des propriétés au cours du temps. L'utilisation d'un langage tel que SystemC pour la description du modèle permet de capturer les différents aspects modélisés et d'analyser l'exécution du modèle vis-à-vis du temps de simulation supporté par le simulateur. La Figure 3.4 donne l'évolution de l'activité A_2 compte tenu des transactions reçues et vis-à-vis du temps de simulation. L'observation de la propriété Cc_{A2} est également donnée.

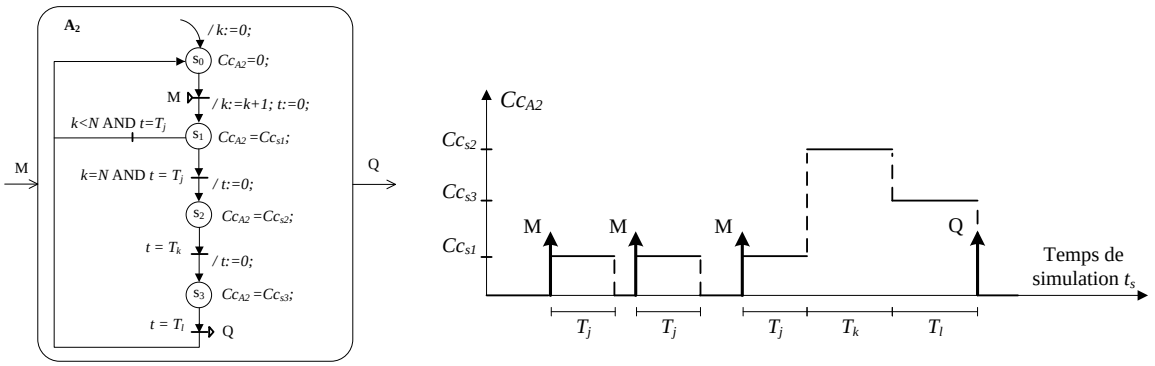


Figure 3.4 – Observation de l'évolution au cours de temps de Cc_{A2}.

Considérons le cas où l'activité A_2 présentée sur la Figure 3.4 consomme trois transactions M pour produire une transaction Q. A chaque réception de M, Cc_{S1} opérations de calcul sont effectuées pour une durée T_j. Après la réception et le traitement des trois transactions, Cc_{S2} et Cc_{S3} opérations sont effectuées pour des durées respectives T_k et T_l. A la fin des traitements une transaction Q est produite. Une telle observation est utilisée afin d'analyser le comportement temporel supposé de l'architecture ainsi que l'évolution des observations retenues pour quantifier les ressources de la plate-forme.

Dans le cadre de nos travaux, les modèles des architectures étudiées sont créés et simulés à l'aide de l'outil CoFluent Studio [**58**]. Cet outil est adapté pour la capture et l'évaluation de modèles d'architectures en s'appuyant sur l'approche en Y. Cet outil permet donc de créer séparément le modèle de l'application, le modèle de la plate-forme et le modèle de l'architecture. Dans le cadre de notre approche, nous utilisons uniquement la partie *Timed Behavioral Modeling* de cet outil. Cette partie permet la saisie graphique du modèle d'architecture que nous considérons. Les activités, les relations entre ces activités et les comportements sont exprimés à l'aide d'une notation propre à l'outil. Les algorithmes des actions associées aux comportements peuvent être également saisis. Des attributs

exprimant les propriétés des constituants (durées des opérations, durées de transfert de données ...) peuvent également être définis. L'outil permet ensuite de traduire la saisie graphique des modèles en SystemC et de les compiler afin d'obtenir une description exécutable des modèles saisis. Lors de l'exécution du modèle différents moyens d'analyse sont proposés afin de permettre l'observation de l'évolution du modèle.

3.2 Proposition d'une technique de modélisation en vue de l'amélioration des modèles de performance

3.2.1 Principe de la technique proposée

La technique proposée vise à améliorer les temps de simulation des modèles d'architectures utilisés pour l'évaluation des performances. Afin de réduire les temps d'exécution des modèles transactionnels, il est nécessaire de réduire le nombre de changements de contexte. Le nombre de changements de contexte est directement lié au nombre de transactions nécessaires au sein des modèles. Afin de réduire le nombre nécessaire de transactions, il est possible d'accroître le niveau de granularité des données échangées par transaction. En effet, un échange à un niveau de granularité plus important permet d'économiser l'ensemble des transactions devant être échangées par un modèle à un niveau de granularité plus fin. Par contre, il est nécessaire de modifier relativement la signification et l'impact des échanges à un niveau de granularité plus important afin de satisfaire aux besoins de précision des modèles créés. La Figure 3.5 détaille ce principe en considérant deux représentations distinctes d'un modèle transactionnel.

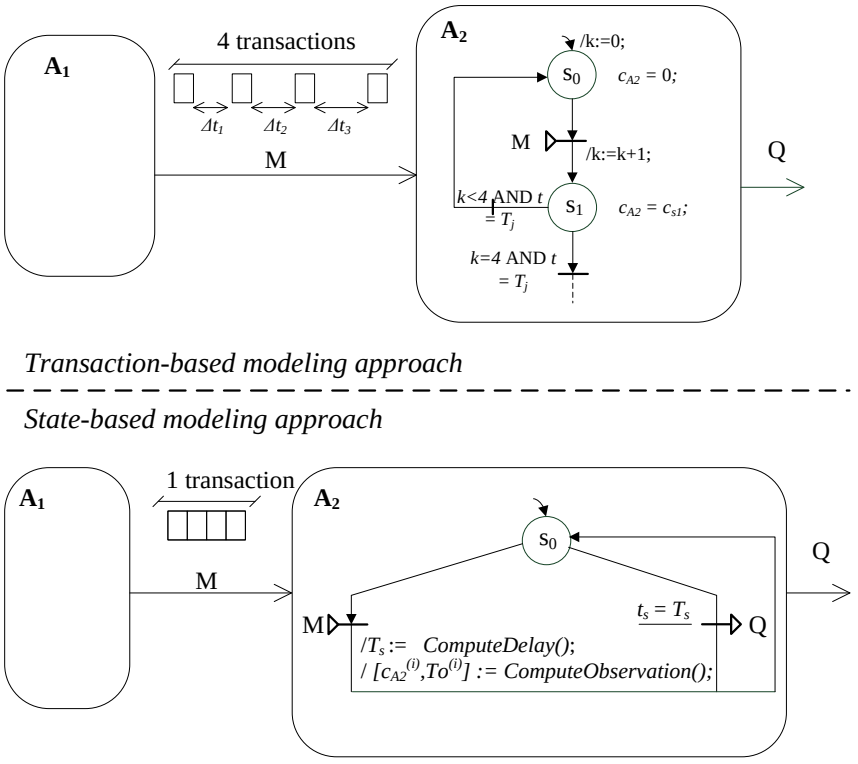


Figure 3.5 – Technique de réduction du nombre de transactions au sein de modèles transactionnels.

La partie haute de la Figure 3.5 correspond à un modèle d'architecture composé de deux activités échangeant des transactions via une relation notée M. Le comportement des activités exprime les échanges de transactions ainsi que l'utilisation faite des ressources de l'architecture modélisée. Cet exemple se focalise sur l'échange de 4 transactions espacées dans le temps par Δt_1, Δt_2 et Δt_3 ainsi que l'utilisation induite des ressources. L'utilisation des ressources de l'architecture modélisée s'exprime par l'évolution du paramètre noté c_{A2}. La réception ou la production de transactions induit un changement de contexte entre processus. Cette représentation indique que l'automate de l'activité A_2 ainsi que le paramètre c_{A2} évoluent à chaque réception d'une transaction M. Cette évolution étant guidée par les transactions, on qualifie ici une telle représentation de *transaction-based modeling*.

La partie basse de la Figure 3.5 représente un modèle similaire pour lequel la technique proposée est appliquée. Ce modèle exprime l'échange d'une transaction regroupant les données associées aux différentes transactions échangées dans le premier cas de figure. Afin de conserver une évolution équivalente de l'activité A_2, les instants de production des transactions Q, notés T_s, sont calculés lors de la réception de la transaction M par l'action notée *ComputeDelay()*. Afin de permettre une évolution du paramètre c_{A2} entre deux transactions une autre référence temporelle est définie. Cette référence temporelle est qualifiée de temps observé. L'intérêt de cette définition réside dans le fait qu'une telle évolution peut être calculée en un temps nul vis-à-vis du temps de simulation noté t_s sur la Figure 3.5. Ces calculs correspondent à l'action notée $[c_{A2}^{(i)}, T_o^{(i)}] := ComputeObservation()$. $c_{A2}^{(i)}$ et $T_o^{(i)}$ représentent respectivement les valeurs de la propriété c_{A2} et de l'instant d'évolution correspondant. L'évolution de ces propriétés se fait en parcourant un ensemble d'états internes. Sur l'exemple de l'activité A_2, il s'agit des états s_1, s_2 et s_3. Dans l'approche proposée, l'évolution des propriétés est faite par un parcours de ces états, effectué en temps nul vis-à-vis du temps de simulation. On qualifie une telle approche de modélisation de *state-based modeling*. Les observations possibles de ces deux modèles sont présentées sur la Figure 3.6.

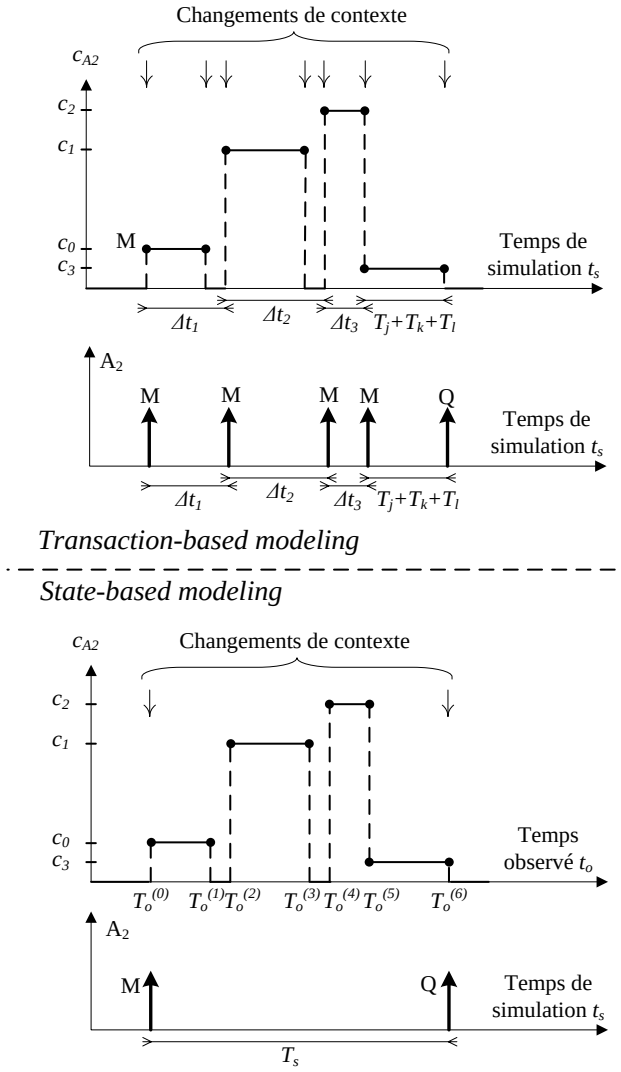


Figure 3.6 – Observations obtenues par application de la technique de simulation.

L'observation de l'évolution du paramètre c_{A2}, illustrée sur la partie haute de la Figure 3.6, est obtenue par la simulation du modèle obtenu selon une approche *transaction-based modeling*. Cette observation est exprimée en fonction du temps de simulation t_s. A chaque réception ou production d'une transaction, le paramètre c_{A2} peut évoluer. L'observation de l'évolution du paramètre c_{A2}, illustrée sur la partie basse de la Figure 3.6, est obtenue par la simulation du modèle obtenu selon une approche *state-based modeling*. Cette observation est obtenue à partir des couples $[c_{A2}^{(i)}, T_o^{(i)}]$ et elle est exprimée en fonction du temps observé noté t_o. L'échange des transactions est effectué en fonction du temps simulé.

Une telle représentation met en évidence la possible diminution du nombre de commutations de contexte lors de l'exécution d'un modèle créé selon l'approche *state-based*. Il est ainsi possible d'améliorer l'exécution d'un modèle de niveau transactionnel utilisé pour l'évaluation des performances en considérant qu'il est possible d'exécuter en temps nul vis-à-vis du simulateur l'observation de propriétés entre deux conditions d'attente de transactions. Cet aspect conduit à introduire la notion de temps observé, évoluant

localement entre deux transactions, et permettant d'éviter l'utilisation de primitives du type wait() dans la description.

La technique proposée repose tout d'abord sur un calcul des instants d'évolution. Ce calcul, noté *ComputeDelay()*, détermine le ou les instants d'évolution de l'activité. Ce calcul dépend des estimations faites sur les propriétés de l'élément de l'architecture modélisé et peut également dépendre du contenu de la transaction reçue. Ce contenu peut correspondre à un ensemble de données nécessaires afin d'effectuer le calcul défini. Il est important de noter que la méthode proposée n'implique pas systématiquement un accroissement du niveau de granularité des données véhiculées au sein des transactions. Le niveau de granularité dépend de la méthode de calcul utilisée pour définir les instants d'évolution. Dans l'étude de cas présentée au chapitre 4, différentes méthodes de calcul seront illustrées. Le second calcul effectué, noté *ComputeEvolution()*, sert à déterminer l'évolution des propriétés exprimant l'utilisation faite des ressources de l'architecture. Ces valeurs et ces instants peuvent dépendre du contenu des transactions et leur calcul suppose la définition d'estimations.

Un aspect important de la technique proposée porte sur le fait qu'elle ne se restreint pas au calcul de propriétés dont les datations sont inférieures à l'instant suivant d'évolution T_s. En effet, il est possible d'envisager un calcul permettant de définir l'évolution relative des propriétés à des instants ultérieurs à T_s. La Figure 3.7 illustre cet aspect.

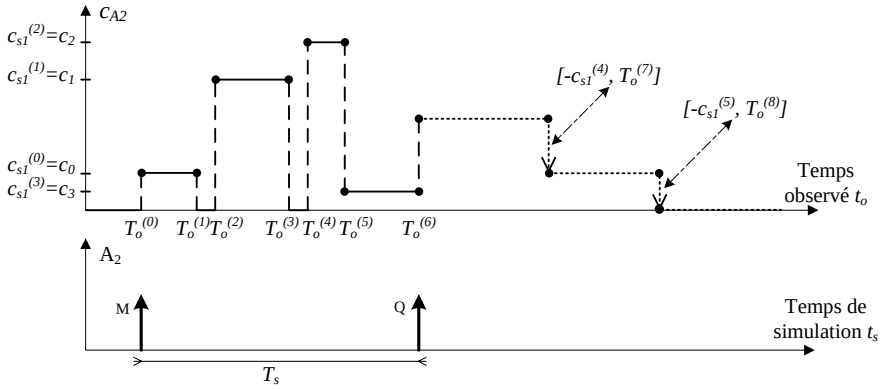


Figure 3.7 – Estimation de la fluctuation des propriétés par la technique proposée.

Sur la partie gauche de la Figure 3.7 les points d'évolution de la propriété, notés par le couple $[c_{A2}^{(i)}, T_o^{(i)}]$, sont calculés par l'action *ComputeObservation()*. Ce calcul peut porter sur des valeurs exactes ou sur des fluctuations de la propriété observée. Les valeurs exactes de la propriété sont calculées pour l'intervalle de temps compris entre les instants de réception et de production des transactions M et Q. La fluctuation de la propriété concerne les instants ultérieurs à l'instant de production de la transaction Q. Dans l'exemple présenté, deux fluctuations relatives de c_{s1} sont calculées à l'instant de réception de la transaction M. Une fluctuation négative est calculée pour l'instant $T_o^{(7)}$ et une deuxième fluctuation négative est calculée pour l'instant $T_o^{(8)}$. La partie droite de la Figure 3.7 montre ces fluctuations en flèche pointillée à la suite de la courbe d'évolution de la propriété.

La méthode ainsi décrite favorise la réduction du nombre de transactions nécessaires au sein de modèles transactionnels. Par la suite nous détaillons la mise en œuvre de cette méthode pour la modélisation des ressources de communication.

3.2.2 Application de la technique de modélisation proposée pour la modélisation des ressources de communications

L'application de la technique de modélisation proposée consiste à réduire le nombre de transactions strictement nécessaires au sein des modèles transactionnels tout en permettant l'observation des propriétés exprimant l'utilisation faite des ressources de l'architecture. L'application de ce principe au sein des modèles suppose de prendre certaines précautions afin d'assurer une bonne précision des estimations. Afin de préciser la mise en œuvre de la technique proposée nous considérons son application à la modélisation des ressources de communication au sein d'une architecture simple, utilisée également au chapitre 2. La Figure 3.8 illustre l'exemple considéré illustrant cette mise en œuvre.

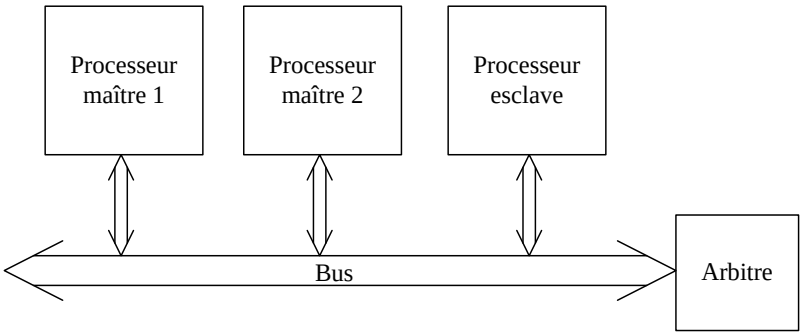


Figure 3.8 – Organisation des ressources matérielles d'une architecture multiprocesseur.

L'architecture ici considérée repose sur l'utilisation de deux processeurs maîtres et d'un processeur esclave. Le rôle de l'élément Arbitre est de correctement définir les accès au bus partagé compte tenu des priorités relatives de chaque processeur. Dans la suite, on considèrera des échanges des processeurs maîtres vers le processeur esclave. La modélisation d'une telle architecture peut être considérée à différents niveaux de précision :

- Dans le cas d'une modélisation au niveau CA, les différents éléments du modèle sont décrits relativement à une horloge de référence, le format des données échangées est élémentaire.
- Dans le cas d'une modélisation transactionnelle, il est possible de décrire les transferts de manières atomiques en considérant des temps approximatifs d'échange.

On suppose que pour ce bus le maître 1 est plus prioritaire que le maître 2 et que la transmission est attribuée toujours au maître le plus prioritaire. Il est donc possible que le maître 1 interrompe la transmission du maître 2. La Figure 3.9 illustre les évolutions associées aux transferts selon le niveau de précision considéré.

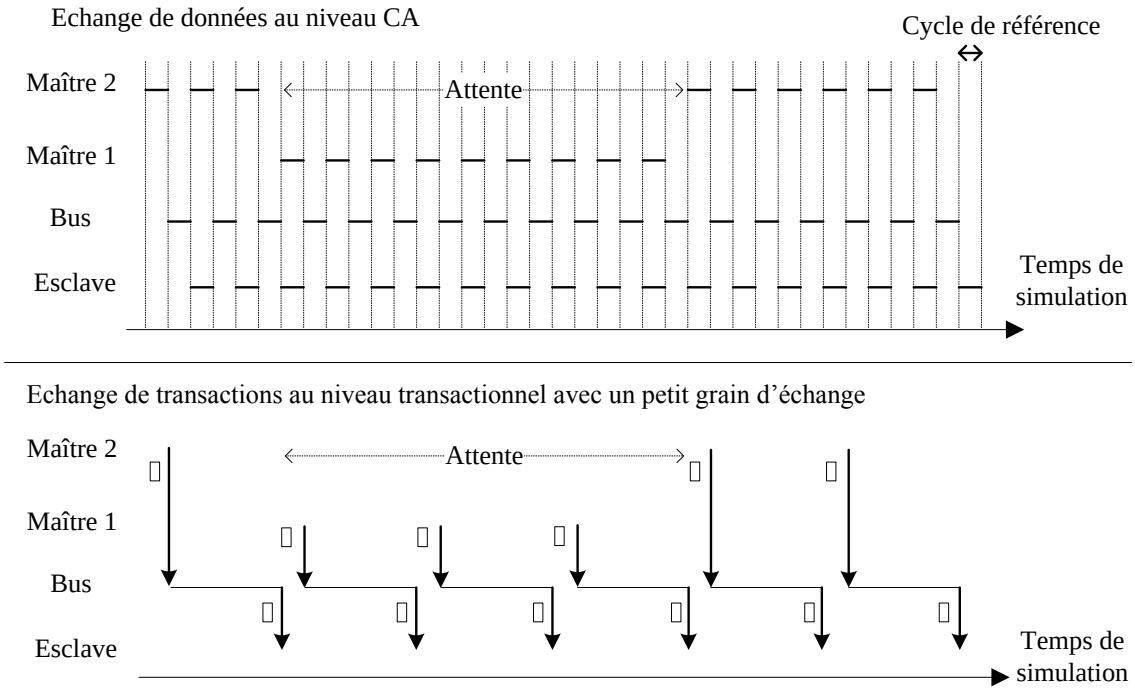


Figure 3.9 – Niveaux de représentation des modèles.

La Figure 3.9 représente deux évolutions possibles résultant de la modélisation de l'architecture considérée sur la Figure 3.8 à deux niveaux distincts. Sur la partie haute de la Figure 3.9, on observe que la transmission de l'ensemble des données du maître 2 est interrompue par la transmission de l'ensemble de données du maître 1. Sur la partie basse de la Figure 3.9, trois données élémentaires sont regroupées au sein d'une transaction. Dans la situation représentée, on suppose la modélisation transactionnelle suffisamment précise pour décrire la préemption possible du bus partagé. Aucune erreur sur les temps de réception de l'ensemble formé par trois données élémentaires n'est alors constatée par rapport à la modélisation au niveau CA. La Figure 3.10 illustre les échanges observables à partir d'un modèle transactionnel avec un grain plus important de données par transaction.

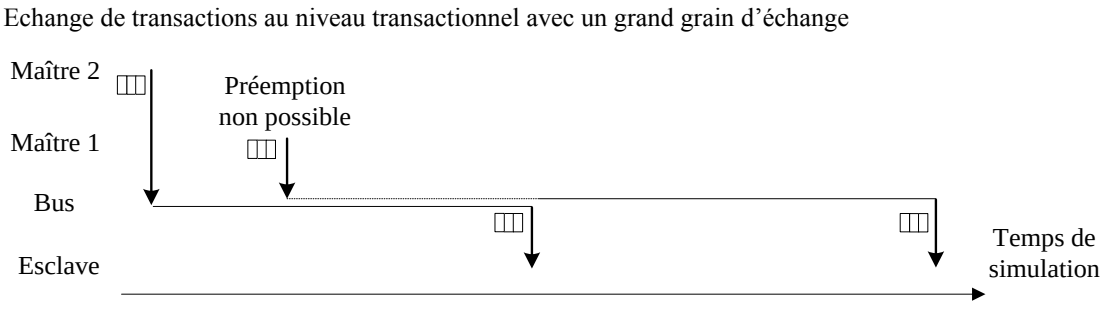


Figure 3.10 – Influence de l'augmentation du grain d'échange sur la précision des modèles transactionnels.

Dans ce modèle, neuf données élémentaires sont représentées par transaction. La Figure 3.10 illustre la situation où la modélisation faite considère un grain d'échange plus important et donc ne permet pas une description correcte des préemptions éventuelles sur le bus.

Cet exemple simple illustre le défaut possible d'un accroissement du niveau de granularité des échanges. Ainsi, il est nécessaire d'adapter la technique précédemment proposée afin d'éviter de mauvaises prédictions des instants de production et des observations. La Figure 3.11 présente le diagramme d'activité du modèle du bus considéré dans la situation de la Figure 3.8 et en utilisant la technique proposée.

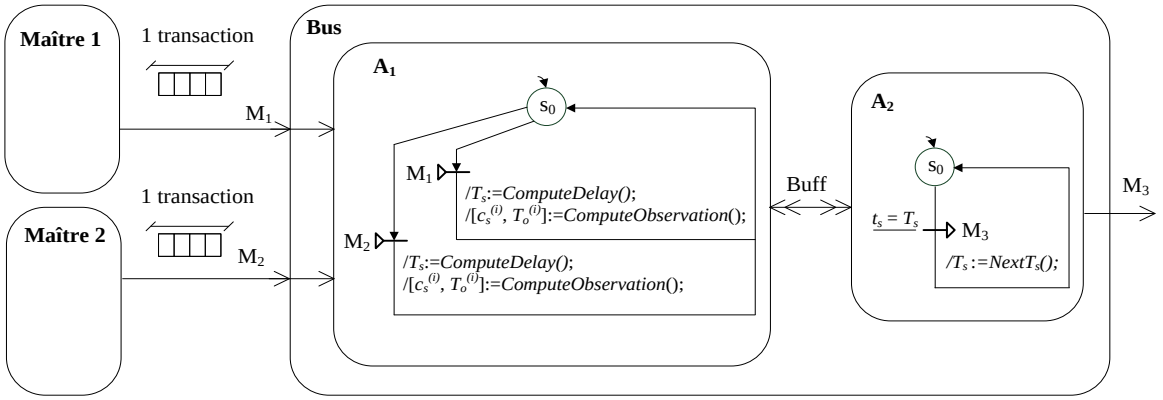


Figure 3.11 – Diagramme d'activité du modèle générale d'un modèle de bus utilisant la technique proposée.

Sur la Figure 3.11, l'activité modélisant le bus de communication est constituée de deux activités. Les activités notées A_1 et A_2 permettent respectivement de consommer les transactions en entrée et de produire les transactions en sortie. Les actions *ComputeDelay()* associées à la réception des transactions en entrée conduisent au calcul des instants de production des transactions en sortie. Ce calcul prend en compte différents aspects :

- les caractéristiques du bus modélisé, correspondant notamment à la vitesse de transfert et à la stratégie de gestion de la préemption selon les priorités des nœuds ;
- la taille des données représentées par la transaction reçue ;
- la priorité du nœud émetteur ;
- les instants de production déjà calculés et encore non atteints afin de pouvoir déterminer l'instant correct de transmission.

Ainsi, une fois les calculs des différents instants effectués, *ComputeDelay()* définit l'instant suivant de production T_s. Pour cela, *ComputeDelay()* peut également effectuer des mises à jour des instants de production calculés et encore non atteints pour les transactions précédentes. En effet, dans le cas où une préemption est possible, il est nécessaire de redéfinir les valeurs de ces instants. La mise à jour dépend ici des caractéristiques du bus modélisé et de la durée de transfert calculée pour les données représentées par la transaction reçue. Sur l'exemple de la Figure 3.11, le résultat de ce calcul est stocké au sein de la variable notée « Buff ». Cette variable est indispensable pour rendre possible ce mécanisme de mise à jour. Elle contient une liste ordonnée dans l'ordre chronologique des valeurs des instants de production T_s. Cette variable peut contenir également les informations contenues dans les transactions. Dans le cas où il resterait des transactions à produire, l'action *NextTs()*

permet de définir l'instant de production suivant T_s à partir de la liste contenue dans la variable « Buff ».

Afin d'illustrer ces différents principes, on applique la technique de modélisation proposée pour la modélisation de l'architecture présentée dans la Figure 3.8. La Figure 3.12 illustre l'observation obtenue pour la situation précédente.

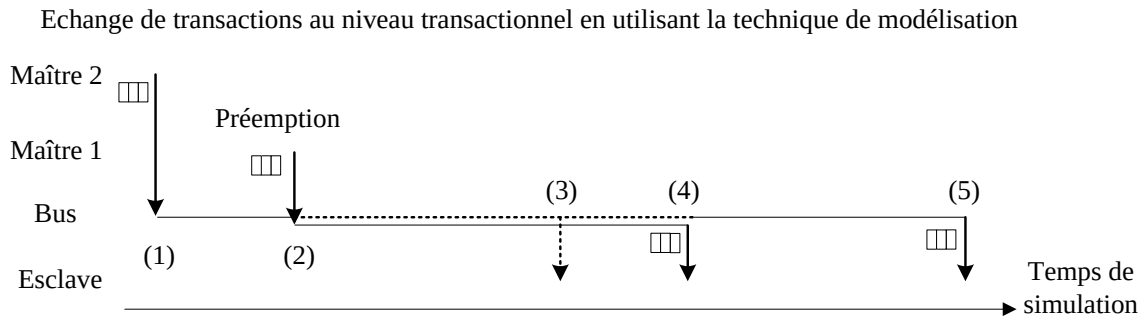


Figure 3.12 – Application de la technique proposée pour la modélisation d'un bus de communication.

Sur la Figure 3.12, à l'instant (1) *ComputeDelay()* calcule la durée de transfert de données représentées par la transaction reçue et définit l'instant de sa production T_s à (3). A l'instant (2) *ComputeDelay()* calcule la durée de transfert de données représentées par la nouvelle transaction reçue et, dans la mesure où cette transaction provient du nœud le plus prioritaire, rajoute la durée de transfert calculée à l'instant actuel (2) afin de définir l'instant de production T_s de cette transaction à (4). *ComputeDelay()* met à jour l'instant de production de la première transaction en rajoutant la durée de transfert de la transaction en cours. Cet instant correspond à (5) et il est enregistré dans la relation « Buff ». A l'instant (4), A_2 produit la transaction du maître 1 et *NextTs()* définit à partir de « Buff » l'instant de production de la transaction du maître 2 à (5). Selon ce principe, même en augmentant le grain d'échange, la préemption est prise en compte et les fins des transferts sont effectuées correctement comme obtenues par le modèle de niveau CA.

Après exécution de l'action *ComputeDelay()*, l'évolution des propriétés à observer suite à la réception de la transaction est calculée par l'action *ComputeObservation()*. On rappelle que ce calcul se base sur des estimations basées sur un ensemble de valeurs $[c_s^{(i)}, T_0^{(i)}]$ prédéfinies et qui peuvent dépendre du contenu de la transaction. Comme expliqué précédemment, ces valeurs sont calculées au sein de *ComputeObservation()* et sont obtenues à partir du parcours des états internes du modèle de l'architecture. Ensuite, *ComputeObservation()* organise ces valeurs afin de définir l'observation globale de la propriété. Cette organisation prend effet à partir de l'instant supposé de la consommation de la transaction reçue. La Figure 3.13 illustre ce mécanisme pour le modèle présenté dans la Figure 3.11.

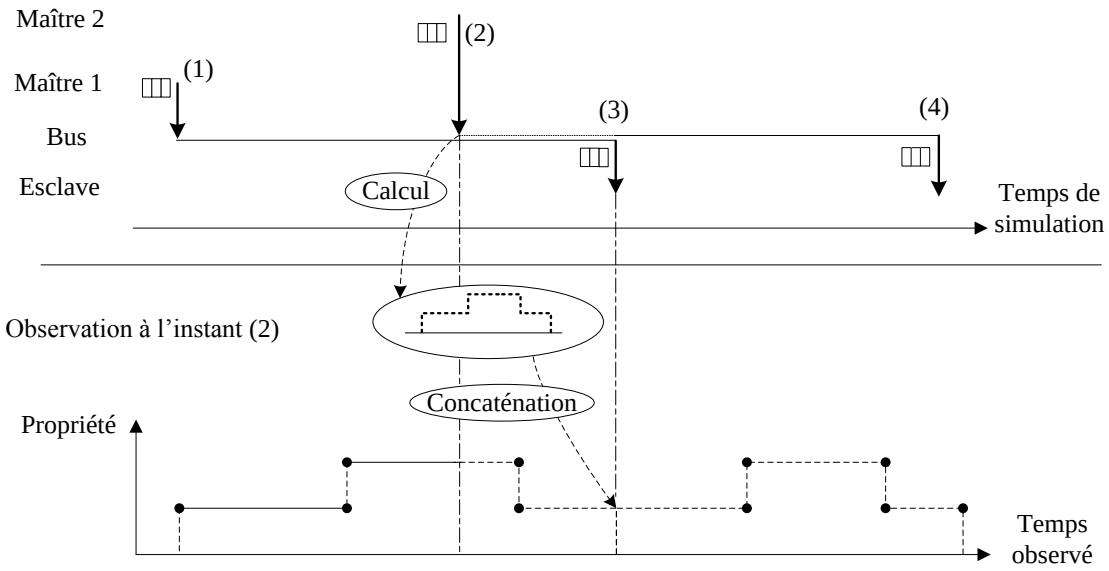


Figure 3.13 – Calcul des propriétés et organisation des valeurs.

La Figure 3.13 détaille l'observation des propriétés lors de la simulation du modèle à l'instant (2). A cet instant, une observation couvrant l'intervalle allant de (1) à (3) est déjà calculée suite à la réception d'une transaction du maître 1 en (1). A l'instant (2), le maître 2 envoie une transaction. Après le calcul de *ComputeDelay()*, (3) est supposé être l'instant de consommation de cette transaction et (4) l'instant de production. *ComputeObservation()* calcule les valeurs de la propriété induite par la réception de cette transaction. Ces valeurs sont alors combinées avec les valeurs déjà calculées. Cette combinaison s'effectue à l'instant (3).

La prise en compte de la préemption conduit à effectuer des mises à jour des propriétés déjà calculées. Cette mise à jour concerne les transactions dont les instants de production ont été mis à jour par *ComputeDelay()*. Les couples $[c_s^{(i)}, T_o^{(i)}]$ calculés suite à la réception de ces transactions sont concernés par la mise à jour. Parmi ces couples, seuls les couples dont les instants $T_0^{(i)}$ sont ultérieurs à l'instant supposé de la consommation de la transaction en cours sont concernés par la mise à jour. La mise à jour consiste au décalage des instants $T_0^{(i)}$ de ces couples d'une durée équivalente à la durée du transfert de la transaction en cours. La Figure 3.14 explique ce mécanisme.

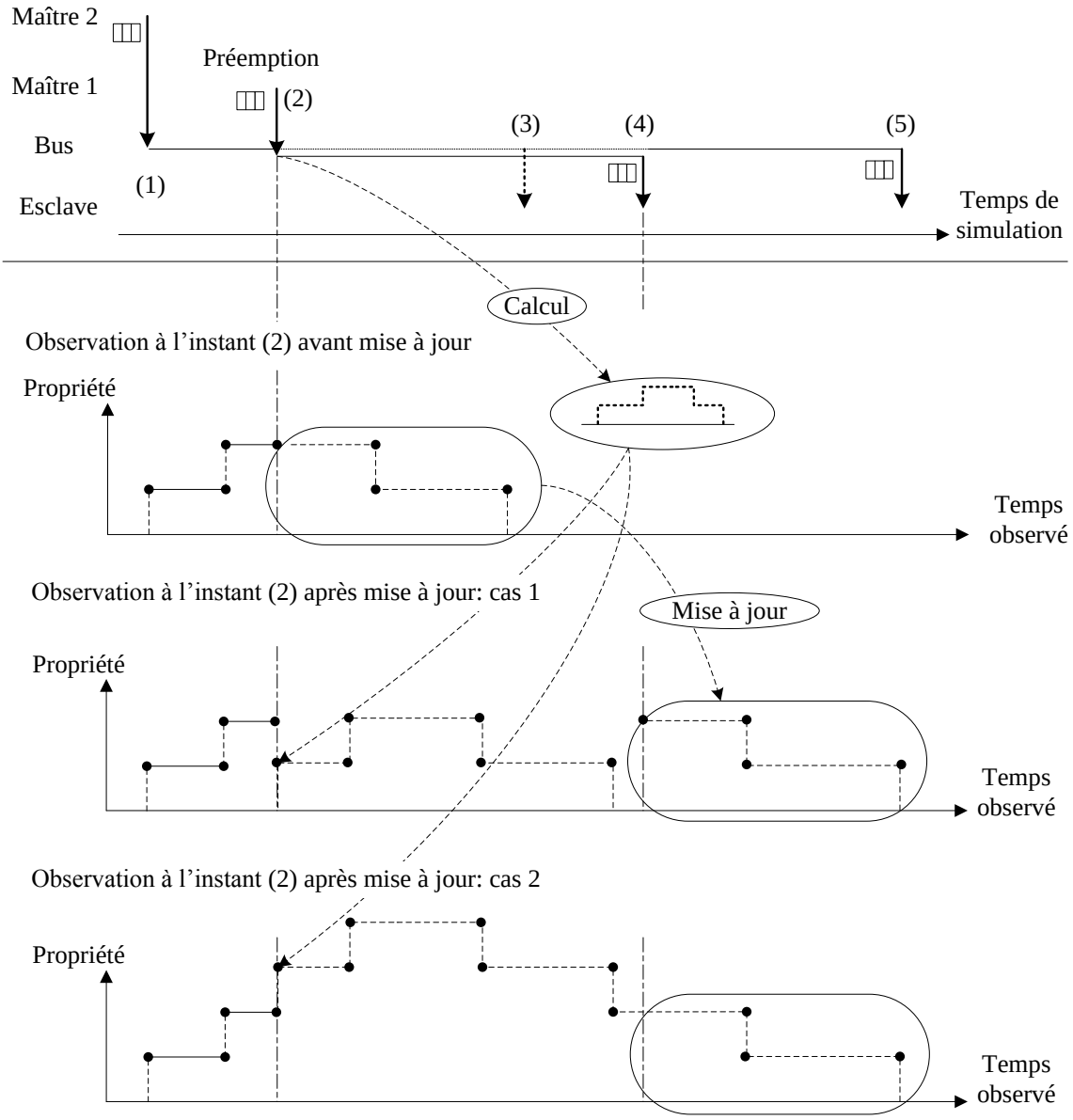


Figure 3.14 – Stratégie de gestion des observations lors de la simulation.

La Figure 3.14 détaille l'observation de propriétés lors de la simulation du modèle à l'instant (2). A cet instant, une observation couvrant l'intervalle allant de (1) à (3) est déjà calculée suite à la réception d'une transaction du maître 2 en (1). A l'instant (2), le maître 1 envoie une transaction. Après le calcul de *ComputeDelay()*, l'instant courant (2) est supposé être l'instant de consommation de cette transaction et (4) l'instant de production. Une mise à jour de (3) vers (5) de l'instant de production de la première transaction est également effectuée. Dans le même temps, *ComputeObservation()* calcule les valeurs de la propriété induite par la réception de cette transaction. Dans la mesure où *ComputeDelay()* a effectué une mise à jour, l'observation déjà calculée pour les instants ultérieurs à (2) est mise à jour en décalant ces points. Ce décalage correspond à la durée définie entre (2) et (4). Les valeurs calculées sont ensuite combinées avec l'observation déjà calculée à l'instant en cours (2). Deux exemples de mises à jour sont illustrés. Le premier cas présente une

combinaison indépendante de la valeur actuelle de la propriété observée. Le deuxième cas correspond à une combinaison additive. Dans ce cas, la valeur actuelle de la propriété observée est rajoutée aux valeurs de la suite calculée.

3.2.3 Positionnement de la contribution

Vis-à-vis des travaux présentés dans [**54**], la technique proposée consiste également en un calcul dynamique des instants de transfert et à une correction de ces instants compte tenu des perturbations induites par l'environnement extérieur. Une première différence porte sur la gestion des instants corrigés. La stratégie mise en place dans [**54**] conduit à l'ajout de *wait* supplémentaires afin de corriger les instants mis à jour. Une autre différence porte dans la gestion faite des observations des propriétés associées au bus modélisé. Dans [**54**], les travaux présentés n'abordent que la correction des instants d'évolution tandis que dans le cadre de nos travaux nous considérons également la prise en compte de l'observation des ressources de l'architecture.

Dans [**55**] et dans [**57**], les techniques proposées s'appuient sur une combinaison de différents niveaux de représentation afin de d'assurer une bonne détection de la préemption sur le bus. Dans le cas de notre proposition, la détection de la préemption est jugée possible au niveau d'abstraction considérée. Pour autant, les observations considérées peuvent correspondre à un niveau de précision plus fin que celui considéré pour la simulation.

3.3 Qualification de la technique de modélisation proposée

L'objet de cette section porte sur la qualification de la technique proposée vis-à-vis des critères de temps de simulation et de précision des estimations fournies.

3.3.1 Qualification de l'apport de la technique sur l'accélération des temps de simulation

Dans un premier temps, nous avons cherché à quantifier l'influence de la réduction du nombre de changements de contexte sur les temps de simulation. Pour cela, nous avons effectué plusieurs simulations présentant un nombre de changements de contexte différent afin de mesurer les temps d'exécution associés. Toutes les simulations sont effectuées avec le même modèle, le même scénario de fonctionnement et avec la même taille de données échangées par transaction. Le modèle utilisé consiste en deux activités échangeant des transactions via une relation selon le principe de la Figure 2.12 du chapitre 2. Ce modèle a été créé avec l'outil CoFluent Studio. Les résultats de simulation sont obtenus à partir d'une machine à base d'un processeur Intel Core 2 Duo cadencé à 2.66 GHz. Les mesures obtenues par simulation sont présentées dans le Tableau 3.1.

Nombre de transactions par simulation	1000	10 000	100 000	1 million	10 millions
Temps d'exécution (ms)	31	141	1156	11 297	112 984

Tableau 3.1 – Mesures des temps de simulation des modèles testés.

Le Tableau 3.1 donne les mesures de temps de simulation obtenus pour plusieurs exécutions du même modèle en faisant varier le nombre de transactions échangées au cours

de la simulation. Le Tableau 3.2 donne les valeurs des ratios des temps de simulation en comparant deux à deux les simulations qui ont été effectuées selon le Tableau 3.1 .Dans chaque case, le ratio du temps de simulation mesuré est donné, soit :

$$ratio\ [i,j] = \frac{temps\ simulation\ ligne\ i}{temps\ simulation\ colonne\ j}$$

Les ratios exprimés entre parenthèse représentent les ratios du nombre de transactions.

Nombre de transactions par simulation	1000	10 000	100 000	1 million	10 millions
10 millions	3644,6 (10 000)	801,3 (1000)	97,7 (100)	9,9 (10)	1 (1)
1 million	364,4 (1000)	80,1 (100)	97,7 (10)	1 (1)	
100 000	37,2 (100)	8,1 (10)	1 (1)		
10 000	4,5 (10)	1 (1)			
1000	1 (1)				

Tableau 3.2 – Ratios des temps de simulation mesurés.

La première colonne et la première ligne du Tableau 3.2 indiquent le nombre de transactions échangées au cours des deux simulations à comparer.

Pour la première colonne (1000 transactions), les écarts entre les ratios des temps de simulation mesurés et les rapports des nombres de transactions par simulation sont importants. Cet écart s'explique par le fait que le temps de simulation comprend les temps d'exécution des différentes phases pour la simulation indiquées dans la Figure 2.11. En effet, pour une simulation contenant 1000 transactions, les temps d'exécution des phases d'évaluation et d'avancement du temps de simulation sont du même ordre que les autres phases d'élaboration, d'initialisation et de nettoyage. Pour les simulations où le nombre de transactions est important, les temps d'exécution des phases d'élaboration, d'initialisation et de nettoyage deviennent négligeables par rapport aux autres phases.

Les simulations où les temps d'exécution des phases d'évaluation et d'avancement du temps de simulation sont plus importants par rapport aux autres phases correspondent aux cas avec plus de 100 000 transactions échangées. Pour ces simulations, les mesures révèlent des ratios des temps de simulation mesurées proches du ratio du nombre de transactions. Pour un nombre important de transactions, nous pouvons donc faire l'observation qu'en gardant la même taille de données échangées par transaction, une variation de N% du nombre de transactions entraine une variation de N% sur le temps de simulation des modèles transactionnels.

Cette première observation faite on peut noter que l'application de la technique proposée peut conduire à une augmentation de la taille de données échangées par transaction. En effet, il peut être nécessaire de préserver l'intégralité ou une partie des données associées aux transactions économisées par application de la technique. Ces données peuvent servir aux calculs des instants d'évolution et aux observations.

Dans la suite, nous étudions l'influence de la taille des données échangées par transaction sur le temps de simulation. Tout d'abord, expliquons ici le mécanisme d'échange des données lorsqu'une transaction intervient entre deux activités lors de la simulation. La Figure 3.15 explique ce mécanisme. Il existe alors un mécanisme transparent de copie et de recopie de données contenues dans les relations lors d'une transaction. Le mécanisme de copie vers la relation intervient afin de modifier le contenu d'une relation. Le mécanisme de copie depuis la relation est nécessaire pour préserver les données associées à la transaction lors de l'échange, notamment si une nouvelle transaction modifie le contenu de la relation.

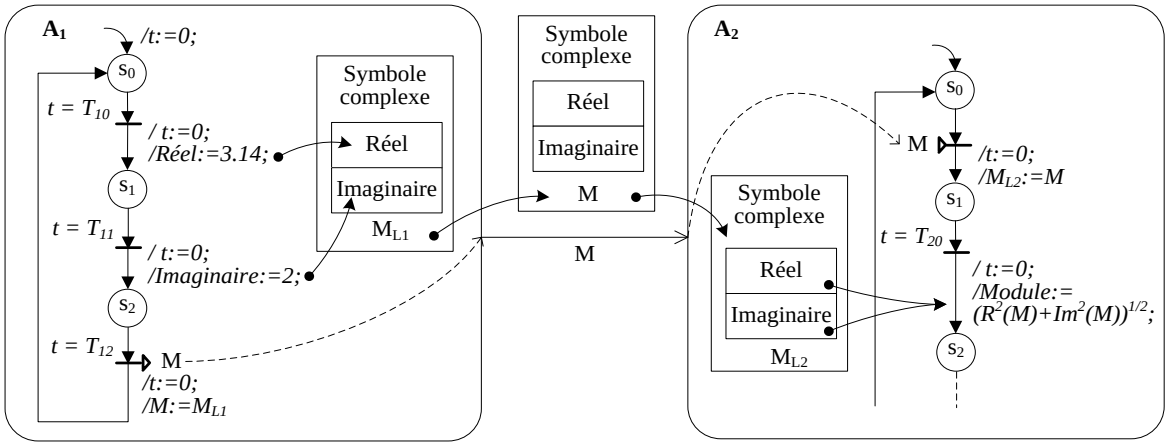


Figure 3.15 –Mécanisme de copie et de recopie des données contenues dans les relations.

L'exemple est constitué de deux activités A_1 et A_2 et d'une relation M. Cette relation permet l'échange de transactions et d'une valeur d'un nombre complexe associé à la transaction. Les activités A_1 et A_2 disposent, respectivement, d'images de la relation M notées M_{L1} et M_{L2}. L'activité A_1 modifie en local les champs de l'image M_{L1}. A la transmission d'une transaction, l'ensemble du contenu de l'image locale est copié dans la relation M. Réciproquement, à la réception d'une transaction, l'activité A_2 copie dans l'image locale le contenu de la relation M.

Le mécanisme de copie et de recopie de données contenues dans les relations se traduit donc par des accès à la mémoire au sein de la machine qui exécute le modèle. Ce mécanisme induit des temps supplémentaires à la simulation. Plus la taille des données copiées et recopiées sont importantes plus les temps de simulation sont importants. La Figure 3.16 montre l'influence de la taille des données contenues dans les relations sur les temps de simulation.

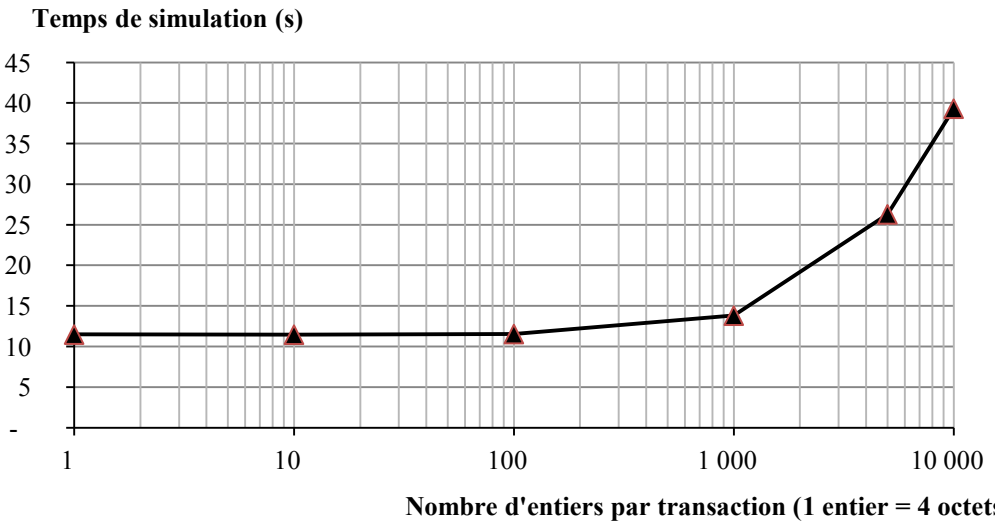


Figure 3.16 – Influence de la taille de données contenues dans les relations sur les temps de simulation.

La courbe de la Figure 3.16 est obtenue par la simulation du modèle utilisé précédemment afin d'obtenir les résultats donnés sur le Tableau 3.1. Cette courbe présente les temps de simulation obtenus à partir de plusieurs simulations avec le même nombre de changements de contexte. Chaque simulation correspond à 500 000 transactions échangées. A chaque simulation, on fait varier la taille des données contenues dans une transaction. Une transaction contient un tableau d'entier. On note qu'à partir de 1000 entiers par transaction le temps de simulation augmente fortement. Cette variation illustre l'influence de la taille des données sur les temps de simulation. Cette influence se traduit par un ralentissement des simulations dès lors que la taille des données devient trop importante. On peut remarquer que l'évolution de ce ralentissement en fonction de la taille de données est non linéaire. Ce ralentissement peut augmenter considérablement quand la limite matérielle de la machine qui exécute la simulation est atteinte. Cette limite peut se manifester par l'utilisation du disque dur de la machine comme une extension de la mémoire RAM lorsque celle-ci est saturée. Les simulations pour ce cas ne sont pas représentées dans la Figure 3.16. Dès lors, le temps de simulation devient dépendant de la machine d'exécution et la mesure est alors incertaine.

A ce niveau, nous avons illustré successivement la dépendance des temps de simulation des modèles vis-à-vis du nombre de transactions échangées au cours de la simulation et des tailles de données échangées par transaction. L'application de la technique proposée permet de réduire le nombre de transactions échangées au cours de la simulation mais peut conduire à une augmentation des tailles de données échangées par transaction. Ces deux facteurs ont des influences opposées sur les temps de la simulation dans des proportions différentes. Ainsi, il est intéressant d'analyser l'influence combinée de ces deux facteurs. Il s'agit d'étudier le gain potentiel en temps de simulation lorsqu'on diminue le nombre de transactions et qu'on augmente d'autant la taille des données échangées par transaction. La Figure 3.17 illustre la variation du facteur d'accélération en fonction de ces deux facteurs.

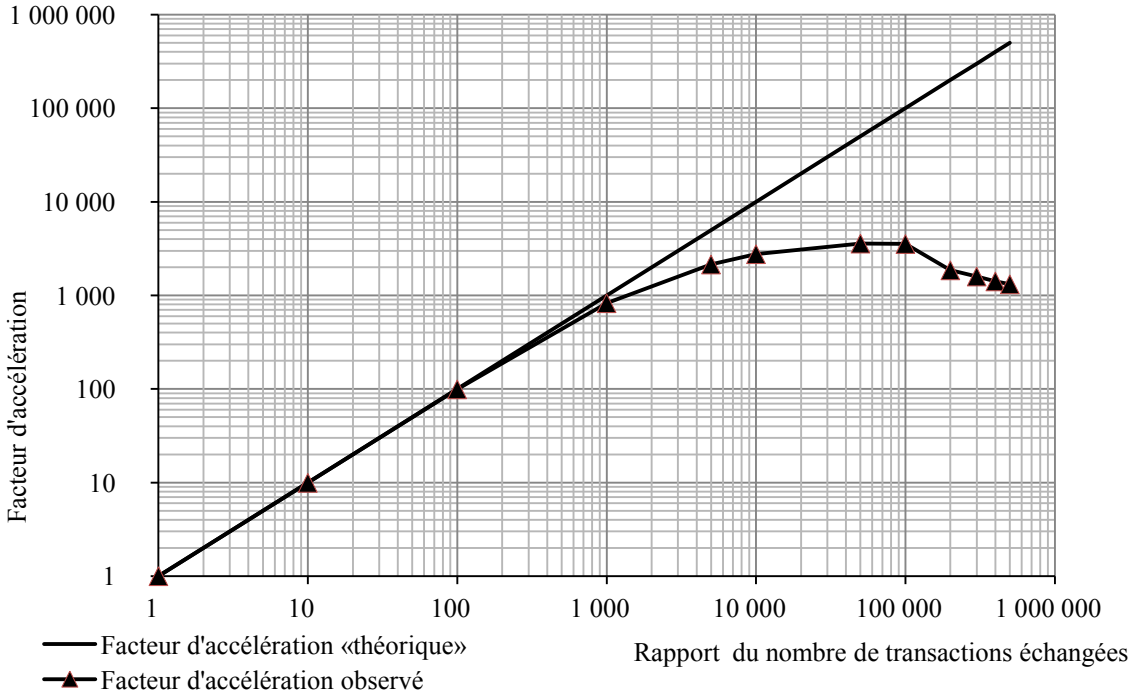


Figure 3.17 – Facteur d'accélération en fonction du compromis entre le nombre de transaction économisées et la taille de donné par transaction.

La Figure 3.17 présente deux courbes. La première courbe correspond à l'évolution « théorique » du facteur d'accélération obtenu dès lors qu'on réduit le nombre de transactions sans augmenter la taille des données. Dans cette situation, lorsqu'on diminue, par exemple, d'un facteur 10 000 le nombre de transactions, on obtient un facteur d'accélération de 10 000. La seconde courbe représente l'évolution du facteur d'accélération observé en réduisant le nombre de transactions au cours de la simulation et en augmentant d'autant la taille de données échangées par transaction. Cette courbe est obtenue en comparant les temps de simulation mesurés par rapport au temps d'une simulation de référence. Le point 100 000 de l'axe horizontal correspond à la simulation où 5 000 transactions sont échangées avec 100 000 entiers échangés par transaction. Ce point correspond au sommet de la courbe « Facteur d'accélération observé». A partir de ce point la dégradation du facteur d'accélération mesuré devient importante. Cette dégradation peut s'expliquer par l'atteinte de la limite matérielle de la machine.

La zone située entre les deux courbes correspond au domaine du facteur d'accélération obtenu selon le compromis entre le nombre de transactions économisées et la taille de données échangées par transaction. Cette zone représente l'espace dans lequel peut se situer le gain potentiel en temps de simulation en appliquant la technique de modélisation proposée.

Après avoir étudié le facteur d'accélération envisageable par application de la technique de modélisation proposée, l'influence de son application sur la précision des estimations fournies par les modèles est étudié.

3.3.2 Qualification de l'apport de la technique sur la précision des modèles

La technique de modélisation proposée peut conduire à accroitre le niveau de granularité des transactions afin de réduire le nombre des transactions échangées lors de la simulation. Cette sous-section vise à qualifier l'influence de la technique proposée sur la précision des modèles de ressources de communication. Pour cela nous allons considérer différents niveaux de représentation de l'architecture modélisée sur la Figure 3.18.

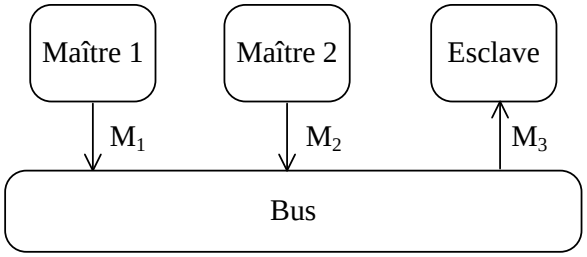


Figure 3.18 – Modèle de l'architecture du cas d'étude considéré.

L'architecture présentée est constituée d'un bus et de trois nœuds. On suppose que les deux maîtres transmettent des données à l'esclave et on suppose que le maître 1 est plus prioritaire que le maître 2. L'accès au bus est toujours attribué au nœud le plus prioritaire. La transmission via le bus de données en provenance du maître 2 peut donc être interrompue une ou plusieurs fois par le maître 1 si celui-ci demande l'accès au bus. Si le maître 2 demande l'accès au bus alors que le maître 1 est en train de transmettre ses données, cette demande est mise en attente.

Par la suite nous allons considérer trois niveaux de modélisation de cette architecture :

- une modélisation simplifiée de l'architecture à un niveau CA qui servira de référence pour mesurer la précision des résultats;
- une modélisation transactionnelle simplifiée de l'architecture permettant d'illustrer les erreurs possibles et de les quantifier;
- une modélisation transactionnelle utilisant la technique de modélisation proposée et permettant de justifier de la validité de la mise en œuvre de cette technique.

La Figure 3.19 illustre la perte possible de précision, par rapport à un modèle de niveau CA, en considérant un modèle transactionnel du bus.

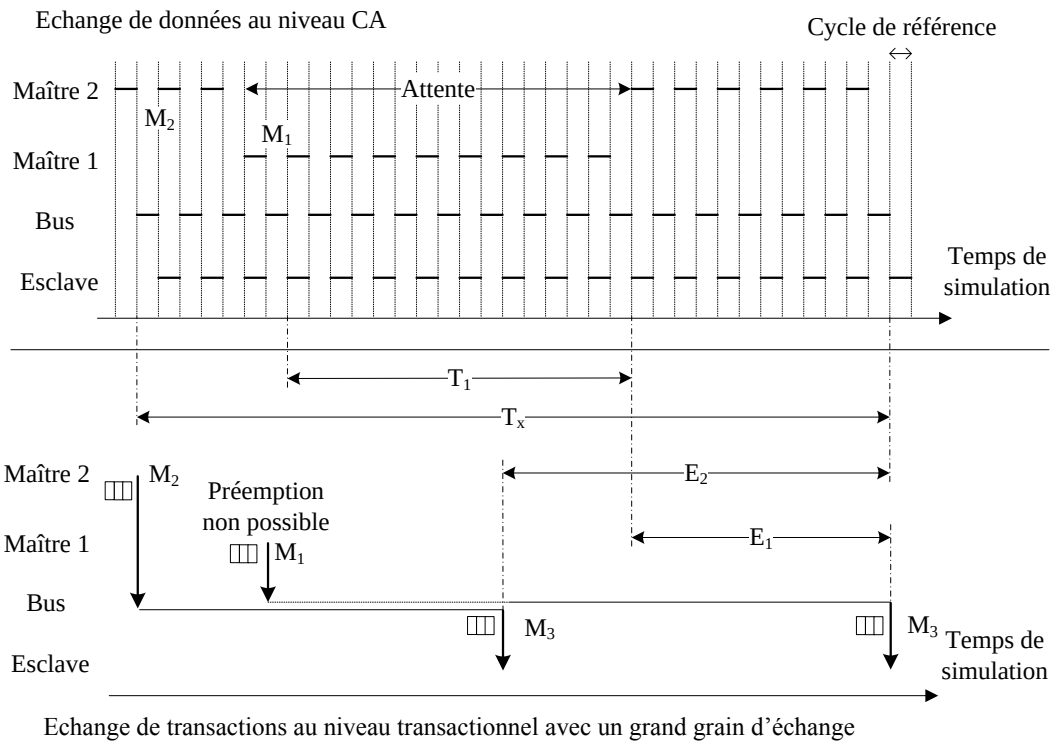


Figure 3.19 – Erreurs des estimations dues à l'augmentation du grain d'échange.

Sur la Figure 3.19, on constate, lors de l'envoie de la transaction M_1, que la préemption n'est pas prise en compte et que cette transaction est mise en attente de la fin de l'envoie de la transaction M_2. Pour comparer les résultats des deux modèles, on se base sur les instants où le nœud esclave dispose de la totalité d'un ensemble de données. Dans le modèle transactionnel, ces instants correspondent aux instants de réception des transactions. Dans le modèle CA, ces instants correspondent aux instants de réception des derniers octets qui complètent les ensembles de données transmis. Pour un même ensemble de données, la différence entre ces instants dans les deux modèles représente l'erreur sur la réception de données. Ces erreurs sur la réception de données sont notées E_1 et E_2 sur la Figure 3.19.

Afin de quantifier les erreurs des résultats des modèles transactionnels, on procède à une normalisation des erreurs par rapport au modèle CA. On définit alors le taux d'erreur sur la réception sous la forme du rapport entre l'erreur sur la réception de données et la durée de transmission de l'ensemble de données en question. Par exemple, pour les échanges du modèle transactionnel simplifié présentés dans la Figure 3.19, le taux d'erreur sur la réception de l'ensemble de données M_1 est égal à E_1 divisé par T_1.

Le taux d'erreur sur la réception est directement lié aux temps d'attente des nœuds pour l'accès au bus. On peut définir alors un taux de contention en se référant au modèle CA. Ce taux représente le rapport entre la somme des durées d'attente pour l'accès au bus et la somme des durées de transmission via le bus. Par exemple, pour le scénario de transmission présenté dans la Figure 3.19, le taux de contention est égal à la durée d'attente divisée par la durée totale de transmission Tx.

Plusieurs séries de simulation des trois modèles (modèle CA, modèle transactionnel simplifié et modèle transactionnel utilisant la technique de modélisation proposée) pour différentes valeurs du taux de contention sont effectuées afin de mesurer le taux d'erreur moyen sur la réception des deux modèles transactionnels (modèle transactionnel simplifié et modèle transactionnel utilisant la technique de modélisation proposée). Pour chaque simulation, nous avons défini un scénario de fonctionnement formé par des ensembles de données de tailles non identiques à transmettre par les maîtres 1 et 2 et de leurs instants de transmission associés. Pour chaque scénario, le nombre des ensembles de données à transmettre par chaque maître est 1000. Cette valeur a été choisie afin de garantir un nombre suffisant d'échantillons permettant d'estimer correctement les taux d'erreurs moyens. Chaque scénario est simulé par le modèle CA afin de mesurer le taux de contention qu'il représente et afin d'obtenir les instants de fin de transmission de chaque ensemble de données. Chaque scénario est ensuite appliqué pour la simulation des deux modèles transactionnels afin d'obtenir les instants de réception des transactions et de les comparer à ceux obtenus par le modèle CA. Le même principe de mesure est adopté dans [**54**]. La Figure 3.20 présente les taux d'erreurs obtenus par les deux modèles transactionnels en fonction du taux de contention.

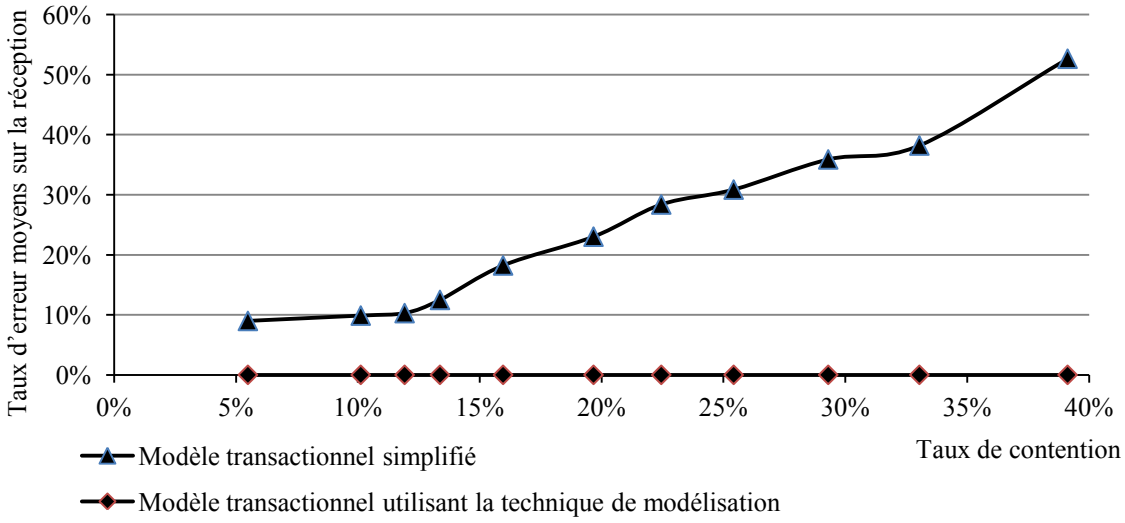


Figure 3.20 – Taux d'erreur moyens sur la réception obtenus par les deux modèles transactionnels.

La Figure 3.20 présente les taux d'erreurs moyen sur les résultats des instants de réception des ensembles de données obtenus par les modèles transactionnels par rapport aux résultats fournis par le modèle CA en fonction du taux de contention. On constate que les taux d'erreurs des résultats pour le modèle transactionnel simplifié augmentent avec l'augmentation du taux de contention. Le taux d'erreur pour le modèle transactionnel utilisant la technique de modélisation est nul pour toutes les simulations, montrant ainsi que le mécanisme de gestion des instants d'évolution est correctement mis en œuvre.

Cette illustration permet de justifier de l'intérêt de la technique proposée vis-à-vis du critère de précision des modèles de performances. Pour autant, les mesures de la précision des observations n'ont pas été effectuées. Ces mesures devraient être validées sur des cas pratiques. La sous-section suivante propose une expérimentation possible.

3.4 Illustration de l'application de l'approche de modélisation et de la technique proposée

Dans cette partie nous considérons la modélisation d'une architecture représentative d'un système à base de microprocesseur afin d'illustrer la mise en œuvre de l'approche considérée et l'application de la technique de modélisation proposée. Cette partie illustre également les observations possibles pour un scénario de fonctionnement donné.

Le système considéré est mis en œuvre sur une carte d'évaluation basée sur un circuit FPGA qui implémente deux processeurs de type *MicroBlaze.* Une IP FFT (*Fast Fourier Transform*) permettant le traitement d'une transformée de Fourier rapide sur 8 points est également implémentée au sein du FPGA. Ces trois éléments communiquent via un bus de type PLB (*Processor Local Bus*). Ce bus est souvent employé pour assurer la communication entre les processeurs et les éléments matériels au sein des microcontrôleurs. La Figure 3.21 présente l'organisation des ressources matérielles de l'architecture considérée.

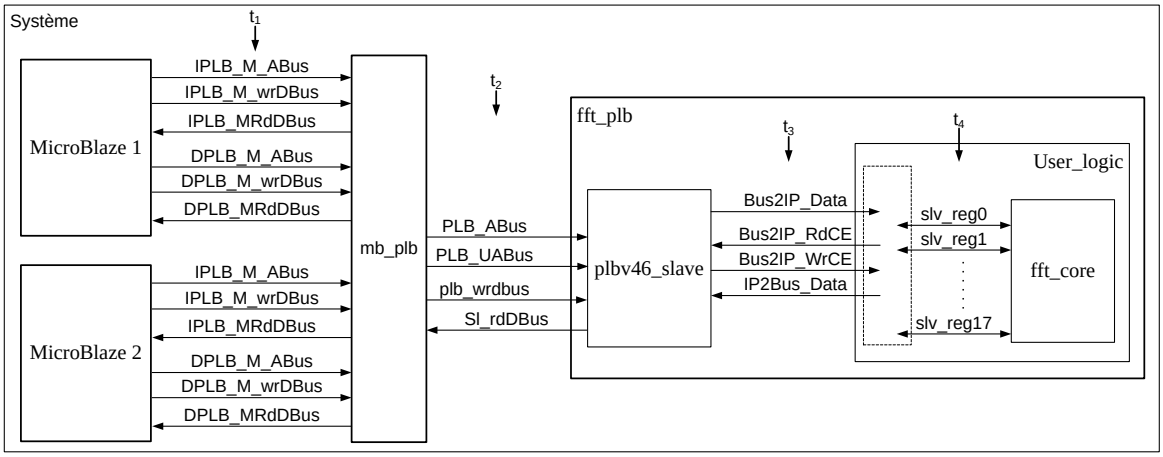


Figure 3.21 – Organisation des ressources matérielles de l'architecture considérée.

Les deux processeurs *MicroBlaze*, le bus PLB et l'IP FFT sont cadencés à 100 Mhz. Le *MicroBlaze* 1 est plus prioritaire que le *MicroBlaze* 2. La largeur du bus de données du bus PLB est de 32 bits. Les symboles complexes sont définis sur 32 bits dont 16 bits sont utilisés pour définir la partie réelle et les autres 16 bits sont utilisés pour définir la partie imaginaire. L'IP FFT permet de traiter 8 symboles complexes et possède 8 registres pour stocker ces symboles. Le résultat de ce traitement correspond à 8 autres symboles complexes qui sont stockés dans 8 autres registres. L'IP FFT possède 2 autres registres permettant de stocker la configuration et l'état de l'IP. Afin de calculer la transformée de Fourier de 8 points, l'un des processeurs transmet un à un via le bus PLB les 8 symboles complexes à l'interface de l'IP FFT. Ensuite, il transmet une donnée destinée au registre de configuration afin de déclencher le traitement de l'IP FFT et une deuxième pour réinitialiser ce registre. Une fois le traitement fini, l'IP indique une valeur particulière dans le registre d'état. Le processeur scrute le contenu de ce registre afin de détecter l'instant où les résultats peuvent être transférés. La transmission des résultats se fait à la demande. En effet,

la valeur d'un symbole complexe est envoyée quand le processeur demande la lecture du registre qui la stocke. Ce système est modélisé par trois activités tel que présenté dans la Figure 3.22. Cette modélisation ne considère pas la réception des résultats.

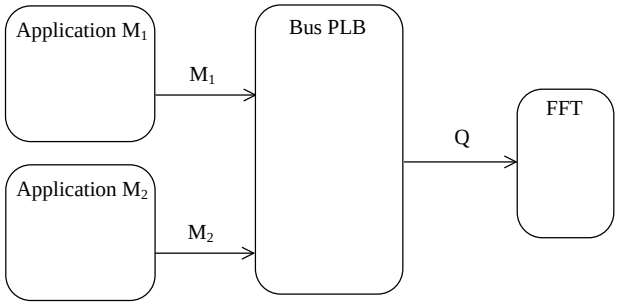


Figure 3.22 – Modèle d'architecture du système considéré.

Les activités « Application M_1 » et « Application M_2 » modélisent respectivement le comportement des algorithmes exécutés par les processeurs MicroBlaze 1 et MicroBlaze 2 et de leurs utilisations respectives. L'activité « Bus PLB » modélise le comportement des interfaces PLB du processeur et de l'IP FFT et du bus. L'activité « FFT » modélise le cœur de l'IP FFT qui effectue le calcul et exprime également l'utilisation des ressources de mémorisation nécessaires au stockage des données traitées. M_1, M_2 et Q représentent les relations qui permettent l'échange de transactions.

Dans un premier temps, nous expliquons la modélisation de cette architecture selon l'approche de modélisation considérée. Dans cette modélisation les transactions véhiculées par les trois relations représentent des données de 32 bits qui peuvent correspondre à un symbole complexe ou à une donnée de configuration. « Application M_1 » et « Application M_2 » génèrent cycliquement des ensembles formés de 8 symboles complexes. Ces symboles sont transmis séquentiellement un à un toutes les périodes T_{sym}. A la fin de cette transmission, deux transactions représentant les données de configuration sont envoyées. « Bus PLB » reçoit ces transactions et les produit avec un retard égal à T_l. Ce retard correspond à la latence induite par le bus PLB. « FFT » est en attente des 8 symboles complexes et de la donnée de configuration pour déclencher le traitement. La Figure 3.23 montre les différentes transactions échangées ainsi que l'utilisation induite des registres de l'IP FFT.

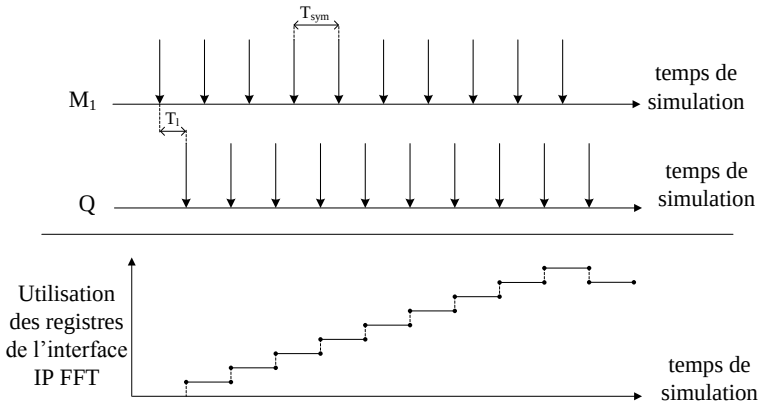


Figure 3.23 – Observations fournis par le modèle transactionnel.

Les observations présentées dans la Figure 3.23 correspondent à l'observation de l'envoie d'une séquence formée de 8 symboles complexes et de 2 données de configuration par le MicroBlaze 1. La partie haute montre les 10 transactions échangées via les relations M_1 et Q. La partie basse illustre l'utilisation des registres en fonction des transactions reçues.

Dans un deuxième temps, nous expliquons la modélisation de l'architecture considérée selon la technique de modélisation proposée. Selon cette modélisation, les transactions véhiculées par les trois relations représentent l'ensemble formé par 8 symboles complexes et deux données de configuration. « Application M_1 » et « Application M_2 » génèrent cycliquement ces ensembles formés de 8 symboles complexes. La transmission de ces symboles ainsi que des deux données de configuration est représentée par l'envoie d'une transaction. « Bus PLB » reçoit cette transaction. L'action *ComputeDelay()* a pour rôle de calculer l'instant de production T_s de cette transaction. Cet instant correspond à l'instant de réception de la première donnée de configuration. L'expression analytique permettant à *ComputeDelay()* de calculer cet instant est donnée par :

$$T_s = CurrentTime() + T_l + 9 \times T_{sym}.$$

CurrentTime() permet de récupérer le temps courant de la simulation. Cette équation est valable pour les transactions envoyées par « Application M_1 ». Pour celles envoyées par « Application M_2 » cette équation devient :

$$T_s = NextTs() + T_l + 9 \times T_{sym}.$$

Dans le cas où il y a une transaction envoyée par « Application M_1 », *NextTs()* donne l'instant T_s calculé de production de cette transaction. Dans le cas contraire, *NextTs()* donne le temps courant de la simulation. La primitive *ComputeObservation()* calcule ensuite les couples $[c_s^{(i)}, T_o^{(i)}]$. Dans le cas d'une préemption, les symboles complexes déjà transmis à l'IP FFT seront simplement effacés. La mise à jour des observations consiste donc à supprimer les couples $[c_s^{(i)}, T_o^{(i)}]$ calculés qui ont des instants ultérieurs à l'instant courant. L'algorithme de calcul de *ComputeObservation()* des couples $[c_s^{(i)}, T_o^{(i)}]$ est le suivant:

Calcul du début de l'observation :

configuration

Si « Application M_1 » envoie une transaction

alors $t_o = CurrentTime() + T_l$

Si « Application M_1 » envoie une transaction

alors $t_o = NextTs() + T_l$

Incrémentation du nombre de registres utilisés pour le stockage de 8 symboles complexes et d'une donnée de configuration :

Pour i de 1 à 9 {

$[c_s^{(i)}, T_o^{(i)}] = [i , t_o]$

$t_o = t_o + T_{sym}$ }

Décrémentation suite à la réinitialisation de la donnée de :

$[c_s^{(10)}, T_o^{(10)}] = [9, t_o + T_{sym}]$

Cette description permet d'exprimer l'utilisation des registres au sein de l'IP FFT. Cette description est exécutée en temps nul vis-à-vis du simulateur. La Figure 3.24 présente les observations obtenues par cette modélisation.

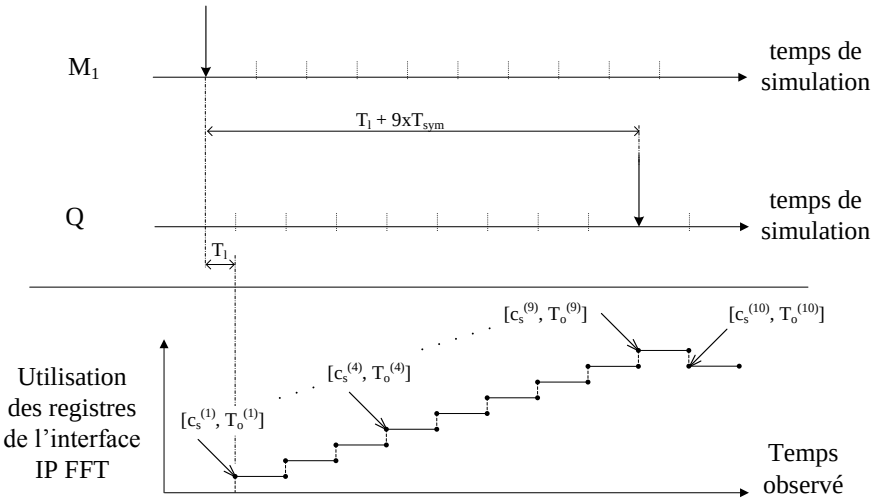


Figure 3.24 – Observations fournis par le modèle après application de la technique de simulation.

Les observations présentées dans la Figure 3.24 correspondent à la simulation de l'envoie d'une transaction qui représente 8 symboles complexes et 2 données de configurations par le MicroBlaze 1. La partie haute montre les transactions échangées en fonction du temps de simulation. La partie basse montre l'utilisation des registres en fonction en fonction du temps observé.

Cette description ouvre la voie à l'expérimentation sur une étude de cas de complexité restreinte de la technique de modélisation proposée.

3.5 Bilan et synthèse des contributions

Comme évoqué précédemment au chapitre 2, la création de modèle simulables d'architectures s'appuie sur la notion de modélisation transactionnelle et est actuellement possible à l'aide de langages tels que SystemC. La contribution présentée au sein de ce chapitre porte sur la définition d'une technique de modélisation visant à améliorer le compromis entre la rapidité de simulation des modèles et la précision des estimations offertes. Cette technique propose de réduire le nombre de transactions nécessaires au sein des modèles tout en permettant de conserver une bonne précision des estimations portant sur l'utilisation des ressources. Pour ce faire, nous proposons d'utiliser un calcul local des propriétés dont l'évolution peut être effectuée en temps nul, sans intervention du simulateur. La réduction obtenue du nombre de transactions conduit à accroitre la vitesse de simulation des modèles. Dans le cas où l'application de la technique proposée conduirait à un accroissement de la taille des données véhiculées au sein des transactions, il a été observé qu'une accélération significative de la simulation pouvait également être obtenue.

L'évolution des propriétés associées aux transactions économisées est obtenue par l'utilisation d'estimations portant sur l'évolution supposée des propriétés. Cette approche est donc intrinsèquement limitée par le niveau d'estimation qui peut être établi afin d'obtenir

une évolution suffisamment fiable des propriétés, et ce sans utilisation de transactions. Un compromis est à trouver entre ce qui peut être exprimé de manière déterministe (et donc calculé en temps nul) et ce qui présente un caractère non déterministe, sporadique, et qui doit alors être exhibé au sein du modèle. Par conséquent, l'abstraction opérée par application de la technique proposée peut conduire à réduire le niveau de précision de prédictions des observations des paramètres recherchés. L'adoption d'une telle approche pose donc la question de la précision des estimations effectuées afin de permettre de réduire le nombre de transaction nécessaires.

La proposition faite dans le cadre de ce travail reprend une idée précédemment introduite dans le cadre de la thèse d'Anthony Barreteau [**60**]. Cette idée portait déjà sur la possible réduction du nombre de transactions au sein des modèles d'architectures. Pour autant son application était restée limitée au cas simple de la modélisation de ressources de calcul sans prise en compte de la préemption des ressources. Dans le cadre de ce travail nous avons donc étendu cette idée par la définition de méthodes spécifiques permettant une gestion correcte des prédictions et permettant ainsi d'étendre l'application possible de cette technique de modélisation. Nous avons également cherché à fournir une qualification plus précise de cette technique en mesurant les facteurs d'accélération possibles ainsi que l'influence possible sur la précision des estimations. Enfin, nous avons cherché à illustrer l'application de la technique à la modélisation de ressources de communication.

Afin d'illustrer l'apport potentiel de notre contribution au cas d'architectures électroniques du domaine automobile, le chapitre suivant présente une étude de cas spécifique.

Chapitre 4
Application de la technique proposée pour le dimensionnement d'une architecture distribuée inspirée du domaine automobile

Sommaire

Ce chapitre vise, à travers une étude de cas, à expliquer l'application de la technique proposée pour la modélisation d'une architecture distribuée inspirée du domaine automobile. Cette modélisation est utilisée en vue du dimensionnement des ressources de mémorisation de l'architecture. Il sera ainsi possible sur une architecture de complexité significative de qualifier l'apport de la technique de modélisation proposée.

La première section présente l'étude de cas considérée ainsi que l'architecture et le protocole de communication retenu. La deuxième section présente la modélisation transactionnelle de cette architecture ainsi que les résultats obtenus par simulation. La troisième section explique l'application de la technique de modélisation proposée sur le cas d'étude. Elle présente également différents résultats obtenus et un bilan.

4.1 Présentation de l'étude de cas

4.1.1 Conception d'une architecture multi calculateur supportant de nouvelles technologies de communication

Actuellement, la complexité des architectures électroniques des véhicules se traduit par une diversité des protocoles supportés et une forte densité des supports de transmission. Les tendances actuelles visent à définir des protocoles de transmission à des débits plus élevés tout en limitant l'impact sur l'infrastructure existante des véhicules. Dans ce contexte, le projet CIFAER abordait l'exploration de nouvelles technologies de communication pour le domaine automobile [**3**]. Ce projet visait alors à étudier la faisabilité de la mise en œuvre des technologies à base de courant porteur en ligne (CPL) et sans fil au sein des véhicules.

Les architectures embarquées dotées de ces nouveaux moyens de communication permettaient de répondre aux exigences évoquées précédemment. Les supports de transmission CPL et sans fil ne nécessitent pas d'installation de câbles supplémentaires, permettant ainsi la réduction des coûts d'infrastructure et de maintenance. Dans le cadre du projet CIFAER, afin de répondre aux besoins croissants des applications émergentes de multimédia embarqué, les protocoles de communication à haut débit HomePlugAv (HPAV) et WiFi ont été respectivement choisis pour les supports de transmissions CPL et sans fil. Ainsi, le projet était organisé en trois activités complémentaires :

- Dimensionnement du réseau de calculateurs, intégrant une gestion optimisée des ressources de calcul et de communication ;
- Etude des performances de transmission des supports CPL et sans fil au sein de véhicules [**61**];
- Réalisation du démonstrateur.

Pour valider l'approche du projet CIFAER, un démonstrateur mettant en évidence l'objectif du projet a été réalisé. Ce démonstrateur correspond à l'ensemble formé par une application multimédia embarquée s'exécutant sur trois calculateurs distribués. Ces calculateurs doivent communiquer selon les protocoles de communication sans fil en utilisant le standard WiFi et à base de courant porteur en utilisant le standard HPAV. L'architecture de ce système est détaillée sur la Figure 4.1.

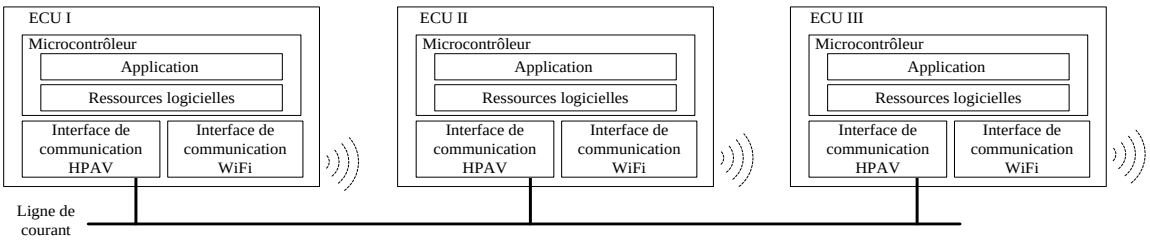


Figure 4.1 – Architecture du démonstrateur du projet CIFAER.

Les ressources matérielles et logicielles des ECU ainsi que l'application sont organisées tel que présenté sur la Figure 4.1. Chaque ECU met en œuvre différentes ressources matérielles et logicielles. Le microcontrôleur exécute les fonctions de l'application et

différentes ressources logicielles telles que les couches protocolaires RTP, UDP et IP. Les protocoles RTP et UDP sont les mieux adaptés pour les applications multimédia car ils préservent la bande passante en éliminant les mécanismes d'acquittement des données reçues et de retransmission des données erronées. En effet, il n'est pas nécessaire de retransmettre des données pour compléter une image déjà affichée. Chaque ECU met également en œuvre différentes ressources de communication inter-ECUs selon les standards HPAV et WiFi. Ces ressources se présentent sous la forme de modems spécifiques. L'application réalisée par le démonstrateur consiste en la diffusion de vidéos entre différents emplacements au sein du véhicule. Un des ECUs est branché à une caméra, acquiert la vidéo et la transmet aux deux autres ECUs. La scène capturée par la caméra doit être affichée sur l'écran de chaque calculateur. La vidéo est cadencée à 25 images par seconde. Chaque image a une résolution de 120x160 pixels. Chaque pixel est codé sur 3 octets. La transmission des flux vidéo est effectuée à travers le CPL et/ou sans fil selon différents modes de fonctionnement.

Dans cette étude de cas, nous nous intéressons particulièrement à l'étude du support de communication CPL. Plus précisément, nous cherchons à dimensionner les ressources nécessaires à la mise en œuvre du protocole HPAV compte tenu des contraintes de temps considérées.

4.1.2 Description du protocole HPAV

Le protocole HomePlugAV [**62**] représente la nouvelle génération de communication utilisant les courants porteurs en ligne. Ce protocole assure le transfert des données encapsulées dans des trames IP et Ethernet. Selon [**63**], le protocole HPAV permet d'assurer une transmission à un débit de 150 Mbit/s de données utiles. L'organisation des différents dispositifs supportant le protocole HPAV est représenté sur la Figure 4.2 extraite de [**63**].

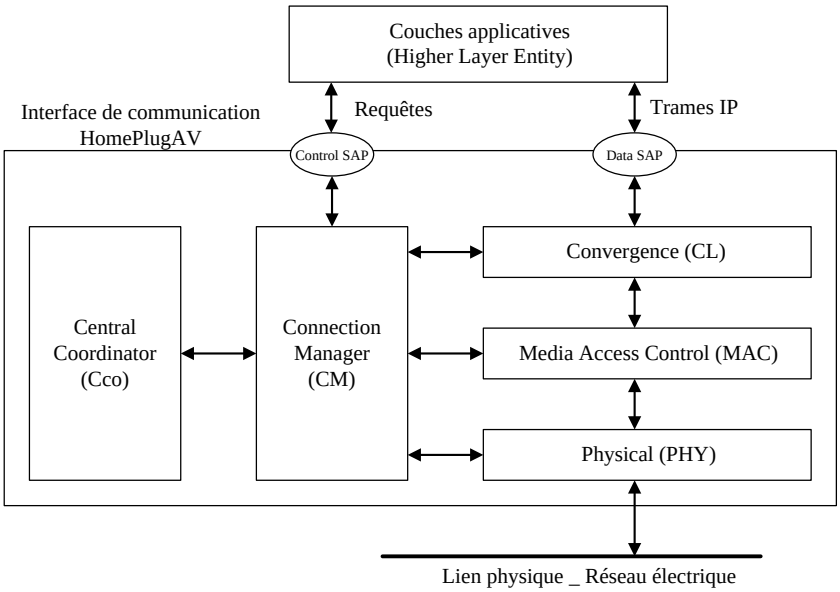


Figure 4.2 – Organisation des dispositifs supportant le protocole HPAV.

L'interface de communication HPAV s'intercale entre les couches applicatives et le lien physique. La description de l'organisation du protocole sépare le flux des informations utiles

des informations de contrôle nécessaires au fonctionnement. Le flux des informations utiles est véhiculé via le canal de transmission après avoir traversé les couches MAC et physique. La couche dite de convergence supporte un ensemble de services garantissant le lien entre les couches applicatives et MAC. Cette couche permet d'établir automatiquement des connexions. Elle classifie les paquets selon la connexion à laquelle ils appartiennent et peut, si demandé, leur associer une information temporelle ATS (*Arrival Time Stamp*). Cette couche peut également servir de buffer à l'émission comme à la réception pour moyenner les temps de latence. Le flux de contrôle représente les requêtes en provenance des couches de plus haut niveau. Ces requêtes portent notamment sur la définition du niveau de qualité de service attendu pour la transmission et sur la détection de nouveaux équipements au sein du réseau électrique. La couche MAC traite les paquets de données en provenance de la couche de convergence, typiquement selon le format des trames Ethernet. Les étapes de ce traitement sont présentées sur la Figure 4.3 extraite de [**63**].

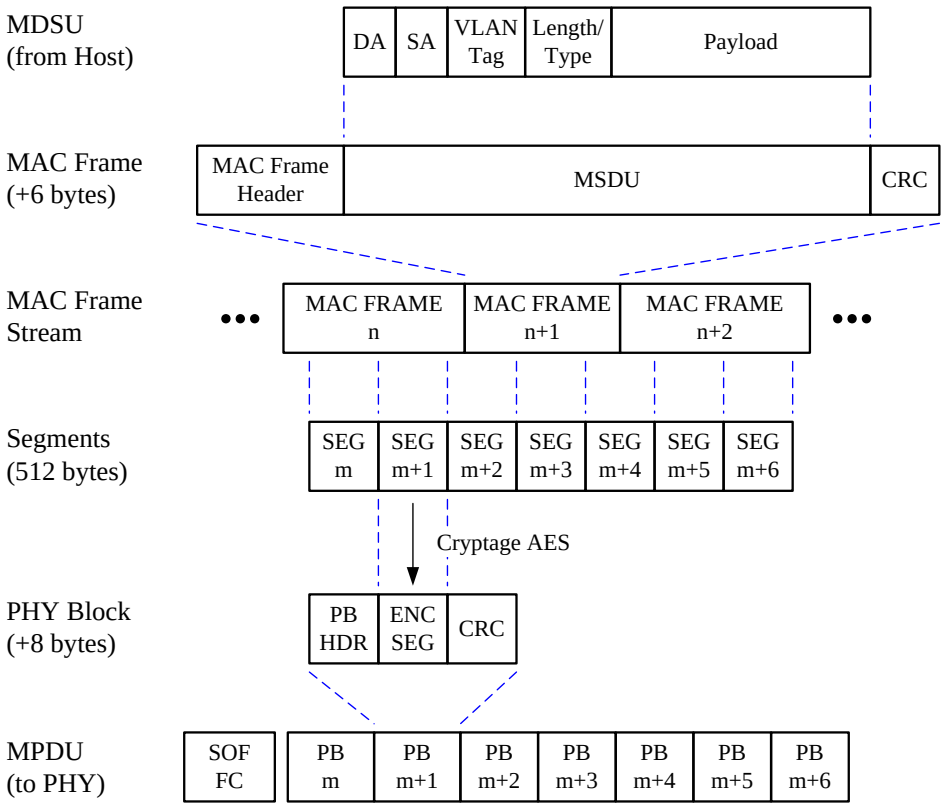


Figure 4.3 – Segmentation des données manipulées par la couche MAC.

La couche MAC encapsule les paquets de données, appelés MSDU (*MAC Service Data Unit*), en y associant un entête et un champ de contrôle d'erreur. Ces trames sont ensuite stockées dans différentes files d'attente. Ces files sont identifiées selon la connexion qu'elles représentent. La concaténation des trames dans chaque file d'attente est appelée *MAC Frame Stream*. En deuxième phase, la couche MAC segmente les données présentes dans le *MAC Frame Stream* en des segments de taille 512 octets. Les segments peuvent être cryptés selon l'algorithme de cryptage AES (*Advanced Encryption Standard*). Les segments sont ensuite encapsulés en y associant un entête et un champ de contrôle d'erreur pour former un bloc physique PB (*Physical Block*). Ces PB sont stockés dans une file d'attente

appelée MPDU (*MAC Protocol Data Unit*). Ces données sont ensuite utilisées par la couche physique. Au passage à la couche physique, un champ de 16 octets indiquant le début d'une trame, appelé SOF (*Start Of Frame*), est rajouté à chaque MPDU. Ces trames sont périodiquement acquittées par le récepteur avec des SACK (*Selective Aknowledge*). Certaines trames peuvent donc être retransmises.

La couche MAC gère également les instants d'accès au canal physique. Le protocole HPAV supporte deux techniques d'accès en associant un accès du type TDMA et CSMA/CA (*Collision Sense Multiple Access/Collision Avoidance*). L'organisation de l'accès considérée pour le protocole HPAV est représentée sur la Figure 4.4.

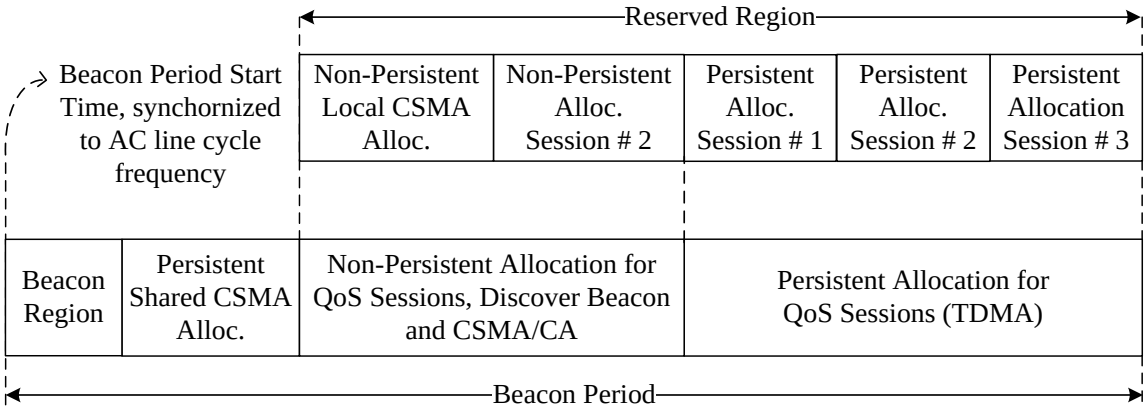


Figure 4.4 – Organisation du cycle de communication selon le protocole HPAV.

L'accès au support de communication selon le protocole HPAV est cyclique. Un cycle de communication est réparti en différentes régions. La région dite *Beacon Region* joue un rôle d'entête du cycle permettant de définir l'organisation temporelle du cycle pour les différents nœuds du réseau. Le nœud coordinateur (CCo) placé dans le réseau délivre à chaque nœud les informations sur la planification des allocations pour la région dite *Reserved Region*. La région dite *CSMA Region* est utilisée pour des transmissions de faible débit. Elle assure l'interopérabilité avec la première version du protocole HPAV. La région *Reserved Region* se trouve découpée en deux parties. La partie d'allocation dite non persistante est une zone au sein de laquelle des données peuvent être transmises par les nœuds selon le type d'accès CSMA/CA. La partie d'allocation dite persistante permet de garantir le transfert d'informations en satisfaisant les contraintes de qualité de service négociées par la partie contrôle du protocole. Cette partie permet un accès aux nœuds selon le type d'accès TDMA. Le découpage de cette partie et la durée de la partie d'allocation dite non persistante sont définis au cours de la région dite *Beacon Region*. Par la suite, compte tenu de l'application considérée, nous considérerons des transmissions selon le type d'accès TDMA uniquement.

Dans la suite, nous nous intéressons à la modélisation de l'architecture du démonstrateur du projet CIFAER en vue de dimensionner les ressources supportant le protocole HPAV.

4.2 Modélisation et simulation de l'architecture étudiée

4.2.1 Modélisation de l'architecture

Le système étudié correspond à un système de transmission vidéo entre trois éléments : un émetteur et deux récepteurs. Chaque élément utilise les protocoles RTP et IP pour le contrôle de la transmission des informations. Dans cette étude de cas, nous ne nous intéressons qu'à la modélisation des ressources de communication utilisant le protocole HPAV et nous supposons que les flux vidéo sont transmis uniquement via le support CPL. La délimitation retenue pour l'architecture du système est représentée sur la Figure 4.5.

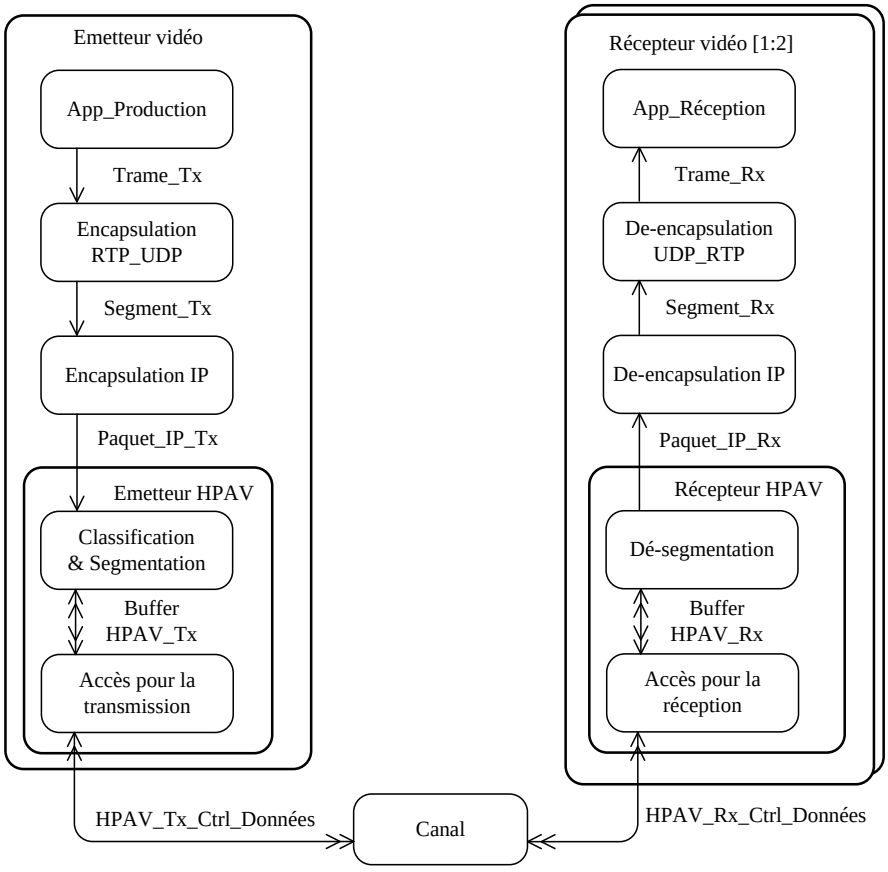


Figure 4.5 – Diagramme d'activité du système de communication étudié.

Les activités « Emetteur vidéo » et « Récepteur vidéo » décrivent l'architecture du système. L'activité « Emetteur vidéo » modélise l'architecture associée à l'ECU qui transmet la vidéo et se compose de quatre activités. « App_Production » modélise l'acquisition de la vidéo et génère périodiquement une transaction qui représente la production d'une image. « Encapsulation RTP_UDP » modélise l'encapsulation des données d'une image par les couches protocolaires RTP et UDP. « Encapsulation IP » modélise la fragmentation et l'encapsulation par le protocole IP des segments en provenance de couches RTP et UDP. « Emetteur HPAV » modélise l'interface de communication HPAV utilisée principalement pour la transmission de données. Cette activité se compose de deux activités. « Classification & Segmentation » modélise la classification de données de la couche Convergence et la segmentation de données de la couche MAC du protocole HPAV.

La relation « Buffer HPAV_Tx » sert au stockage des PBs en attendant leurs transmissions et la réception de leurs acquittements. « Accès pour la transmission » modélise la gestion assurée par la couche MAC pour l'accès au lien physique ainsi que la couche physique du protocole HPAV. Elle modélise également la réception des acquittements et la notification pour la retransmission de PBs ou la libération de la mémoire du buffer.

L'activité « Récepteur vidéo » modélise les architectures associées à chaque ECU recevant la vidéo et se compose de quatre activités. « Récepteur HPAV » modélise l'interface de communication HPAV utilisée principalement pour la réception de données. Cette activité se compose de deux activités. « Accès pour la réception » modélise la gestion de la couche MAC pour l'accès au lien physique et modélise la réception de PB via la couche physique. Elle modélise également la transmission des acquittements. La relation « Buffer HPAV_Rx » sert au stockage des PBs. Un paquet IP est formé à partir de la réception d'un ensemble de PBs. « Dé-segmentation » modélise la dé-segmentation de données des PBs et la reconstruction des paquets IP. Elle modélise également la libération du buffer. « Dé-encapsulation IP » modélise la dé-encapsulation et la défragmentation par le protocole IP des paquets IP et la régénération des segments. « Dé-encapsulation UDP_RTP » modélise la dé-encapsulation par les couches protocolaires UDP et RTP des segments en provenance de la couche IP. « App_Réception » modélise l'affichage des images de la vidéo. L'activité « Canal » modélise l'effet du canal sur la transmission des PBs. Cet effet se traduit par l'introduction d'erreurs sur les PBs. Dans ce cas, le récepteur peut ne pas acquitter certains PBs, qui seront alors retransmis.

Dans le cadre de notre étude, nous cherchons à étudier l'influence des paramètres de l'architecture sur les ressources de mémorisation nécessaires à l'émission et à la réception. Nous détaillons plus particulièrement la modélisation faite des interfaces Emetteur HPAV et Récepteur HPAV dans la mesure où elles influent directement sur les performances de l'architecture.

4.2.2 Modélisation de l'interface d'émission HPAV

Nous détaillons ici la modélisation faite de l'émetteur HPAV. Une description détaillée de l'ensemble de l'architecture est donnée dans [**64**]. Dans cette partie, on se focalise sur les détails de modélisation de cette interface car elle servira pour l'explication de l'application de la technique proposée. En effet, cette technique sera essentiellement appliquée pour la modélisation des interfaces HPAV émettrice et réceptrice.

La Figure 4.6 explique le comportement de l'activité « Classification & Segmentation » de l'émetteur HPAV.

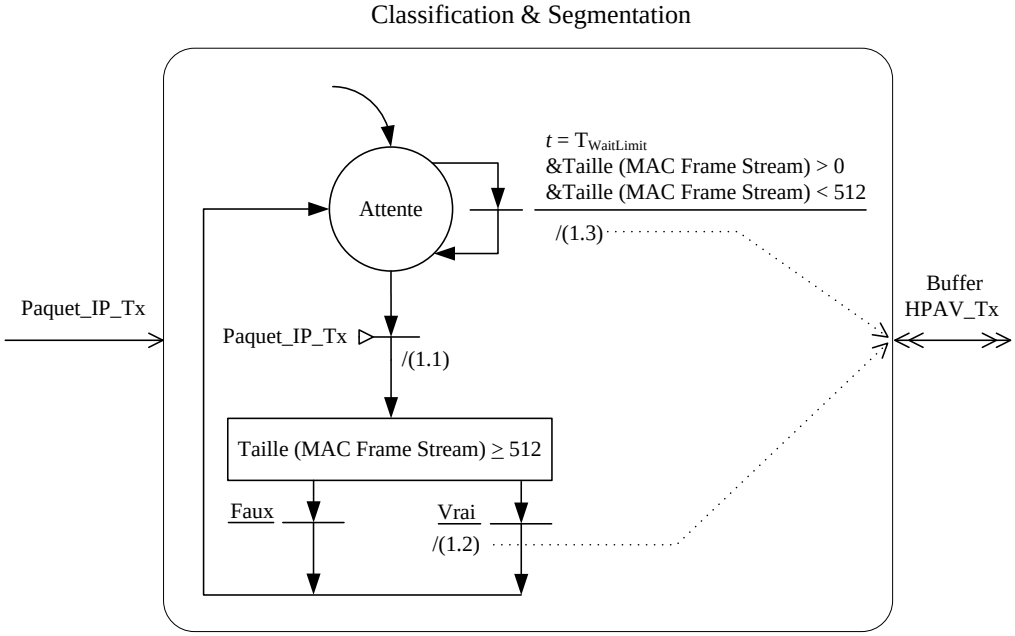


Figure 4.6 –Modèle de l'activité « Classification & Segmentation » de l'émetteur HPAV.

Cette activité peut être déclenchée par deux sources. La première source est liée à l'arrivée des transactions via la relation Paquet_IP_Tx. Les paquets IP reçus sont encapsulés et classés selon leur source et destination dans la file d'attente *MAC Frame Stream* correspondante. Ces actions correspondent à (1.1) sur la Figure 4.6. Si la taille de données dans une des files d'attente atteint un seuil fixe les actions notées par (1.2) sont effectuées. Par la suite, nous faisons l'hypothèse d'un seuil fixé à 512 octets. Il s'agit de fragmenter les données de la file d'attente concernée. Ces fragments sont ensuite cryptés et encapsulés afin de former un PB de taille 520 octets. Ces PBs sont stockés dans la relation Buffer HPAV_Tx afin d'être transmis. Les données converties sont supprimées de la file concernée. On suppose que le reste des données présentes dans les files d'attente et qui n'ont pas atteint le seuil de 512 octets devra être transmis avant la fin de l'intervalle TDMA en cours. $T_{WaitLimit}$ désigne un instant spécifique. Cet instant est calculé de façon à ce que intervalle de temps défini entre $T_{WaitLimit}$ et la fin de l'intervalle TDMA de l'émetteur en cours permette la transmission d'au moins un SOF et un PB. $T_{WaitLimit}$ est défini dans les actions (1.1). Quand $T_{WaitLimit}$ est atteint, les données restantes n'ayant pas atteint le seuil de 512 octets sont bourrées par des 0 dans le but de créer un PB. Ce PB est stocké dans la relation Buffer HPAV_Tx afin d'être transmis. Ces actions correspondent à (1.3) sur la Figure 4.6. A la fin de (1.3) les files d'attente sont vides. Les lignes en pointillé montrent les opérations d'écriture dans la relation Buffer HPAV_Tx.

L'activité « Accès pour la transmission » est une activité échantillonnée. Le comportement décrit la transmission des données à l'instant correct alloué à cet effet, et également la récupération des acquittements envoyés par les récepteurs pendant leurs intervalles TDMA. La Figure 4.7 montre le comportement de cette activité.

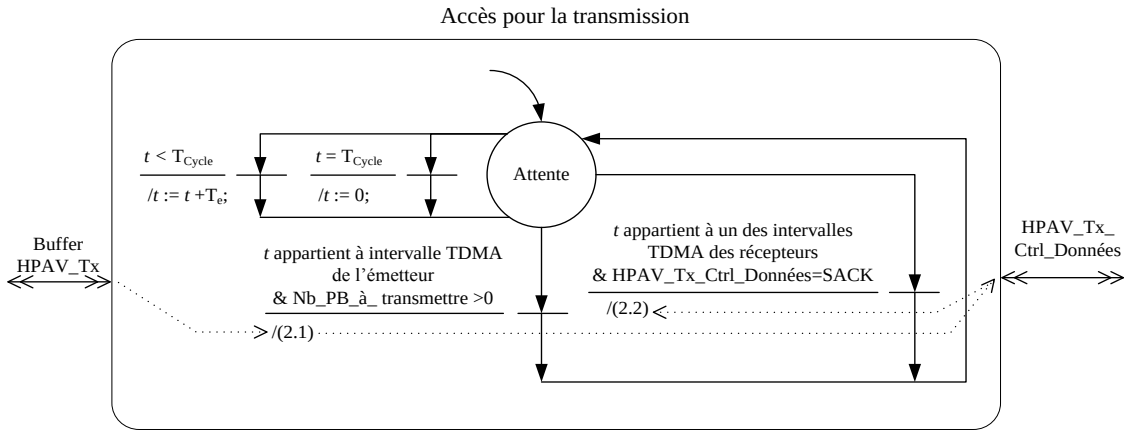


Figure 4.7 – Modèle de l'activité « Accès pour la transmission » de l'émetteur HPAV .

Le pas d'échantillonnage de cette activité est noté T_e dans la Figure 4.7. A chaque fois que cette activité est exécutée, les conditions temporelles sont évaluées. Si la variable, notée *t*, n'atteint pas la durée du cycle de communication, notée T_{Cycle}, alors elle est incrémentée de T_e. Sinon cette variable est mise à 0. A chaque pas d'échantillonnage cette variable est évaluée. Si *t* appartient à l'intervalle TDMA de l'émetteur et s'il existe des PBs à transmettre dans le Buffer HPAV_Tx alors les actions (2.1) sont exécutées. Il s'agit de transmettre les PBs stockés dans le Buffer HPAV_Tx et qui n'ont pas été transmis. Ainsi les PB transmis sont marqués comme transmis mais ne sont pas supprimés. Pendant la durée de la transmission du PB, il n'y a que l'incrémentation de la variable *t* qui est exécutée. Si *t* appartient à un des intervalles TDMA des récepteurs et si la relation HPAV_Tx_Ctrl_Données contient un acquittement SACK, alors les actions (2.2) sont exécutées. Il s'agit de supprimer de Buffer HPAV_Tx les PBs acquittés positivement. Il s'agit également d'identifier ceux qui sont acquittés négativement afin de les retransmettre.

4.2.3 Résultats de simulation

Nous présentons ici certains résultats obtenus par la simulation du modèle d'architecture. Certaines hypothèses ont été considérées lors de la modélisation et la simulation du système, notamment pour le protocole HPAV. Le Tableau 4.1 résume l'ensemble des paramètres utilisés pour la simulation du modèle d'architecture.

Paramètre	Valeur
Paramètres globaux	
Fréquence des images	15 images par seconde
Taille de l'image	160x120x3 (= 57600) Octets
Maximum Transmission Unit	1500 Octets
Débit des paquets IP	100 Mbit/s
Paramètres du HPAV	
Durée du cycle de communication	40 ms
Duré de transmission d'un PB	37 µs (512 Octets / 110 Mbit/s)
Seuil de pour *MAC Frame Stream*	512 Octets
Intervalle TDMA de l'émetteur	Début 25 ms Fin 35 ms
Intervalle TDMA du 1er récepteur	Début 35 ms Fin 37.5 ms
Intervalle TDMA de 2ème récepteur	Début 37.5 ms Fin 40 ms

Tableau 4.1 – Paramètres considérés pour la simulation du modèle HPAV

Les paramètres globaux du modèle sont représentés dans la partie haute du Tableau 4.1. Le cahier des charges du démonstrateur CIFAER précise que la vidéo présente un flux d'images cadencé à 15 images par seconde de résolution 160x120 pixels RGB chacune. Le paramètre MTU (*Maximum Transmission Unit*) définit la taille maximale d'un paquet IP. Le MTU est fixé à 1500 octets. Si la taille d'un paquet IP dépasse la valeur de ce paramètre, la couche IP émettrice fragmente ce paquet IP en plusieurs paquets IP et la couche IP réceptrice le reconstruit. On suppose que le débit de données à l'entrée et la sortie de la couche de convergence est de 100 Mbit/s. Les paramètres liés à HPAV sont représentés dans la partie basse du Tableau 4.1. On suppose que l'échange de données est effectué uniquement dans la région TDMA. La durée du cycle de communication est fixée à 40 ms. L'intervalle TDMA de l'émetteur commence 25 ms après le début du cycle et dure 10 ms. Les intervalles TDMA du premier et du deuxième récepteur sont consécutifs et durent 2,5 ms chacun. On suppose aussi que la durée de la transmission d'un PB est de 37 µs. Cette valeur est équivalente au temps nécessaire pour transmettre 512 octets par une liaison offrant un débit moyen de 110 Mbit/s. Le seuil de taille de donnée dans une des files d'attente *MAC Frame Stream* est fixé à 512 octets.

On supposera par la suite qu'un SOF est envoyé s'il y a eu une interruption de la transmission de série continue de PB. Une interruption peut être due à cause du changement du destinataire ou à cause d'un temps mort. Un temps mort est lié au fait que le « Buffer HPAV_TX » est vide ou bien pour l'attente du début de l'intervalle TDMA de l'émetteur. Les SOFs ont une durée de transmission fixe. Nous avons supposé qu'un acquittement peut acquitter au maximum 16 PBs.

Le modèle de l'architecture complète a été capturé au sein de l'environnement CoFluent Studio. A partir de cette capture, l'outil génère un code en SystemC. Pour ce modèle ce code contient aux alentours de 3000 lignes dont 48% ont été générées automatiquement par l'outil. Nous présentons ci-après certains résultats de simulation du modèle obtenu à l'aide de cet outil pour les paramètres précisés dans le Tableau 4.1.

Nous expliquons ici la signification des échanges observées entre les activités du modèle lors de la simulation. La Figure 4.8 montre les échanges des activités de l'émetteur vidéo pour la production d'une image.

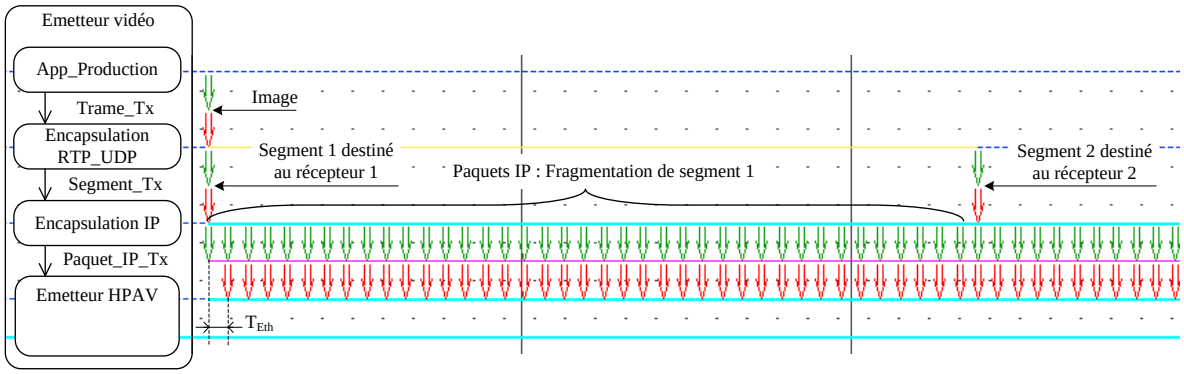


Figure 4.8 – Observation du processus de fragmentation IP.

« App_Production » produit une transaction exprimant une image véhiculé à travers la relation « Trame_Tx ». « Encapsulation RTP_UDP » envoie alors deux transactions via la relation « Segment_Tx ». Ces deux transactions expriment les segments UDP destinés au premier et au deuxième récepteur. A la réception d'un segment, « Encapsulation IP » produit plusieurs transactions véhiculées à travers la relation « Paquet_IP_Tx ». Ces transactions représentent les fragments IP d'une taille ne dépassant pas le MTU. Les fragments sont générés à partir du segment reçu. L'espacement entre paquets IP est noté T_{ETH}. Il correspond à la durée de transmission d'un paquet IP de taille 1500 octets à un débit de 100 Mbit/s.

La Figure 4.9 montre les échanges effectués entre les interfaces HPAV de l'émetteur et du premier récepteur.

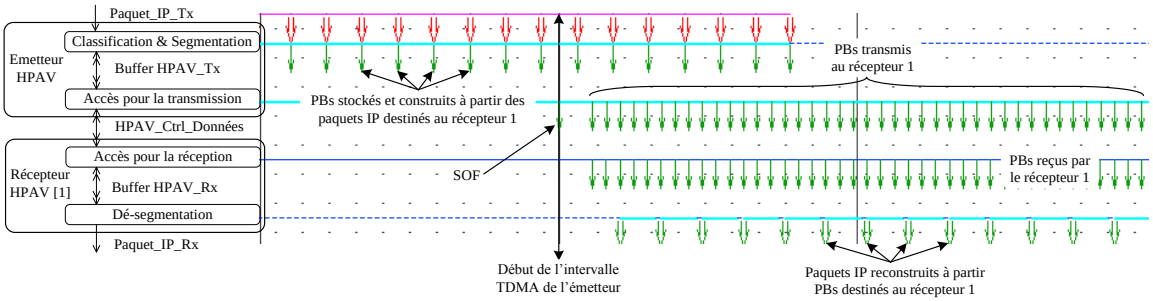


Figure 4.9 – Observation du processus de d'échange entre les interfaces HPAV.

« Classification & Segmentation » reçoit les paquets IP via « Paquet_IP_Tx ». Elle produit alors plusieurs transactions exprimant la conversion en PB de ces paquets. Les PBs ont la même destination que les paquets IP et sont stockés dans la relation « Buffer HPAV_Tx ». « Accès pour la transmission » envoie des transactions via « HPAV_Ctrl_Données » à partir du début de l'intervalle TDMA de l'émetteur. Ces transactions expriment des PBs destinés au premier récepteur. Elles sont précédées par l'envoie d'une transaction qui exprime le SOF. « Accès pour la réception » du premier récepteur récupère les PBs qui lui sont destinés et les stocke dans la relation « Buffer

HPAV_Rx ». Une fois un ensemble de PBs permettant de reconstruire un paquet IP reçu, « Dé-segmentation » envoie une transaction exprimant un paquet IP reconstruit et destiné au premier récepteur. Cette transaction est envoyée via la relation « Paquet_IP_Rx ».

La Figure 4.10Figure 4.21 montre les échanges effectués entre les activités du premier récepteur.

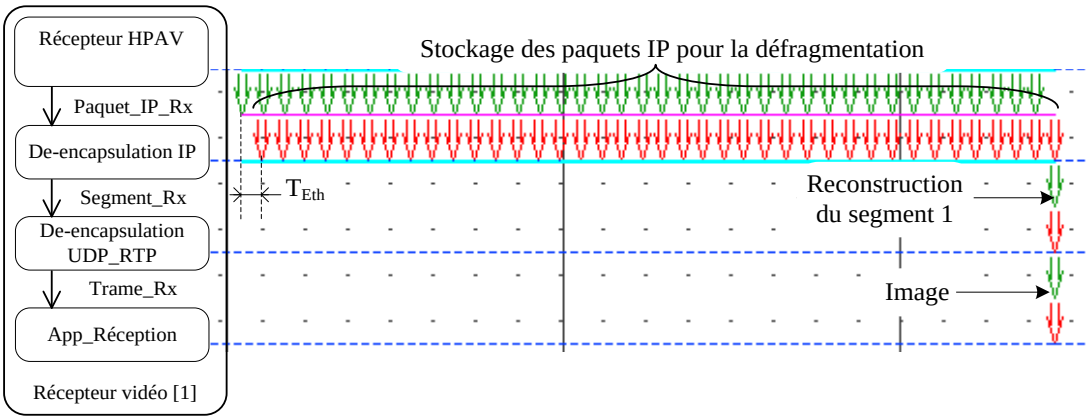


Figure 4.10 – Observation du processus de reconstruction d'image.

« De-encapsulation IP » stocke les paquets IP reçus. Une fois un ensemble de paquets IP permettant la reconstruction d'un segment récupéré, l'activité envoie une transaction en exprimant le segment reconstruit via la relation « Segment_Rx ». « De-encapsulation UDP_RTP » régénère l'image à partir du segment. Elle envoi alors une transaction à « App_Réception » via la relation « Trame_Rx ».

La Figure 4.11 propose une observation sur un intervalle de temps plus large.

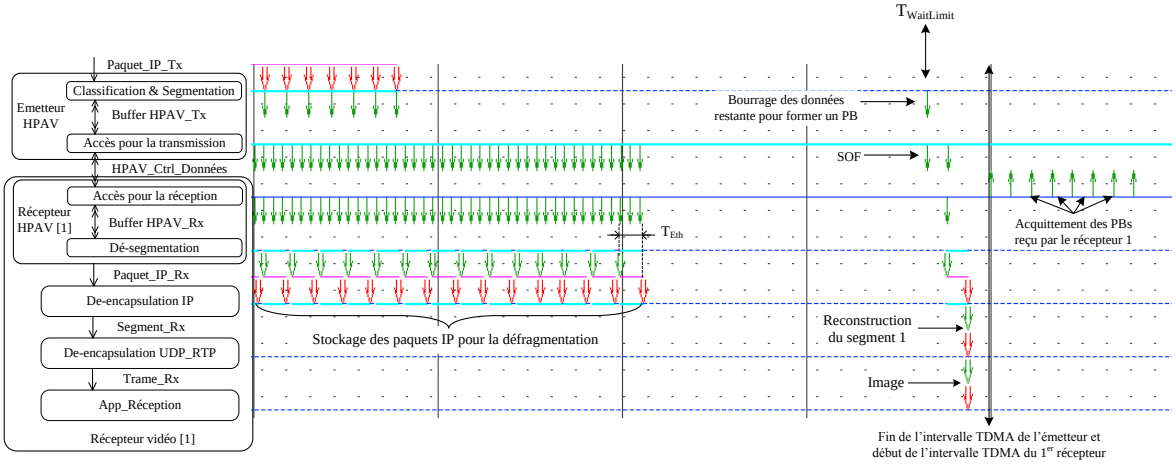


Figure 4.11 – Observation du processus de bourrage en fin de l'intervalle TDMA.

« Accès pour la transmission » envoie des transactions jusqu'à la transmission de tous les PBs stockés dans la relation « Buffer HPAV_Tx ». A « $T_{WaitLimit}$ », « Classification & Segmentation » procède au bourrage des données restantes afin de former un PB. Une transaction est ainsi produite, exprimant le stockage de ce PB dans « Buffer HPAV_Tx ». « Accès pour la transmission » envoie alors une transaction exprimant le SOF suivi d'une autre exprimant le dernier PB. A partir du début de l'intervalle TDMA du premier récepteur,

« Accès pour la réception » du premier récepteur envoi des transactions via « HPAV_Ctrl_Données ». Ces transactions expriment l'acquittement des PBs reçus.

La Figure 4.12 montre l'évolution de l'architecture pour le scénario où deux cycles sont nécessaires pour la transmission d'une image.

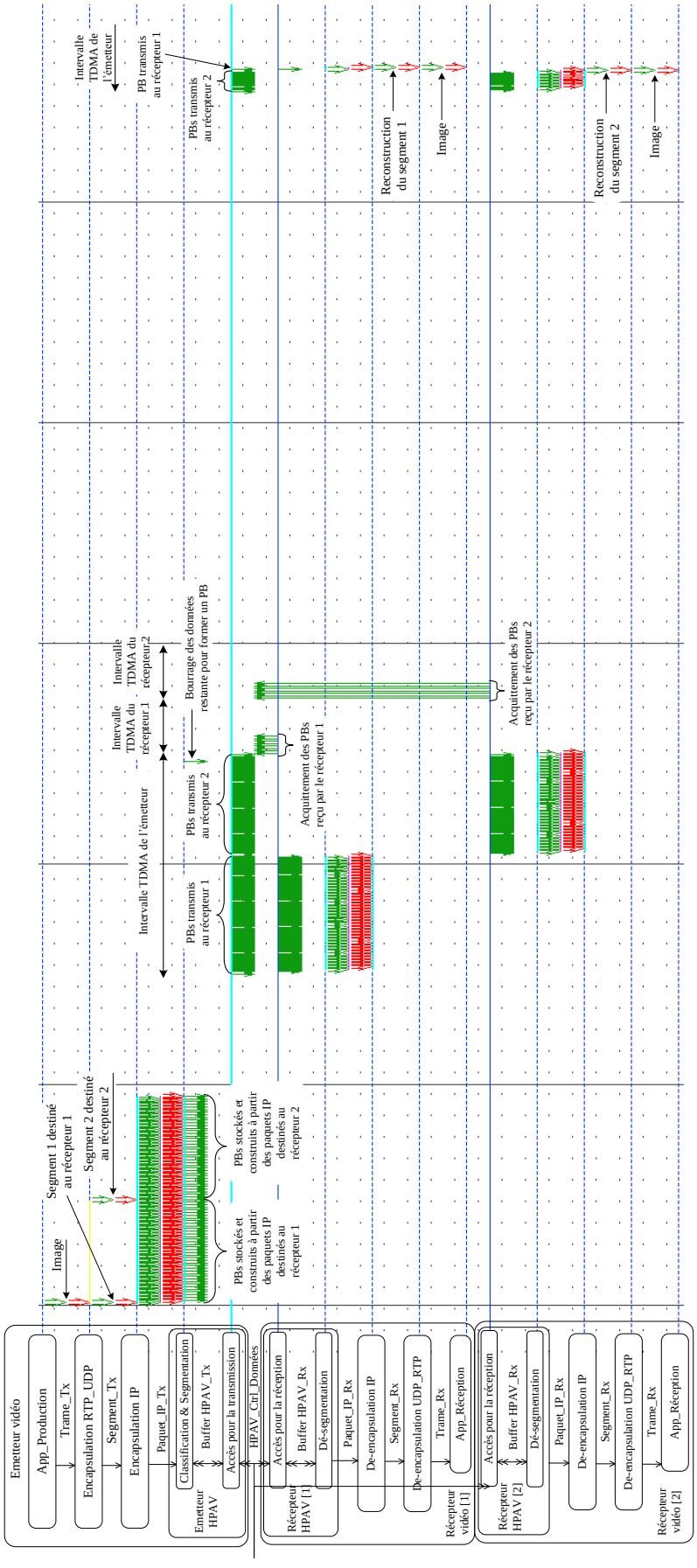


Figure 4.12 – Observation de la transmission d'une image sur deux cycles.

Au cours de l'intervalle TDMA de l'émetteur, « Accès pour la transmission » envoie des transactions jusqu'à la transmission de tous les PBs destinés au premier récepteur. Ensuite, elle envoie des transactions représentant des PBs destinés au deuxième récepteur jusqu'à la fin de l'intervalle TDMA de l'émetteur. La transmission est alors reprise à l'intervalle TDMA suivant de l'émetteur. Les données restantes destinées au deux récepteurs sont bourrées, transformées en PBs et stockées dans la relation « Buffer HPAV_Tx » avant la fin du premier intervalle TDMA de l'émetteur. Au cours du deuxième intervalle TDMA de l'émetteur, toutes les données des segments destinés au deux récepteurs sont transmises. Dès lors, les deux récepteurs sont capables de régénérer l'image transmise.

La Figure 4.13 présente l'utilisation de la mémoire de l'interface HPAV de l'émetteur pour la transmission d'une image vers les deux récepteurs.

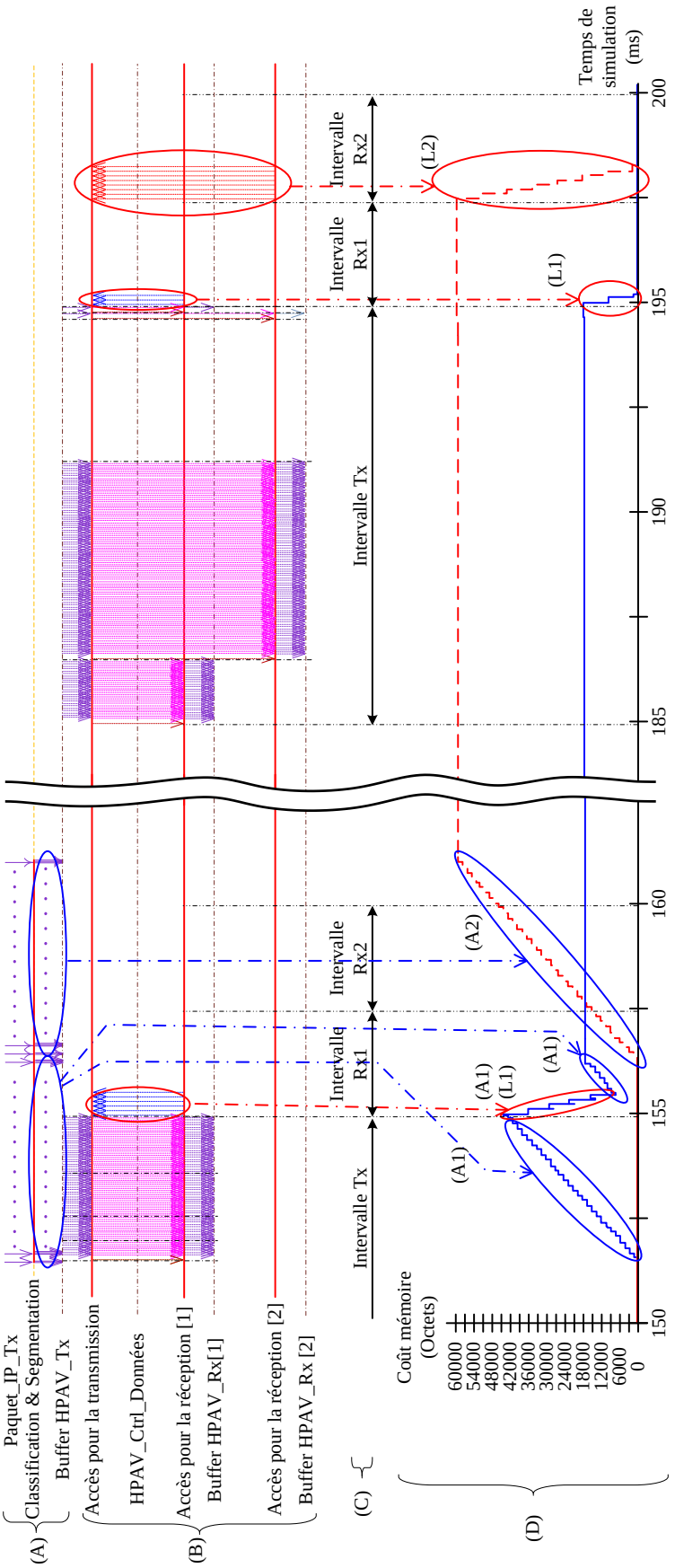


Figure 4.13 – Allocation de la mémoire au sein de l'interface HPAV de l'émetteur.

On distingue quatre parties sur la Figure 4.13. La partie (A) présente l'évolution de l'activité « Classification & Segmentation » en fonction des arrivées des paquets IP. La partie (B) présente l'évolution des activités « Accès pour la transmission », « Accès pour la réception » des deux récepteurs, ainsi que les transactions échangées entre elles. La partie (C) présente les allocations TDMA pour l'émetteur et les deux récepteurs. Les échanges présentés dans la partie (A) et (B) sont effectués en fonction des allocations présentées par la partie (C). Les explications précédentes des différents échanges sont valables pour le scénario de la Figure 4.13. La partie (D) présente l'évolution de la quantité de mémoire allouée afin de stoker les PBs dans le buffer de l'interface HPAV de l'émetteur. Deux courbes sont observées afin de distinguer la mémoire allouée pour les données destinées à chaque récepteur. Les courbes en trait continu et en trait interrompu correspondent respectivement à la quantité de mémoire utilisée pour la transmission vers les récepteurs 1 et 2. Les zones notées (A1) et (A2) correspondent respectivement à l'allocation de la mémoire pour le stockage des PB suite à la réception des paquets IP destinés au premier et au deuxième récepteur. Les zones notées (L1) et (L2) correspondent respectivement à la libération de mémoire suite à la réception des acquittements de la part du premier et du deuxième récepteur.

La Figure 4.14 montre l'évolution en fonction du temps de la quantité de mémoire allouée au sein de l'interface HPAV de l'émetteur lors de la transmission de trois images.

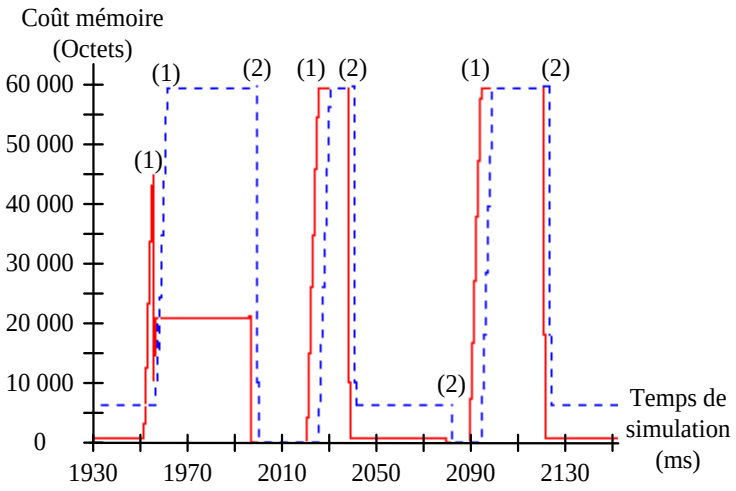


Figure 4.14 – Allocation de la mémoire au sein de l'interface HPAV de l'émetteur.

Les courbes en trait continu et interrompu représentent respectivement l'allocation de la mémoire pour la transmission d'images destinées au récepteur 1 et 2. Les instants (1) correspondent à l'allocation de mémoire pour le stockage des PBs lors de la réception des paquets IP. Les instants (2) correspondent à la libération de la mémoire suite à la réception des acquittements. Les valeurs sont obtenues par la simulation du modèle avec les paramètres précisés dans le Tableau 4.1. A titre d'information, dans ces conditions, la mémoire totale allouée au sein de l'interface HPAV ne dépasse pas 120 Koctets.

La Figure 4.15 présente l'utilisation de mémoire de l'interface HPAV du premier récepteur pour la réception d'une image.

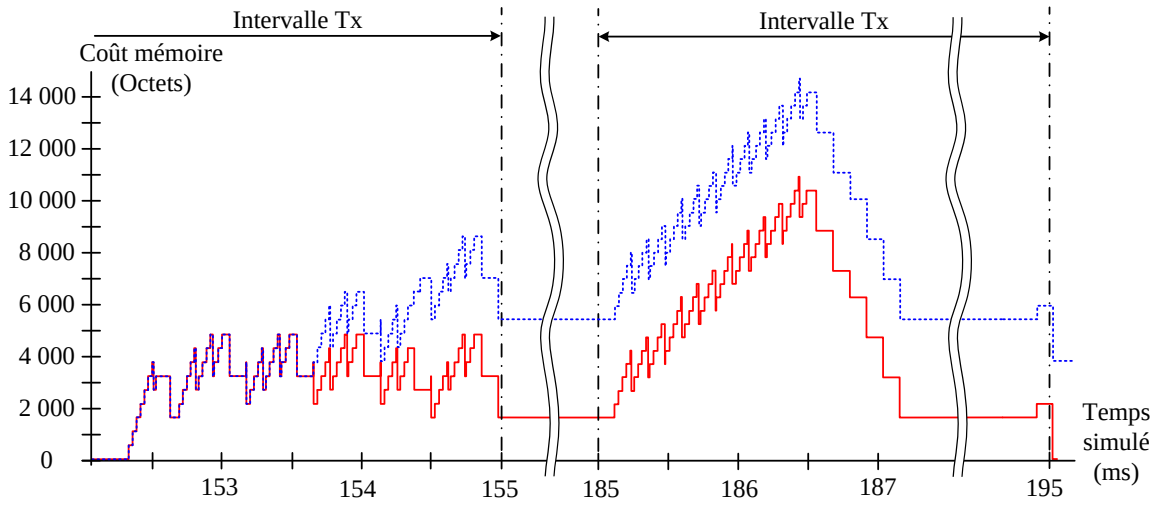


Figure 4.15 – Allocation de la mémoire au sein de l'interface HPAV du récepteur avec et sans erreur sur la transmission.

La courbe en ligne continue représente l'évolution de l'allocation de la mémoire avec aucune erreur lors de la transmission. La deuxième courbe en ligne pointillée, représente l'évolution de l'allocation de la mémoire avec 2% d'erreur au cours de la transmission des PBs. Chaque allocation de mémoire correspond à la réception correcte d'un PB. La libération de mémoire correspond à la suppression de PB qui ont contribué à une reconstruction de paquet IP. Dans son intervalle TDMA, l'émetteur commence à recevoir des paquets IP à 152 ms. Il envoie alors les PB jusqu'à la fin de son intervalle TDMA. L'émetteur reprend la transmission au début de son intervalle TDMA suivant, à 185ms. A 186,5 ms, il termine la transmission de tous les PBs, mais il reste une quantité de données inferieure au seuil. Ces données sont converties en PBs et transmises avant 195 ms. Pendant ce temps, le récepteur récupère les PBs, les stocke et reconstruit, si possible, les paquets IP. Dans ce scénario, s'il y a des erreurs sur la transmission des PB, la quantité de mémoire utilisé par l'interface HPAV du récepteur augmente et peut ne pas être totalement libérée à 195 ms. Ceci s'explique par la sauvegarde des PBs qui ne permettent pas encore de reconstruire un paquet IP en attendant la retransmission du PB reçu erroné et qui contient le reste de ce paquet IP.

Il est possible d'observer les performances temporelles du système modélisé. La Figure 4.16 montre la latence observée par le premier récepteur sur la réception des images.

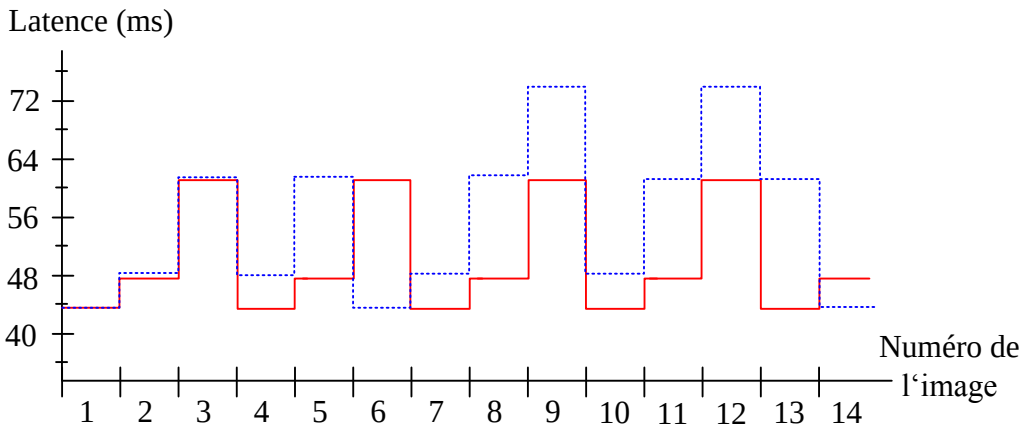


Figure 4.16 – Latence de réception des images avec et sans erreur sur la transmission.

La courbe en trait continu représente la latence observée par le premier récepteur pour la réception des images sans erreur sur la transmission. Dans ce cas, les valeurs de latence minimale et maximale sont respectivement 43,6 ms et 61,1 ms. La valeur moyenne de latence est de 49,7 ms pour la transmission de 45 images. La courbe en trait interrompu représente la latence de réception des images avec un taux de PBs irrécupérables égal à 1%. Dans ce cas, la valeur de latence minimale est de 43,6 ms, la valeur de latence maximale est de 73,8 ms, et la valeur des temps de latence moyen est de 52,1 ms. La valeur de latence augmente à cause des durées de retransmission des PB erronés. Elle peut augmenter considérablement si la retransmission des PBs erroné demande un cycle de communication supplémentaire.

Après avoir exposé différents résultats de simulation permettant de comprendre le fonctionnement de l'architecture étudiée, nous présentons l'application de la technique proposée.

4.3 Application de la technique proposée pour la modélisation des interfaces HPAV

4.3.1 Principe de l'application de la technique de modélisation

Cette section explique l'application de la technique proposée pour la modélisation des activités « Accès pour la transmission » et « Récepteur HPAV ». L'architecture formée par les interfaces HPAV est alors décrite comme présenté sur la Figure 4.17.

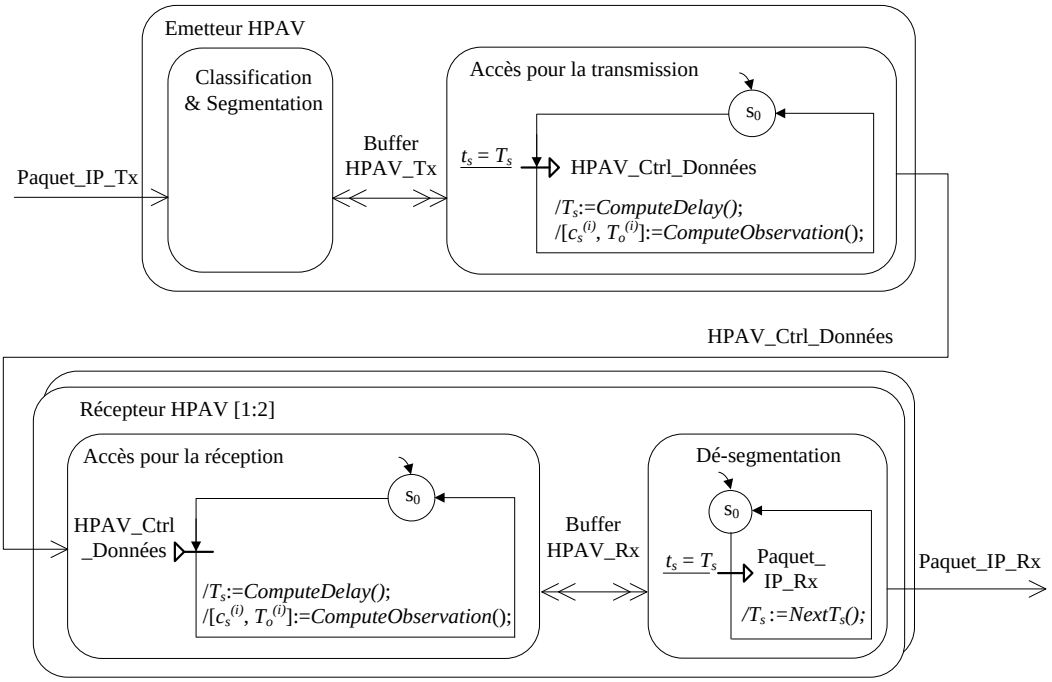


Figure 4.17 – Automate des interfaces HPAV après application de la technique.

Dans la Figure 4.17, la technique de modélisation proposée est appliquée pour la modélisation des activités « Récepteur HPAV » des deux récepteurs et « Emetteur HPAV ». L'application de la technique conduit à modifier la description faite du comportement des activités « Accès pour la transmission » et « Accès pour la réception». Par rapport à la description présentée dans la section précédente, le calcul des instants significatifs d'évolution permet de s'affranchir de l'échantillonnage des activités « Accès pour la transmission » et « Accès pour la réception». L'application de la technique porte sur trois aspects :

- l'accroissement du niveau de granularité des données échangées au sein du modèle,
- le calcul des instants d'évolution,
- le calcul des observations.

Dans la suite nous expliquons successivement ces différents points.

- **Accroissement du niveau de granularité des données échangées**

Précédemment, le grain d'échange au niveau de la relation « HPAV_Ctrl_Données » correspondait à un seul PB. Dans le modèle présenté sur la Figure 4.17, une transaction regroupe un nombre variable de PBs tel que présenté dans la Figure 4.18. Une transaction contient également les instants supposés de réception de ces PBs.

Transaction				
SOF	PB_1	PB_2	...	PB_n
T_{SOF}	T_{PB1}	T_{PB2}	...	T_{PBn}

Figure 4.18 – Contenu d'une transaction.

Initialement, des PBs sont stockés au sein de la relation interne « Buffer HPAV_Tx ». Au moment de l'échange de la transaction entre « Accès pour la transmission » et « Accès pour la réception », un ensemble de PBs est formé puis transmis sous la forme d'une seule transaction via HPAV_Ctrl_Données. Le nombre des PBs contenus dans une transaction est variable et dépend du temps restant de l'intervalle TDMA de l'émetteur. En effet, l'instant supposé de réception du dernier PB ne doit pas dépasser la fin de cet intervalle. De plus, le nombre maximal des PBs contenus dans une transaction ne dépasse pas le nombre des PBs en attente de transmission dans « Buffer HPAV_Tx » à l'instant de construction de cette transaction.

Comme nous l'expliquerons par la suite, il est également nécessaire d'associer à chaque PB l'instant, noté T_{PBi}, de sa réception par le récepteur correspondant. Ces instants seront nécessaires à l'activité « Accès pour la réception » afin de calculer les instants d'évolution des récepteurs et les observations considérées. La durée de transmission d'un SOF est fixe tandis que la durée de transmission d'un PB est déterminée à partir de sa taille et du débit considéré pour la couche MAC.

- **Calcul des instants d'évolution**

L'action *ComputeDelay()* de l'activité « Accès pour la transmission » détermine les instants d'évolution T_s. La Figure 4.19 illustre les différentes situations rencontrées en cours d'exécution du modèle et les différentes façons de mener ce calcul.

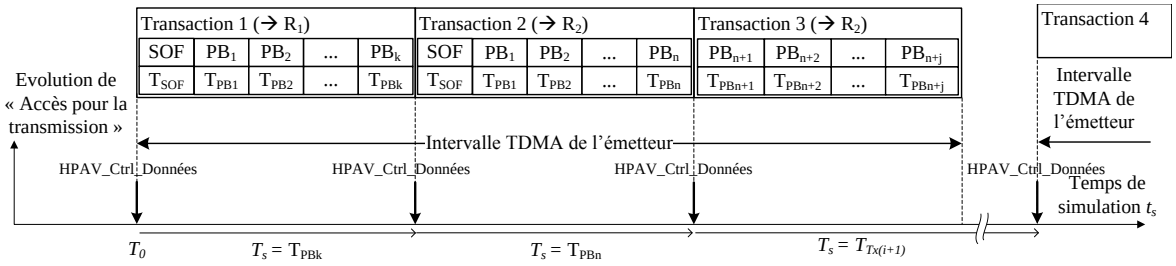


Figure 4.19 – Calcul des instants d'évolution.

La Figure 4.19 représente trois situations différentes : le transfert de données vers le récepteur 1 (Transaction 1) et le transfert de données vers le récepteur 2 (Transaction 2 et Transaction 3). Ces transferts s'effectuent au sein du même intervalle TDMA. Dans cet exemple, on suppose qu'à l'instant T_0 il existe au sein de « Buffer HPAV_Tx » des PBs à transmettre pour les deux récepteurs.

A l'instant T_0, l'échange de Transaction 1 est effectué et l'instant suivant d'évolution T_s est déterminé. Dans la situation présentée, le calcul correspond au fait de déterminer la valeur T_{PBk}, correspondant à l'instant de réception du dernier PB constituant la transaction. De façon analytique cette valeur s'exprime par :

$$T_{PBk} = T_0 + D_{SOF} + k \times D_{PB}.$$

D_{SOF} et D_{PB} sont respectivement les durées de transmission d'un SOF et d'un PB. D_{SOF} est fixé dans la norme à la valeur 110,48 µs et D_{PB} est donné dans le Tableau 4.1.

Le transfert de la Transaction 2 vers le récepteur 2 s'effectue à l'instant T_{PBk}. Dès lors tous les PBs à transmettre pour les deux récepteurs ont été pris en compte. L'instant suivant d'évolution est déterminé de la même façon que précédemment et correspond à l'instant T_{PBn}.

A l'instant T_{PBn}, de nouvelles données à transmettre pour le récepteur 2 ont été reçues. A cet instant, Transaction 3 est construite et transmise au récepteur 2. Le nombre de PBs contenus dans Transaction 3 est limité compte tenu du fait que ces données doivent pouvoir être transmises avant la fin de l'intervalle TDMA. Dans le cas où des PBs resteraient à transmettre, *ComputeDelay()* fixerait le prochain instant d'évolution à l'instant du début du prochain intervalle TDMA de l'émetteur.

La Figure 4.20 illustre une quatrième situation rencontrée pour la détermination de l'instant suivant d'évolution.

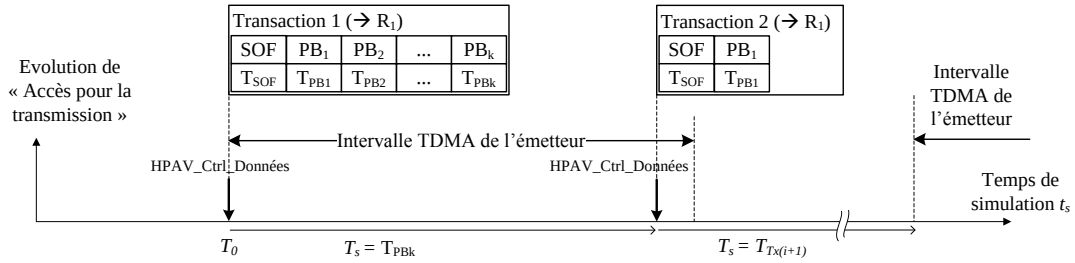


Figure 4.20 – Calcul des instants d'évolution pour *« $T_{WaitLimit}$ »*.

Dans la situation présentée dans la Figure 4.20, à l'instant T_0, Transaction 1 contient tous les PBs contenus dans « Buffer HPAV_Tx ». Cependant, certaines données ne permettant pas de constituer un PB complet subsistent. Dans la situation où ces données doivent être envoyées avant la fin de l'intervalle TDMA actuel de l'émetteur, elles doivent être bourrées afin de former un PB complet. L'action *ComputeDelay()* détermine alors l'instant prochain d'évolution à l'aide de l'équation suivante :

$$T_s = T_{finTx} - D_{SOF} - D_{PB}.$$

T_{finTx} correspond à l'instant de fin de l'intervalle TDMA actuel de l'émetteur.

Ces différentes situations illustrent la façon dont les instants d'évolution peuvent être déterminés dynamiquement au cours de la simulation. Ces instants dépendent essentiellement du nombre de PBs transmis et de l'utilisation possible des intervalles TDMA de transmission de l'émetteur. Au niveau des récepteurs la détermination des instants d'évolution est effectuée de manière différente.

Au niveau des récepteurs, le contenu des transactions envoyées est copié dans la relation « Buffer HPAV_Rx ». A partir des PBs stockés dans cette relation et de leurs instants supposés de réception, l'action *ComputeDelay()* de « Accès pour la réception» détermine les instants suivant d'évolution de « Dé-segmentation ». La Figure 4.21 explique comment ces instants sont déterminés.

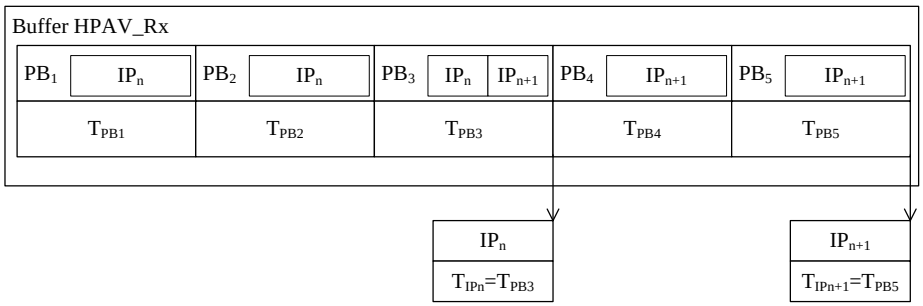


Figure 4.21 – Interprétation d'une transaction et reconstruction des paquets IP.

Dans cet exemple, la relation « Buffer HPAV_Rx » contient cinq PBs et leurs instants supposés de réception déterminés préalablement. La détermination des instants d'évolution de « Dé-segmentation » est faite en considérant les instants où un paquet IP peut être construit. Dans l'exemple de la Figure 4.21, PB_1, PB_2 et PB_3 contiennent toutes les données qui permettent de reconstruire le paquet IP_n et PB_3, PB_4 et PB_5 permettent également la reconstruction du paquet IP_{n+1}. Afin de reconstruire IP_n et IP_{n+1}, *ComputeDelay()* dresse donc une liste des instants suivants d'évolution. Ces instants sont déterminés à l'aide des expressions :

$$T_{s(n)} = T_{IPn} = \max (T_{PB1}, T_{PB2}, T_{PB3}).$$

$$T_{s(n+1)} = T_{IPn+1} = \max (T_{PB3}, T_{PB4}, T_{PB5}).$$

La liste ordonnée des instants suivants d'évolution est communiquée à l'activité « Dé-segmentation » via la relation « Buffer HPAV_Rx ».

- **Calcul des observations**

On explique tout d'abord l'application de la technique pour l'observation de l'utilisation de la mémoire au sein de l'interface HPAV émettrice. Typiquement, au fur et mesure de la réception des paquets IP, la mémoire est allouée au sein de l'émetteur. Cette mémoire est libérée lors de la réception des acquittements fournis par les récepteurs. L'application de la technique de modélisation consiste à estimer les instants d'acquittements. La Figure 4.22 explique ce principe.

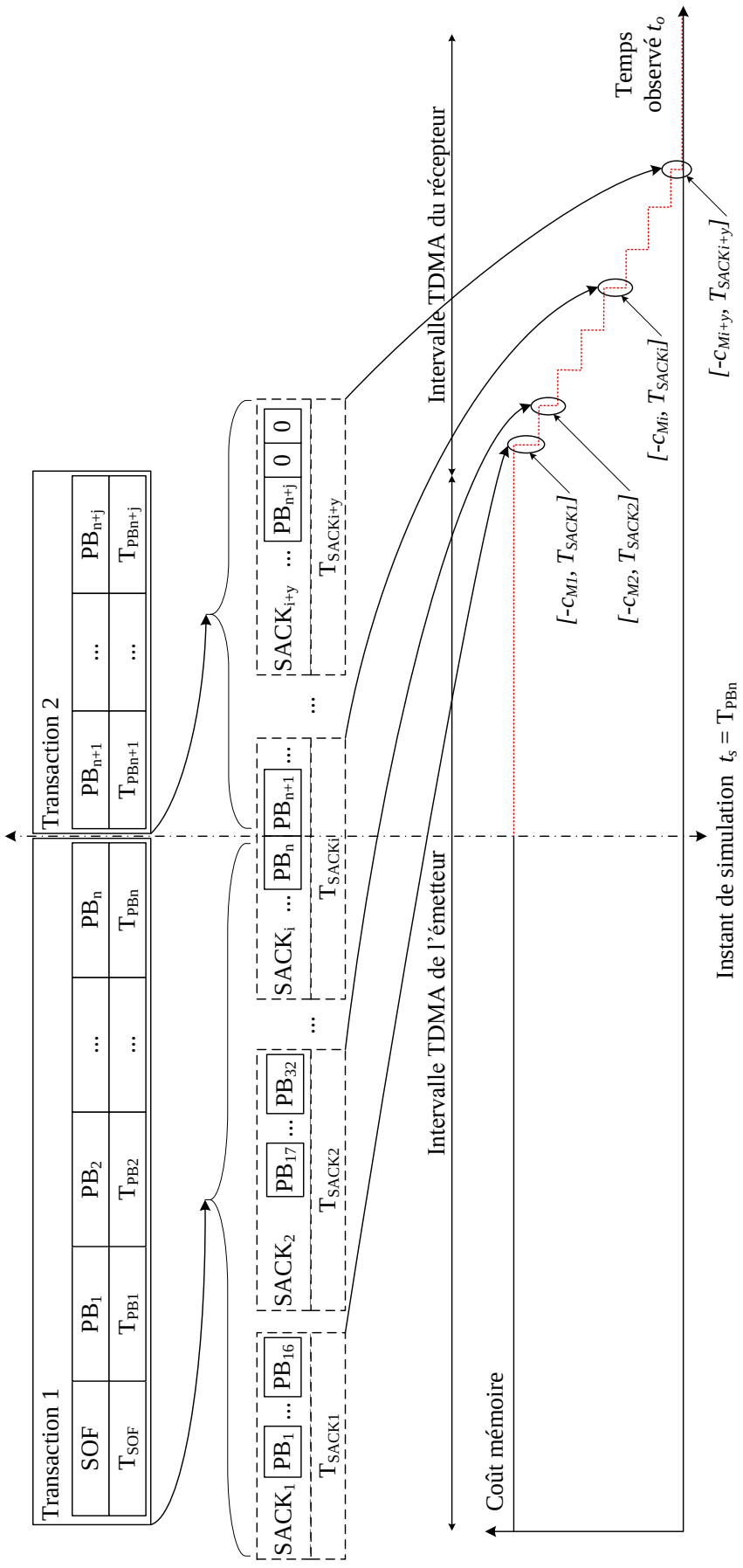


Figure 4.22 – Estimation des acquittements au niveau de l'émetteur.

On rappelle qu'un SACK est supposé acquitter 16 PBs au maximum. Dans la Figure 4.22 , l'instant T_{SACKi} correspond à l'acquittement effectué par $SACK_i$. Les instants T_{SACK1} jusqu'à T_{SACKi} où ces acquittements devraient être reçus par « Accès pour la transmission » peuvent être déterminés à l'aide de l'expression :

$$T_{SACKi} = T_{débutRx} + i \times D_{SACK} .$$

$T_{débutRx}$ correspond à l'instant du début de l'intervalle TDMA du récepteur concerné. D_{SACK} correspond à la durée de transmission d'un acquittement.

A partir de ces estimations, *ComputeObservation()* calcule les couples $[C_s^{(i)}, T_0^{(i)}]$ correspondants aux fluctuations de la mémoire allouée :

$$[C_s^{(i)}, T_0^{(i)}] = [-C_{Mi}, T_{SACKi}].$$

où $-C_{Mi}$ correspond à la quantité de mémoire libérée :

$$C_{Mi} = S_{PB} \times Nb_{PB}(SACK_i).$$

S_{PB} correspond à la taille d'un PB. $Nb_{PB}(SACK_i)$ correspond au nombre de PBs positivement acquittés par le $i^{ème}$ SACK.

Au niveau des récepteurs, l'allocation de la mémoire s'effectue lors de la réception des PBs, la mémoire est libérée à la fin de la transmission des paquets IP. *ComputeObservation()* de « Accès pour la réception » déduit les couples $[C_s^{(i)}, T_0^{(i)}]$ correspondant aux allocations de la mémoire au niveau du récepteur :

$$[C_s^{(i)}, T_0^{(i)}] = [S_{PB,} T_{PB1}].$$

ComputeObservation() déduit également, à partir des instants de reconstruction des paquets IP les couples correspondant à la libération de la mémoire :

$$[C_s^{(i)}, T_0^{(i)}] = [-S_{IPn}, T_{IPn} + D_{IPn}].$$

$$S_{IPn} = \sum_i Taille_i(Ensemble(PB)).$$

D_{IPn} correspond à la durée de transmission d'un paquet IP en fonction de sa taille et du débit de transmission de données considéré au niveau du lien Ethernet. Ensemble(PB) correspond à l'ensemble de PB qui permettent la reconstruction du paquet IP_n.

4.3.2 Résultats obtenus

4.3.2.1 Observation des propriétés

On présente dans la suite les résultats observés par application de la technique de modélisation pour la modélisation des interfaces HAPV.

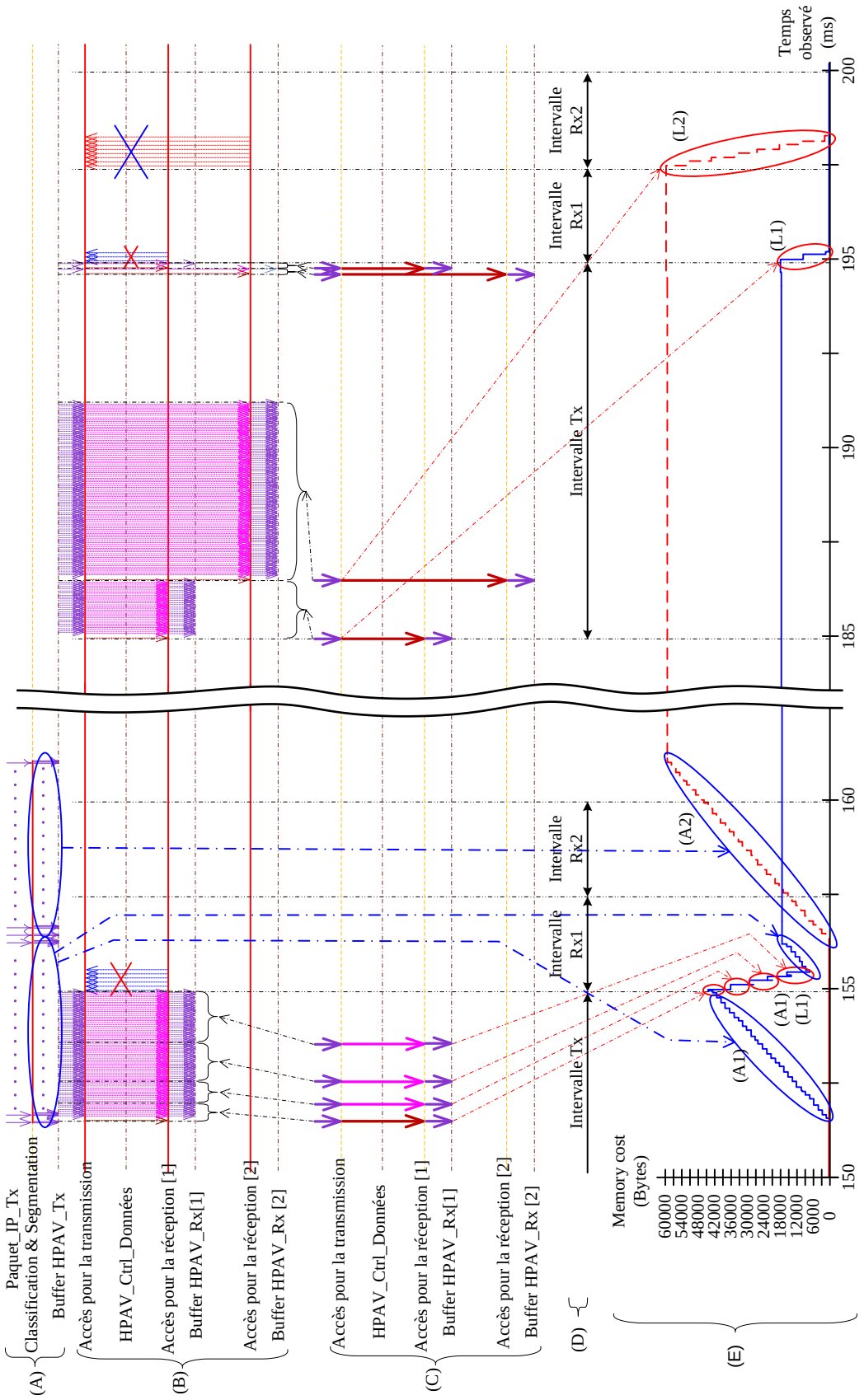


Figure 4.23 – Observation des transactions échangées et du cout mémoire en appliquant la technique de modélisation.

La partie (A) de la figure présente l'évolution de l'activité « Classification & Segmentation » en fonction des arrivées des paquets IP. Les parties (B) et (C) de la figure montrent les transactions échangées entre les activités des fonctions « Accès pour la transmission » et « Accès pour la réception » pour les deux récepteurs. La partie (B) montre ces échanges observés à partir du modèle transactionnel précédemment présenté au sein de la partie 4.1. La partie (C) montre les échanges observés à partir du modèle transactionnel utilisant la technique de modélisation proposée. La partie (D) de la figure présente les allocations TDMA pour l'émetteur et les deux récepteurs. La partie (E) de la figure montre l'évolution de la quantité de mémoire allouée afin de stocker les PBs dans le buffer de l'émetteur HPAV. Les deux courbes présentées traduisent la quantité de mémoire allouée pour les données destinées à chaque récepteur.

En faisant une comparaison entre les parties (B) et (C), les transactions échangées dans la partie (C) sont censés remplacer celles de la partie (B). L'application de la technique de modélisation proposée conduit dans (C) à grouper en une seule transaction plusieurs transactions transmises en (B). Les informations des transactions dans (B) et leurs instants supposés de réception sont inclus dans les transactions considérées dans (C). Les instants supposés de réception des transactions permettent aux récepteurs de pouvoir tracer l'évolution des ressources utilisées. Une transaction dans (C) couvre un intervalle de temps déterminé. A la fin de cet intervalle une autre transaction doit être produite pour couvrir progressivement la totalité de l'intervalle TDMA de l'émetteur. Les allocations de mémoire (A1) et (A2) sont calculées aux instants de réception des transactions « Packet_IP_Tx ». Les instants de libération de la mémoire (L1) et (L2) sont calculés aux instants de production des transactions « HPAV_Ctrl_Données ».

4.3.2.2 Mesures des temps de simulation

Dans cette section, nous cherchons à quantifier l'accélération des temps d'exécution des modèles apportée par l'application de la technique de modélisation proposée. Dans un premier temps, nous présentons les résultats obtenus pour la simulation des interfaces de communication HPAV seules, dans le cas d'un émetteur et d'un unique récepteur. Le Tableau 4.2 donne différents résultats obtenues.

	Temps simulé (s)	10	100	300
Modèle transactionnel uniquement	Nombre de changements de contexte	20 024 300	200 243 000	600 729 000
	Temps d'exécution (s)	136,110	1 352,766	4 067,375
Modèle transactionnel utilisant la technique de modélisation	Nombre de changements de contexte	48 059	480 820	1 442 490
	Temps d'exécution (s)	0,922	8,313	24,844
Ratio du nombre de changements de contexte		416,66	416,46	416,45
Ratio des temps d'exécution		147,62	162,72	163,71

Tableau 4.2 – Mesures des temps de simulation des modèles des interfaces HPAV.

Les mesures présentées dans le Tableau 4.2 correspondent aux résultats obtenus par les simulations des deux modèles présentés précédemment, avec ou sans application de la technique de modélisation proposée. Trois situations ont été simulées, correspondant à des temps de simulation de 10, 100 et 300 s. Pour ces trois situations les temps d'exécution des modèles sont mesurés. Le ratio obtenu des temps d'exécution montre un facteur d'accélération de l'ordre de 160 entre le modèle utilisant la technique de modélisation proposée et le modèle de référence des interfaces HPAV. Nous avons également estimé le nombre de changements de contexte pour chaque modèle. Le rapport estimé est alors de l'ordre de 416. La différence observée entre les deux ratios peut s'expliquer par la différence de tailles de données considérées par transaction. En effet, dans le cas du modèle utilisant la technique de modélisation proposée, la taille des données échangées par transaction est 50 fois plus importante que pour le cas du modèle de référence, représentant alors 56000 octets. Nous avons pu observer au chapitre 3 dans quelle mesure l'accroissement des tailles de données pouvait influencer sur les temps de simulation. Un autre aspect pouvant influer sur le temps de simulation du modèle mettant en œuvre la technique de modélisation proposée peut porter sur l'accroissement de la complexité des algorithmes au sein du modèle. En effet, dans le cas du modèle mettant en œuvre la technique de modélisation il est nécessaire de calculer au cours de la simulation les instants d'évolution. Enfin, nous pouvons également évoquer certains mécanismes spécifiques à l'outil CoFluent Studio qui peuvent avoir une influence sur les temps d'exécution mesurés.

Le Tableau 4.3 présente différents résultats de simulation obtenus cette fois en considérant l'architecture représentée sur la Figure 4.5. Les modèles simulés comportent alors la description de trois ECUs, un émetteur et deux récepteurs, mettant en œuvre les protocoles RTP, UDP, IP, et HPAV.

	Temps simulé (s)	10	100	300
Modèle transactionnel uniquement	Nombre de changements de contexte	30 036 450	300 364 500	901 093 500
	Temps d'exécution (s)	249,360	2 498,578	7 459,328
Modèle transactionnel utilisant la technique de modélisation	Nombre de changements de contexte	73 252	732 502	2 198 389
	Temps d'exécution (s)	1,328	12,047	35,907
Ratio du nombre de changement de contexte		410,04	410,05	409,88
Ratio des temps de simulation		187,77	200,88	199,16

Tableau 4.3 – Mesures des temps de simulation obtenus pour les modèles du démonstrateur CIFAER.

Des observations similaires aux précédentes peuvent être faites. La technique de modélisation proposée conduit à une accélération de l'exécution des modèles d'un facteur 200. L'observation de l'utilisation des ressources mémoires est préservée entre les deux types de modèles ainsi que la prédiction du comportement temporel de l'architecture.

L'étude de cas présentée a permis d'aborder l'application de la technique de modélisation proposée sur un exemple d'une complexité représentative des architectures du domaine automobile. La technique de modélisation a été appliquée afin d'améliorer les temps d'exécution des modèles. Le premier modèle décrit a permis d'expliquer le fonctionnement du protocole de communication, son influence sur les performances de l'architecture globale ainsi que l'utilisation faite des ressources de mémorisation. Le second modèle présenté a permis d'expliquer la mise en œuvre de la technique de modélisation. Cette mise en œuvre consiste en un calcul dynamique des instants significatifs d'évolution des modèles de l'architecture, permettant ainsi de réduire le nombre de transactions au sein des modèles. L'évolution des observations associées est également calculée localement en cours de simulation afin de préserver la précision des résultats. Utilisant ce principe, nous avons également pu intégrer la prise en compte des erreurs de transmission. Ainsi, en utilisant la technique proposée, cette prise en compte n'influe pas sur les temps d'exécution des modèles. Sur les mesures présentées, nous relevons une accélération significative des temps d'exécution. Ainsi, il devient possible, pour les architectes de pouvoir plus rapidement explorer les différentes possibilités de conception de ces architectures.

Conclusion et perspectives

Dans le domaine automobile, l'évolution des services rendus au sein des véhicules et l'augmentation du nombre des fonctionnalités assurées par la partie électronique ont pour effet de densifier et de complexifier les architectures électroniques. Ces architectures embarquées sont formées d'un ensemble de cartes électroniques interconnectées selon différents réseaux de communication. La conception de ces systèmes s'avère alors complexe et suppose de pouvoir appréhender les architectures dans leur ensemble et de prendre en compte l'influence des échanges entre calculateurs au plus tôt dans le processus de conception. Dans ces phases, l'architecte de systèmes est amené à dimensionner les architectures en définissant les ressources matérielles et logicielles nécessaires permettant de supporter les besoins applicatifs auxquels doivent répondre le système. Dans ce contexte, différentes approches et méthodes basées sur l'emploie de modèles d'architectures ont été proposées afin de favoriser le travail des architectes de systèmes. Ces modèles sont définis à un haut niveau d'abstraction afin de permettre une exploration rapide et efficace des différentes solutions de réalisation envisageables. Dès lors, le concept de modélisation transactionnelle présente une solution intéressante pour la création de modèles d'architectures afin de faciliter le dimensionnement de ces systèmes. La simulation de ces modèles permet une évaluation du comportement dynamique des architectures et des performances associées. Les modèles peuvent être définis à différents niveaux d'abstraction conduisant à un compromis entre les temps de simulation et le niveau de précision des résultats obtenus par ces modèles. Il devient alors nécessaire d'améliorer le compromis entre les temps de simulation et le niveau de précision des modèles transactionnels. Dans ce contexte, le travail de thèse a porté sur la définition d'une technique de modélisation permettant d'améliorer ce compromis pour la création de modèles transactionnels en vue du dimensionnement des ressources matérielles et logicielles des architectures électroniques embraquées dans les véhicules.

Le premier chapitre a permis d'introduire le contexte applicatif de notre travail. L'analyse de l'évolution des architectures électroniques dans le domaine automobile suite à l'augmentation du nombre de fonctionnalités à intégrer a illustré la densification des réseaux embarqués au sein des véhicules, la densification des ressources matérielles disponibles au sein des ECUs et la diversification des ressources logicielles. L'analyse du processus de conception de ces systèmes a démontré l'augmentation de la complexité des phases de développement et les limites des méthodes de conception actuelles face à l'interdépendance croissante entre les ECUs. Cette analyse a permis d'identifier de nouvelles problématiques vis-à-vis du dimensionnement des architectures électroniques distribuées. L'augmentation du nombre de fonctionnalités à intégrer et l'accroissement de l'interdépendance entre les ECUs conduisent à un élargissement de l'espace de conception à explorer. La maîtrise de la complexité amenée par ces évolutions implique la définition et l'adoption de nouvelles méthodes, s'appuyant sur l'utilisation de modèles de haut niveau, afin d'identifier et de caractériser rapidement et efficacement une solution architecturale.

Le second chapitre a permis de positionner notre travail parmi les différents travaux portant sur l'amélioration des processus de dimensionnement des architectures de systèmes embarqué. Parmi les différentes approches existantes, l'emploi de modèles d'architectures simulables s'avère indispensable afin de permettre d'évaluer le comportement dynamique et d'estimer l'utilisation des ressources et les caractéristiques associées selon différents scénarios possibles de fonctionnement. Les différentes approches que nous avons présentées proposent différentes méthodes de capture des propriétés des architectures. Ces propriétés sont ainsi traduites en une description exécutable, à l'aide de langages tels que SystemC. L'intérêt d'un tel langage réside dans la possibilité de pouvoir capturer les différents aspects d'une architecture d'un système embarqué tout en permettant une description selon différents niveaux de précision. Il en résulte un compromis entre la vitesse de simulation des modèles et la précision des estimations délivrées. Certains travaux ont permis d'évaluer ce compromis. L'objet de notre contribution a donc porté sur la définition d'une technique de modélisation visant à améliorer ce compromis afin de favoriser la création de modèles efficaces d'architectures.

Le troisième chapitre a permis de présenter notre contribution. La technique de modélisation proposée repose sur le constat qu'une réduction du nombre des transactions nécessaires au sein de modèles transactionnels conduit à un accroissement de la vitesse de simulation. Selon ce constat et afin de conserver la précision des observations, nous avons présenté un principe de modélisation permettant une exécution locale de certaines parties des modèles, sans influence sur le temps de simulation. Ce principe a été étendu afin de correctement gérer la préemption possible entre les modules constituant les modèles d'architectures. Les méthodes mises en œuvre ont été illustrées dans le cas de la modélisation de ressources de communication. Par ailleurs, nous avons cherché à qualifier l'apport de la technique de modélisation selon les critères d'accélération du temps de simulation et selon la précision des résultats. Pour autant, il aurait été intéressant de pouvoir mener ce travail jusqu'à une confirmation pour un prototype réel et l'étude de cas présentée à la fin du chapitre 3 devra être prolongée.

Le quatrième chapitre a permis de présenter l'application de la technique de modélisation proposée sur un cas d'étude représentatif du domaine automobile. Ce cas d'étude a porté sur une architecture distribuée de calculateurs et sur le dimensionnement des ressources de mémorisation associées. Cette architecture reposait sur l'emploi d'un protocole de communication utilisant les technologies à base de courant porteur en ligne. L'application de la technique de modélisation proposée a permis d'améliorer la modélisation de ce protocole afin de réduire les temps de simulation nécessaires. Nous avons ainsi pu mesurer un facteur d'accélération d'un rapport 100 du temps de simulation, ouvrant la voie à une exploration rapide de l'espace de conception. Là aussi il aurait été intéressant de pouvoir confronter de telles estimations à un prototype réel.

A l'issue de ces travaux différentes perspectives de recherche peuvent être envisagées.

Une première perspective porte sur la mise en œuvre de la technique de modélisation. Dans le cadre de cette thèse nous avons utilisé l'environnement CoFlent Studio afin de pouvoir créer rapidement des modèles SystemC simulables et ainsi pouvoir mesurer le gain

apporté par la technique proposée. Pour autant, la technique de modélisation proposée ne se limite pas à l'utilisation de cet outil et pourrait être appliquée de façon plus générale au sein des différentes approches utilisant la notion de modélisation transactionnelle. Différentes études pourraient être menées dans ce sens afin de démontrer la portée de notre proposition.

Une seconde perspective porte sur la définition d'un modèle générique de comportement utilisant la technique de modélisation proposée. En effet, la technique de modélisation proposée permet de limiter la complexité de la description des modèles d'architectures. Le comportement traduisant l'utilisation faite des ressources se retrouve exprimé au sein de méthodes et les évolutions associées s'effectuent en temps nul vis-à-vis du simulateur. Ainsi, il pourrait s'avérer possible de définir un modèle générique de comportement et ce afin de faciliter la création de modèles d'architectures. Dans le domaine automobile, les architectures étudiées reposent sur l'emploi de différents protocoles de communication et il convient donc de proposer des solutions afin de limiter les temps de création des modèles pour ces différents protocoles.

Une autre perspective de nos travaux porte sur l'application des méthodes proposées à la modélisation des ressources de calcul. Ce travail a été initié dans le cadre de la thèse de Anthony Barreteau [**60**] mais sans prise en compte des préemptions éventuelles entre fonctions. Notre travail a permis de montrer de quelle manière la technique de modélisation proposée pouvait être efficacement mise en œuvre afin de correctement gérer la préemption éventuelle au sein des ressources de communication.

Un autre aspect porte sur l'obtention des estimations que nous avons utilisées au sein des modèles étudiés. En effet, de telles estimations peuvent être obtenues par des mesures préalables sur des prototypes existants ou à partir d'autres modèles disponibles. Il pourrait être intéressant d'étudier de quelles manières il serait possible d'améliorer, voire d'automatiser, l'intégration de ces estimations au sein de modèles de plus haut niveau d'abstraction. L'intérêt d'une telle étude réside toujours dans le fait de favoriser le processus de création des modèles d'architectures.

Enfin, une autre piste d'étude pourrait porter sur l'amélioration du processus d'exploration. Jusqu'à présent, nos travaux ont essentiellement porté sur la phase d'évaluation des performances des architectures mais cette phase s'inscrit le plus souvent dans un processus plus large d'exploration. Compte tenu des nombreux paramètres de conception et des différents critères qu'il convient de satisfaire, il demeure indispensable de pouvoir assister les concepteurs dans le processus d'exploration d'architectures distribuées.

Acronymes & Abréviations

ABS	**Anti Blockier System**
AUTOSAR	**Automotive Open System Architecture**
BSW	**Basic Software**
CA	**Cycle Accurate**
CAN	**Cntroller Area Network**
CFSM	**Codesign Finite State Machine**
CIFAER	**Communication Intra-véhicule Flexible et Architecture Embarquée Reconfigurable**
CPL	**Courant Porteur en Ligne**
CSMA/CD	**Carrier Sense Multiple Access with Collision Detection**
CSMA/CR	**Carrier Sense Multiple Access with Contention Resolution**
ECU	**Electronic Controller Unit**
FIFO	**First In First Out**
FPGA	**Field-programmable gate array**
FTP	**File Transfer Protocol**
FTT-CAN	**Flexible Time-Triggered CAN**
GPS	**Global Positioning System**
HTTP	**Hypertext Transfer Protocol**
I2C	**Inter Integrated Circuit**
IP	**Intellectual Property**
I-PDU	**Interaction Layer Protocol Data Unit**
KPN	**Kahn Process Network**
LIN	**Local Interconnect Network**
MIPS	**Mega Instructions Per Second**
MOST	**Media Oriented Systems Transport**
N-PDU	**Data Link Layer Protocol Data Unit**
NS2	**Network Simulator version 2**
OFDM	**Orthogonal Frequency-Division Multiplexing**
OS	**Operating System**
OSCI	**Open SystemC Iniatitive**
OSEK/VDX	**Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen/Vehicle Distributed eXecutive**
OSI	**Open Systems Interconnection**

PDM	**Pulse-Duration Modulation**
PIO	**Parallels Input/Output**
PWM	**Pulse Width Modulation**
RAM	**Random Access Memory**
ROM	**Result Oriented Modeling**
RTE	**Runtime Environment**
RTL	**Register Transfer Level**
SCE	**System-on-Chip Environment**
SDF	**Synchronous Data Flow**
Sesame	**Simulation of Embedded System Architectures for Multilevel Exploration**
SoC	**System on Chip**
SPADE	**System Level Performance Analysis And Design Space Exploration**
SPI	**Serial Peripheral Interface**
SymTA/S	**Symbolic Timing Analysis for Systems**
TAPES	**Trace-based Architecture Performance Evaluation with SystemC**
TCP	**Transmission Control Protocol**
TDEU	**Trace Driven Execution Unit**
TDMA	**Time division multiple access**
TLM	**Transaction Level Modeling**
TNT	**Télévision numérique terrestre**
TT-CAN	**Time-Triggered CAN**
UDP	**User Datagram Protocol**
USB	**Universal Serial Bus**
VFB	**Virtual Functional Bus**
VNA	**Volcano Network Architect**

Bibliographie

[1] TRW Automotive. [Online]. http://ir.trw.com/

[2] N. Navet and F. Simonot-Lion, *Networked embedded system, Chapitre 13 Trends in Automotive Communication Systems*, R. Zurawski, Ed. California, USA: Taylor & Francis Group, 2009.

[3] Projet CIFAER. [Online]. http://www.insa-rennes.fr/ietr-cifaer/

[4] [Online]. http://www.can-cia.de

[5] [Online]. http://www.lin-subbus.org

[6] [Online]. http://www.mostcooperation.com

[7] [Online]. http://www.flexray.com/

[8] H.-M. Pham, "Embedded computing architecture with dynamic hardware reconfiguration for intelligent automotive systems," UNIVERSITÉ DE RENNES 1 THÈSE 3970, 2010.

[9] [Online]. www.osek-vdx.org

[10] [Online]. www.autosar.org

[11] J. P. Calvez, *Spécification et conception des systèmes : une méthodologie; Chapitre 3: Cycle de développement d'un système*. Masson, 1992.

[12] "System Level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, vol. 19, no. 12, 2000.

[13] A. Sangiovanni-Vincentelli, "Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design," *Proceedings of the IEEE*, 2007.

[14] [Online]. http://www.symtavision.com/symtas.html

[15] [Online]. http://www.mentor.com/products/vnd/communication-management/vna/

[16] [Online]. www.opnet.com/

[17] [Online]. http://www.isi.edu/nsnam/ns/

[18] C. P. Luciano Lavano, *Embedded systems. Chapitre3: Design of embedded systems*, R. Zurawski, Ed. USA: Taylor & Francis Group, 2006.

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

[20] The Embedded Microprocessor Benchmark Consortium. [Online]. http://www.eembc.org/

[21] Berkeley Design Technology, Inc. [Online]. http://www.bdti.com/

[22] Standard Performance Evaluation Corporation. [Online]. http://www.spec.org/

[23] J. Schnerr, O. Bringmann, M. Krause, A. Viehl, and W. Rosentiel, *Model-based design for embedded systems. Chapitre 2: SystemC-based performance analysis of embedded systems*, P. J. M. Gabriela Nicolescu, Ed. USA: Taylor & Francis Group, 2010.

[24] A. Jantsch and I. Sander, "Models of computation and languages for embedded system design," *IEE Proceedings on computers and digital techniques, Special issue on embedded microeletronic systems*, vol. 152, 2005.

[25] L. Gomes, J. P. Barros, and A. Costa, *Embedded System Handbook, Chapitre 5: Modeling formalisms for embedded system design*, R. Zurawski, Ed. 2006.

[26] L. Lavagno, A. Sangiovanni-Vincentelli, and E. Sentovich, *System-Level Synthesis : Models of computation for embedded system design*. Kluwer Academic Publishers, 1998.

[27] D. Harel and M. Politi, *Modeling Reactive Systems With Statecharts*, McGraw-Hill, Ed. 1998.

[28] A. Davare, et al., "A next-generation design framework for platform-based design," *Design and Verification Conference (DV-CON)*, 2007.

[29] G. Kahn, "The semantics of simple language for parallel programming," *in proceedings of IFIP Congress*, pp. 471-475, 1974.

[30] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *proceedings of the IEEE*, vol. 75, no. 9, pp. 1234-1245, 1987.

[31] P. Lieverse, P. Van der Wolf, E. Deprettere, and K. Vissers, "A Methodolology for Architecture Exploration of Heterogeneous Signal Processing Systems," *IEEE Workshop on Signal Processing Systems*, 1999.

[32] A. D. Pimentel, C. Erbas, and S. Polstra, "A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels," *IEEE Transaction on Computers*, 2006.

[33] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A Framework for System-Level Modeling and Simulation of Embedded Systems Architectures," *EURASIP Journal on Embedded Systems*, 2007.

[34] A. D. Pimentel, "The Artemis Workbench for System-level Performance Evaluation of Embedded Systems," *International journal of embedded systems*, pp. 181-196, 2008.

[35] C. Haubelt, et al., "A SystemC-Based Design Methodology for Digital Signal Processing Systems," *EURASIP Journal on Embedded Systems*, 2007.

[36] W. Thomas, H. Andreas, and L. Gyoo-Yeong, "TAPES—Trace-based architecture performance evaluation with SystemC," *Springer Science*, pp. 157-179, 2006.

[37] J. Kreku, et al., "Combining UML2 Application and SystemC Platform Modelling for Performance Evaluation of Real-Time Embedded Systems," *EURASIP Journal on Embedded Systems*, 2008.

[38] J. Kreku, K. Tiensyrjä, and G. Vanmeerbeeck, "Automatic workload generation for system-level exploration based on modified GCC compiler," *Design Automation and Test in Europe (DATE)*, 2010.

[39] T. Arpinen, E. Salminen, T. D. Hämäläinen, and M. Hännikäinen, "Performance Evaluation of UML-2 Modeled Embedded Streaming Applications with System-Level Simulation," *EURASIP Journal on Embedded Systems*, 2009.

[40] O. M. G. (OMG), "A UML profile for MARTE, beta 1 specification," 2007.

[41] K. TERO, et al., "UML-Based Multiprocessor SoC Design Framework," *ACM Transactions on Embedded Computing Systems (TECS)* , vol. 5, no. 2, pp. 281-320, 2006.

[42] A. Viehl, B. Sander, O. Bringmann, and W. Rosenstiel, "Integrated Requirement Evaluation of Non-Functional System-on-Chip Properties," *Forum on Specification and Design Languages*, 2008.

[43] R. Dömer, et al., "System-on-Chip Environment: A SpecC-Based Framework for heterogeneous MPSoC Design," *EURASIP Journal on Embedded Systems,*, 2008.

[44] G. Daniel, D. Rainer, P. Junyu, and G. Andreas, *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers , 2000.

[45] F. Balarin, et al., "Metropolis: an integrated electronic system design environment," *Computer*, vol. 36, no. 4, pp. 45-52, 2003.

[46] A. Sangiovanni-Vincentelli, "Quo vadis, SLD? Reasoning about the trends and challenges of system level design," *Proceedings of the IEEE*, vol. 95, no. 3, Mar. 2007.

[47] J. P. Calvez and G. Nicolescu, *Spécification et validation des systèmes monopuces. Chapitre 2: Spécification et modélisation des systèmes logiciels/materiels*, N. G. JERRAYA Ahmed-Amine, Ed. France: LAVOISIER, 2004.

[48] L. Cai and D. Gajski, "Transaction level modeling: an overview," *in Proceedings of CODES+ISSS*, 2003.

[49] G. Schirner and R. Dömer, "Quantitative analysis of the speed/accuracy trade-off in transaction level modeling," *ACM Transactions on Embedded Computing Systems*, vol. 8, pp. 1-29, 2009.

[50] F. Ghenassia, *Transaction-level modeling with SystemC: TLM concepts and applications for embedded systems*. 2005.

[51] D. C. Black and J. Donovan, *SystemC: From the Ground Up*, 1st ed. Eklectic Ally Inc, 2004.

[52] J. Peeters, N. Ventroux, T. Sassolas, and L. Lacassagne, "A systemc TLM framework for distributed simulation of complex systems with unpredictable communication," *in proceedings of Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2011.

[53] N. Savoiu, S. K. Shukla, and R. Gupta, "Automated Concurrency Re-Assignment in High Level System Models for Efficient System-Level Simulation," *in Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2002.

[54] G. Schirner and R. Dömer, "Result-oriented modeling - A novel technique for fast and accurate TLM," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, Sep. 2007.

[55] M. Radetzki and R. S. Khaligh, "Accuracy-Adaptive Simulation of Transaction Level Models," *Design, Automation and Test in Europe, 2008. DATE '08* , 2008.

[56] R. Salimi Khaligh and M. Radetzki, "A latency, preemption and data transfer accurate adaptive transaction level model for efficient simulation of pipelined buses," *in proceedings of Forum on specification and Design Languages (FDL)*, 2008.

[57] M. Karner, C. Steger, R. Weiss, and E. Armengaud, "Optimizing HW/SW Co-simulation based on run-time model switching," *in proceeding of Forum specification and design languages FDL*, 2009.

[58] CoFluent Design. [Online]. http://www.cofluentdesign.com

[59] J. P. Calvez, *Embedded real-time systems*, P. Wiley Series In Software Engineering, Ed. 1993.

[60] A. Barreteau, "Techniques de modélisation transactionnelle pour le dimensionnement des futurs systèmes de radiocommunication mobiles," Ecole doctorale Sciences et technologies de l'information et mathématiques Thèse ED 503-116, 2010.

[61] P. Tanguy, "Etude et optimisations d'une communication à haut débit par courant porteur en ligne pour l'automobile," INSA de Rennes Thèse, 2012.

[62] HomePlug Powerline Alliance. [Online]. www.homeplug.org

[63] I. Powerline Alliance, "HomePlug AV White Paper," Patent HPAVWP-050818, 2005.

[64] T. Majdoub, S. Le Nours, O. Pasquier, and F. Nouvel, "Performance evaluation of an automotive distributed architecture based on HPAV communication protocol using a transaction level modeling approach," *in proceedings of The Conference on Design and Architectures for Signal and Image Processing, DASIP*, 2011.

Publications et communications

Conférences internationales avec actes et comité de lecture

T. MAJDOUB, S. LE NOURS, O. PASQUIER, F. NOUVEL: *Transaction level modeling of a networked embedded system based on a power line communication protocol*, 14th Euromicro Conference on Digital System Design, DSD'2011, 31 août-2 septembre 2011, Oulu, Finlande

T. MAJDOUB, S. LE NOURS, O. PASQUIER, F. NOUVEL: *Performance evaluation of an automotive distributed architecture based on HPAV communication protocol using a transaction level modeling approach*, The 2011 Conference on Design and Architectures for Signal and Image Processing, DASIP 2011, 2-4 Novembre 2011, Tampere, Finlande

T. MAJDOUB, S. LE NOURS, O. PASQUIER, F. NOUVEL: A*pplication of temporal decoupling to the creation of efficient performance models of automotive architectures*, The 2012 Conference on Design and Architectures for Signal and Image Processing, DASIP 2012, 23-25 Octobre 2012, Karlsruhe, Allemagne

Colloques nationaux

T. MAJDOUB, S. LE NOURS, O. PASQUIER, F. NOUVEL: *Modélisation transactionnelle d'une architecture distribuée en vue du dimensionnement des ressources de mémorisation*, GDR SoC-SiP CNRS, 15-17 juin 2011, Lyon

T. MAJDOUB, S. LE NOURS, O. PASQUIER, F. NOUVEL: *Technique de modélisation transactionnelle en vue de la réduction des temps de simulation des modèles de performances des architectures*, GDR SoC-SiP CNRS, 13-15 juin 2012, Paris

Communications orales

T. MAJDOUB, S. LE NOURS: *Modélisation transactionnelle d'architectures distribuées, reconfigurables, tolerantes aux fautes*, JDOC 14 avril 2011, Nantes, France

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Takieddine MAJDOUB

Technique de modélisation transactionnelle en vue de l'amélioration de la simulation des modèles de performances des architectures électroniques dans le domaine automobile

Transaction level modeling technique for improving the simulation of performance models of electronic architectures in the automotive domain

Résumé

Dans le domaine automobile, l'architecture électronique des véhicules repose sur un ensemble de calculateurs interconnectés au travers différents réseaux de communication. Compte tenu des évolutions applicatives observées, la conception de ces systèmes tend à se complexifier. Il s'avère alors indispensable de pouvoir prendre en compte au plus tôt lors des phases de conception l'influence des échanges entre calculateurs. Afin d'aider les architectes à concevoir de telles architectures, différentes méthodes basées sur l'utilisation de modèles ont été proposées. Les travaux de recherche présentés dans cette thèse visent à tirer parti des possibilités offertes par le concept de modélisation transactionnelle (*TLM*) pour améliorer la simulation des modèles des architectures électroniques dans le domaine automobile. Notre travail a porté sur la définition d'une technique de modélisation permettant d'améliorer le compromis entre la rapidité de simulation et la précision des résultats des modèles d'architectures afin de favoriser l'exploration de l'espace de conception. La technique de modélisation proposée vise à favoriser la création de modèles transactionnels d'architectures distribuées en vue du dimensionnement des ressources matérielles et logicielles de ces architectures. L'intérêt de nos travaux a été illustré à travers l'étude d'un démonstrateur défini dans le cadre du projet CIFAER. Cette étude porte sur la modélisation en vue du dimensionnement de l'architecture de ce démonstrateur. La modélisation proposée de ce cas d'étude a permis de vérifier l'accélération significative de la simulation obtenue par l'application de la technique proposée.

Mots clés

Architecture électronique automobile distribuée, modélisation transactionnelle, simulation à événements discrets, évaluation des performances, dimensionnement et exploration architecturale.

Abstract

In the automotive domain, vehicle electronic architecture is based on a set of electronic boards connected through different communication networks. Due to application evolutions, the design of these systems tends to become more complicated. Then, it is essential to take into account at the earliest design phases the influence of exchanges between electronic boards. In order to help architects to design such architectures, different methods based on the use of models have been proposed. The research presented in this thesis aims to take advantage of the opportunities offered by the concept of transactional modeling (TLM) to improve the models simulation of electronic architectures in the automotive domain. Our work has focused on the definition of a modeling technique allowing the improving of the tradeoff between simulation speed and results accuracy of architecture models in order to facilitate design space exploration. The proposed modeling technique aims to facilitate the creation of transactional models of distributed architectures for the design of hardware and software resources of these architectures. The interest of our work was illustrated through the study of a demonstrator defined in the CIFAER project. This study focuses on the modeling of this demonstrator architecture. The proposed modeling of this case of study has allowed to verify the significant simulation acceleration obtained by the application of the proposed technique.

Key Words

Distributed automotive electronic architecture, transaction level modeling, discrete event simulation, performance evaluation, design and architectural exploration.