République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A.Mira-Béjaia
Faculté des Sciences Exactes
Département d'Informatique
Laboratoire d'Informatique Médicale LIMED

# Thèse

Présentée par

Khaled Allem

Pour l'obtention du grade de

# Docteur en Sciences

Filière : Informatique
Option : Réseaux et Systèmes Distribués

Thème

# De UML/MARTE à SystemC/TLM : Une approche IDM pour la modélisation et l'analyse des SoCs reconfigurables.

Soutenue le 28/09/2025 devant le jury composé de :

| | | | |
|---|---|---|---|
| Pr. | Ahror Belaid | Université de Béjaia | Président du jury |
| Pr. | El-Bay Bourennane | Université de Bourgogne | Rapporteur |
| Pr. | Youcef Khelfaoui | Université de Béjaia | Co-Rapporteur |
| Pr. | Salim Chikhi | Université de Constantine-2 | Examinateur |
| Pr. | Kamel Messaoudi | Université de Souk Ahras | Examinateur |
| Dr. | Achour achroufene | Université de Béjaia | Examinateur |

Année Universitaire : 2024 - 2025

# Résumé

Face à la complexité croissante des processus de conception des systèmes sur puce (SoCs) dynamiquement et partiellement reconfigurables, il apparaît crucial d'élever le niveau d'abstraction utilisé pour modéliser ces systèmes. Un tel niveau d'abstraction permet de s'affranchir des contraintes liées aux détails d'implémentation bas niveau. Dans ce contexte, l'Ingénierie Dirigée par les Modèles (IDM) s'impose comme une approche méthodologique robuste, en fournissant un support formel pour la création, la transformation et le raffinement progressif de modèles structurés, en fonction des objectifs visés, à travers différents niveaux d'abstraction. Les modèles exprimés à un niveau élevé sont ainsi progressivement transformés en représentations plus concrètes, jusqu'à atteindre des modèles exécutables.

Ce travail s'inscrit dans cette démarche en se positionnant au niveau système (ESL), et plus spécifiquement au niveau transactionnel (TLM). Il propose un framework de modélisation permettant la génération automatique de code SystemC/TLM à partir de modèles UML/MARTE étendu, dans le respect des principes de l'IDM. Selon qu'il s'agisse d'un objectif de conception ou d'analyse, les modèles sont successivement raffinés à travers une chaîne de transformation définie, jusqu'à l'obtention d'un code source exploitable pour la simulation ou la synthèse de haut niveau. Dans le but d'atteindre un degré élevé d'expressivité dans la modélisation de la reconfiguration dynamique partielle (DPR), une approche inspirée de l'ingénierie logicielle a été adoptée, reposant sur une architecture de composants orientée services. Afin de remédier à l'absence de prise en charge native de la DPR tant dans le profil UML/MARTE que dans le langage SystemC, l'approche proposée s'appuie sur une extension de MARTE intégrant des constructions spécifiques, des stéréotypes dédiés ainsi que des valeurs étiquetées appropriées. Ces éléments sont regroupés au sein de trois sous-profils d'annotation complémentaires : MARTE4DPR, pour la modélisation de la reconfiguration dynamique ; MARTE4SCTLM, pour la modélisation des constructions SystemC au niveau transactionnels et MARTE4AF, pour l'intégration des concepts du patron de conception Abstract Factory dans la modélisation des composants reconfigurables. Une chaîne d'outils entièrement intégrée à l'environnement Eclipse a été développée conformément au flot de conception proposé afin de prendre en charge la création et l'édition des modèles, l'application des profils, ainsi que l'exécution des transformations nécessaires à la génération automatique de code. La validation expérimentale du framework proposé, réalisée à travers l'étude d'un crossover actif 3-voies reconfigurable, démontre que, bien qu'exigeant un effort de modélisation légèrement élevé, notre approche permet un gain significatif en temps de conception. Ce qui se traduit ainsi par une amélioration notable de la productivité, surpassant les méthodes manuelles classiques.

**Mots-clés :** Reconfiguration Dynamique Partielle, Système sur Puce, Ingénierie Dirigée par les Modèles, Composant Orienté Service, UML, MARTE, SystemC, TLM.

# Abstract

In light of the growing complexity inherent in the design processes of dynamically and partially reconfigurable systems-on-chip (SoCs), it becomes essential to raise the level of abstraction used to model such systems. A higher abstraction level enables designers to circumvent the constraints associated with low-level implementation details. Within this context, Model-Driven Engineering (MDE) emerges as a robust methodological approach, offering formal support for the creation, transformation, and progressive refinement of structured models across various abstraction levels, in alignment with specific design objectives. High-level models are thus gradually transformed into more concrete representations, ultimately resulting in executable models.

This work adheres to this methodology by operating at the Electronic System Level (ESL), and more specifically, at the Transaction Level Modeling (TLM) layer. It introduces a modeling framework that supports the automatic generation of SystemC/TLM code from extended UML/MARTE models, while strictly adhering to MDE principles. Depending on whether the objective is design-oriented or analytical, models are successively refined through a defined transformation chain, leading to the generation of source code suitable for high-level synthesis or simulation. To achieve a high degree of expressiveness in the modeling of Dynamic Partial Reconfiguration (DPR), the proposed approach adopts a software engineering-inspired methodology, based on a service-oriented component architecture. In response to the lack of native support for DPR in both the UML/MARTE profile and the SystemC language, the proposed solution builds on an extension of MARTE incorporating specific modeling constructs, dedicated stereotypes, and appropriate tagged values. These elements are organized into three complementary annotation sub-profiles : MARTE4DPR, dedicated to the modeling of dynamic reconfiguration ; MARTE4SCTLM, aimed at modeling SystemC constructs at the transactional level and MARTE4AF, which facilitates the integration of Abstract Factory design pattern concepts into the modeling of reconfigurable components. An end-to-end toolchain, fully integrated into the Eclipse environment, has been developed in accordance with the proposed design flow. This toolchain supports the creation and editing of models, the application of the aforementioned profiles, and the execution of the necessary transformations to enable automatic code generation. Experimental validation of the proposed framework, conducted through the case study of a three-way reconfigurable active crossover, demonstrates that, despite requiring a moderately higher modeling effort, this approach yields a significant reduction in design time, thereby leading to a notable increase in productivity when compared to conventional manual methods.

**Keywords :** Dynamic Partial Reconfiguration, System-on-Chip, Model-Driven Engineering, Service-Oriented Component, UML, MARTE, SystemC, TLM.

# Remerciements

Je rends grâce à Allah, le Tout-Puissant, pour m'avoir doté de la patience et de la force intérieure qui m'ont permis de persévérer tout au long de ces années de labeur. C'est par Sa volonté et Son assistance que j'ai pu surmonter les épreuves et mener à terme ce travail de thèse.

Je tiens tout d'abord à exprimer ma profonde reconnaissance à Monsieur El-Bay Bourennane, Directeur de cette thèse, pour sa confiance, sa disponibilité, ainsi que pour la richesse de ses conseils scientifiques et humains tout au long de ce travail. Son encadrement rigoureux, sa vision claire et sa bienveillance ont été essentiels à l'aboutissement de ce projet.

Mes remerciements sincères s'adressent également à Monsieur Youcef Khelfaoui, Co-directeur de cette thèse, pour son accompagnement précieux, ses orientations méthodologiques pertinentes et son soutien constant durant toutes ces années.

Je suis profondément honorée par la présence et l'investissement des membres du jury : je remercie Monsieur Ahror Belaid, Président du jury, pour l'intérêt manifesté à l'égard de ce travail et pour avoir accepté de présider la soutenance avec bienveillance et rigueur. Ma gratitude va à Monsieur Salim Chikhi, Monsieur Kamel Messaoudi et Monsieur Achour Achroufene , Examinateurs, pour leurs retours éclairés, leurs suggestions scientifiques fines, et le temps qu'ils ont généreusement consacré à l'analyse de ce travail.

Je tiens également à remercier l'ensemble de mes collègues, en particulier ceux du Laboratoire LIMED et du département d'Informatique.

A ma famille, j'adresse mes pensées les plus reconnaissantes, dont la présence affectueuse, les encouragements indéfectibles et la confiance jamais démentie m'ont donné la force de poursuivre jusqu'au bout.

Je tiens à adresser une pensée toute particulière à mon épouse, à mes deux princesses, Lilya et Nina, ainsi qu'à ma chère grand-mère. Je suis pleinement conscient des absences et des moments partagés que j'ai sacrifiés pour me consacrer à l'achèvement de cette thèse. Leur patience exemplaire, leur amour discret mais constant, et leur soutien indéfectible ont été pour moi une source essentielle de force et de sérénité tout au long de ce parcours exigeant. Qu'elles trouvent ici l'expression de ma profonde reconnaissance.

Enfin, je garde une pensée empreinte de tendresse et de profonde émotion pour ma mère Lili, partie trop tôt, et dont l'absence demeure douloureuse. Bien qu'elle ne soit plus parmi nous, sa mémoire, ses principes et l'exemple de sa droiture continuent de m'accompagner et de m'inspirer. C'est à elle que je dédie humblement ce travail, en hommage à tout ce qu'elle a été pour moi.

Béjaia, le 1er octobre 2025.

*A Lili qui aurait été si fière.*

# Table des matières

# Liste des figures

# Liste des tableaux

# Liste des listings

# Introduction

## Contexte de la thèse et objectifs

L'évolution technologique des circuits intégrés, caractérisée par une réduction continue des dimensions physiques et une augmentation significative de la densité fonctionnelle, a profondément transformé les processus de conception des systèmes électroniques. Toutefois, cette évolution rapide a donné naissance à un phénomène identifié par l'International Technology Roadmap for Semiconductors (ITRS) comme un gap de productivité, lequel traduit un décalage croissant entre la capacité d'intégration technologique (nombre de transistors/portes par circuit) et la productivité effective des concepteurs en électronique fournie par les outils EDA (Electronic Design Automation), tel que montré dans la figure 1. Bien que de nombreux obstacles technologiques aient été franchis, chaque avancée a complexifié davantage le travail des concepteurs et des outils EDA. Les premiers outils EDA visaient principalement à maximiser l'intégration dans une surface de silicium restreinte. Par la suite, la performance est devenue la métrique dominante. Les contraintes croissantes en matière de puissance, performance, coût et fiabilité, combinées à la complexité des conceptions, ont considérablement creusé le gap de productivité entre l'évolution rapide des technologies de fabrication et les capacités effectives des outils EDA.

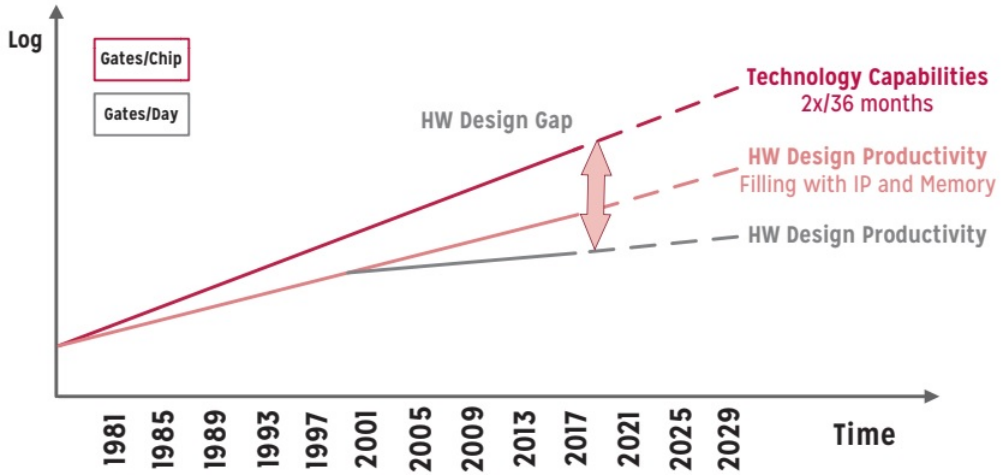



Figure 1 – *L'écart de productivité dans le processus de conception [Bahar et al., 2020].*

Face à ces défis, la conception des systèmes sur puce (SoC), en particulier les architectures reconfigurables, nécessite une remise en question fondamentale des méthodologies classiques basées sur des niveaux d'abstraction relativement bas (niveau RTL : Register Transfer Level) et ce en accord avec les exigences des outils EDA modernes. En effet, les SoCs reconfigurables, capables d'adapter dynamiquement leur architecture en fonction des besoins applicatifs, accroissent davantage la complexité de la conception, du fait des multiples possibilités d'optimisation et d'agencement architectural qu'ils impliquent.

L'élévation du niveau d'abstraction vers la conception au niveau système (Electronic System Level - ESL) apparaît comme une réponse appropriée pour combler ce gap de productivité. En adoptant une perspective plus abstraite, les concepteurs peuvent mieux appréhender la complexité globale des systèmes tout en s'affranchissant des détails techniques de bas niveau. Le langage SystemC, notamment à son niveau transactionnel (Transaction Level Modeling - TLM), constitue une cible privilégiée dans les flots de conception à haut niveau d'abstraction. Il permet de fournir des modèles avec des performances de simulation nettement supérieures à celles du RTL. En effet, La modélisation au niveau des transactions basée sur SystemC implique la communication entre les processus SystemC à l'aide d'appels de fonction, tout en respectant le principe de séparation de la communication et du calcul. En remplaçant tous les événements au niveau des broches par un seul appel de fonction, il est possible d'atteindre des facteurs d'accélération allant jusqu'à 10 000x [IEEE, 2023].

Dans ce contexte, notre travail s'inscrit dans une démarche visant à élever davantage le niveau d'abstraction des modèles qui se voient proches du domaine du problème et non de la solution tout en favorisant l'automatisation du processus de conception. L'adoption du profil UML/MARTE, au sein d'une approche fondée sur l'Ingénierie Dirigée par les Modèles (IDM), constitue en ce sens une méthodologie rigoureuse et structurante, apte à répondre aux défis posés par la complexité croissante des systèmes électroniques. La mise en place d'une chaîne de transformation automatisée assurant le raffinement progressif des modèles architecturaux exprimés en UML/MARTE, vers des représentations exécutables en SystemC/TLM, s'avère particulièrement pertinente dans le contexte des SoCs dynamiquement reconfigurables. Cette chaîne repose sur une démarche d'IDM, permettant de formaliser, analyser et transformer successivement les abstractions fonctionnelles, structurelles et temporelles du système. En prenant en charge les contraintes liées à la variabilité dynamique, elle assure une traçabilité entre les niveaux d'abstraction tout en facilitant la simulation comportementale, l'évaluation précoce des performances, et la génération de code compatible avec les outils de co-simulation matérielle/logicielle. Cette approche contribue ainsi à réduire le temps de conception et à garantir la cohérence entre modélisation abstraite et implémentation simulable.

## Contributions

Les principales contributions de ce travail peuvent être synthétisées comme suit :

En se positionnant au niveau système ESL et plus particulièrement au niveau transactionnel TLM, nous proposons un framework de modélisation architecturé en couches dans lequel les niveaux d'abstraction sont construit les uns sur les autres jusqu'à atteindre les niveaux techniques. En outre, un flot de conception à haut niveau d'abstraction, structuré selon le modèle en Y, et mis en œuvre par la combinaison des principes de l'IDM avec une approche de développement orientée modèle fondée sur le profil UML/MARTE est proposé. Ce flot intègre des techniques de transformation de modèles et repose sur l'utilisation d'outils spécialisés pour la modélisation et l'analyse, offrant ainsi un cadre formel et progressif pour la conception outillée des systèmes sur puce reconfigurables.

Afin d'assurer un haut niveau d'expressivité dans la modélisation de la reconfiguration dynamique, tout en garantissant une gestion efficace de la disponibilité au sein des composants reconfigurables, nous proposons un modèle de composant fondé sur un paradigme issu de l'ingénierie logicielle, s'appuyant sur une architecture de composants orientés services. Ce modèle vise à concilier les bénéfices des architectures à base de composants, en particulier la modularité et la séparation des préoccupations, avec

les principes du paradigme orienté services, tels que le faible couplage, la liaison dynamique et la découverte de services. Cette approche hybride permet ainsi de répondre de manière efficace aux exigences de flexibilité, de réutilisabilité et de reconfigurabilité, caractéristiques des systèmes dynamiquement reconfigurables.

Ce travail introduit un modèle de contrôle semi-distribué fondé sur l'intégration de la boucle de rétroaction MAPE-K (Monitor, Analyze, Plan, Execute – Knowledge), permettant une application décentralisée mais cohérente de la logique de reconfiguration. Cette approche vise à garantir l'autonomie locale des composants, en réduisant la dépendance à un point de contrôle centralisé, tout en assurant une coordination globale facilitée, favorisant la collaboration entre les contrôleurs locaux et déchargeant le coordinateur central.

Nous proposons un modèle de coût prédictif destiné à estimer, en amont, le temps de reconfiguration attendu. Ce modèle joue un rôle d'outil d'aide à la décision, en facilitant l'évaluation des performances potentielles du système dès les premières étapes de la conception. L'objectif est double : d'une part, éviter au concepteur d'engager un processus de mise en œuvre de la DPR lorsque les bénéfices attendus sont faibles ; d'autre part, permettre un ajustement précoce des paramètres de configuration en tenant compte des contraintes temporelles et structurelles identifiées.

La contribution majeure de ce travail réside dans l'extension ciblée du profil UML/-MARTE, visant à combler les lacunes en matière de modélisation des concepts fondamentaux relatifs à la reconfiguration dynamique et à l'architecture de composants orientés services. Par ailleurs, en réponse à l'absence de support natif de la reconfiguration dynamique dans le langage SystemC, une architecture d'un module dynamiquement reconfigurable est proposée. Celle-ci s'appuie sur le pattern de conception Abstract Factory et l'utilisation des appels à la fonction sc_spawn() afin de permettre l'instanciation dynamique des composants au moment de l'exécution. Ainsi, l'extension s'appuie sur l'introduction de constructions spécifiques, de stéréotypes dédiés et de valeurs étiquetées appropriées, référencées dans trois sous profils d'annotation, à savoir : MARTE4DPR, MARTE4SCTLM et MARTE4AF.

Concrètement, une chaîne d'outils entièrement intégrée à Eclipse est développée pour assurer la prise en charge de la création et de l'édition des modèles, de l'application des profils et de l'exécution des transformations pour la génération automatique de code. La validation expérimentale du framework proposé à travers un crossover actif 3 voies reconfigurable montre que, bien que notre framework requière un effort initial de modélisation légèrement supérieur, il permet, en contrepartie, un gain substantiel en termes de temps de conception et par conséquent une meilleure productivité, surpassant ainsi les approches manuelles classiques.

## Plan de la thèse

Le présent manuscrit est structuré en cinq chapitres comme suit :
Le premier chapitre expose les éléments nécessaires à la modélisation des SoCs reconfigurables. Il commence par une présentation des architectures reconfigurables et leur classification avec un intérêt particulier pour les systèmes sur puces et les FPGAs. Le concept de reconfiguration dynamique partielle est présenté par la suite, le mécanisme de contrôle et le processus de reconfiguration, les avantages et le coût de la reconfiguration ainsi que le flot de conception des systèmes reconfigurable y sont détaillés. Dans un second temps, la modélisation des SoCs à différents niveaux d'abstraction est abordée avant de présenter un aperçu sur le langage SystemC et les concepts de modélisations au niveau transactionnel, éléments essentiels à la conception et à la simulation de systèmes complexes à un haut niveau d'abstraction.

Dans le deuxième chapitre nous abordons les principes généraux de l'IDM, en l'occurrence : le système, le modèle et le métamodèle ainsi que l'approche MDA, ses modèles et son cycle de développement. Par la suite, le principe de transformation de modèles est présenté avec sa taxonomie et ses approches. Les langages dédiés à la modélisation, leurs types, leurs limites ainsi que les profils UML sont également exposés. Une partie substantielle du chapitre est consacrée à la reconfiguration dynamique en tant que préoccupation transverse en établissant le rapprochement entre adaptation logicielle et reconfiguration matérielle. Les différents paradigmes de conception des systèmes logiciels dynamiquement adaptatifs sont également étudiés, notamment l'ingénierie logicielle basée composants, l'ingénierie logicielle orientée services et les composants orientés services.

Le troisième chapitre dresse un état de l'art des approches de conception des systèmes reconfigurables. Il commence par une présentation des flots de conception proposés par les fournisseurs industriels, suivie d'une revue des outils de développement issus de la recherche académique. Dans un second temps, le chapitre examine les travaux portant sur la modélisation de la reconfiguration dynamique à un haut niveau d'abstraction, ainsi que sur la génération automatique de code. Ces travaux ont été classés selon le niveau d'abstraction adopté : d'une part, les approches reposant sur le langage SystemC, et d'autre part, celles fondées sur UML et ses profils pour la modélisation à haut niveau. Le chapitre se termine par une synthèse comparative des différents travaux étudiés, mettant en évidence leurs réponses à des critères pertinents dans le cadre de la modélisation des systèmes reconfigurables.

Le quatrième chapitre présente le framework de modélisation proposé, en détaillant son architecture en couches ainsi que le flot de conception associé. Au début, une définition approfondie du domaine métier est fournie, incluant notamment le modèle du composant reconfigurable proposé, le patron d'interaction ainsi que la modélisation de l'analyse temporelle. Par la suite, les extensions apportées au profil MARTE sont explicitées à travers les nouveaux profils MARTE4DPR, MARTE4SCTLM et MARTE4AF intégrant respectivement à MARTE des concepts liés à la reconfiguration dynamique, au langage SystemC au niveau transactionnel et au pattern de conception Abstract Factory. Enfin, le chapitre présente les différentes transformations de modèles mises en œuvre, qu'elles soient de type modèle-à-modèle (M2M) ou modèle-à-texte (M2T).

Le cinquième chapitre est consacré à la validation expérimentale du framework proposé, à travers une étude de cas portant sur un crossover actif 3 voies reconfigurable. L'application du framework débute par la modélisation du système cible à plusieurs niveaux d'abstraction, incluant la définition des modèles d'application, de plateforme d'exécution, de mapping, ainsi que de simulation SystemC/TLM. Une analyse d'ordonnançabilité basée sur les modèles est également réalisée. Finalement, le chapitre propose une analyse des performances du framework afin d'en apprécier l'applicabilité et la pertinence.

Enfin, le chapitre consacré aux conclusions et perspectives rappelle succinctement les objectifs fixés au début de cette thèse, dresse un bilan synthétique des travaux effectués, et met en évidence les principales contributions, ainsi que les limites identifiées. Il propose également les perspectives de recherche qui découlent naturellement des résultats obtenus.

# Conventions et typographie

Afin de garantir la clarté, la cohérence et l'uniformité tout au long du manuscrit et conformément à la spécification de MARTE et de SystemC, les conventions et typographie suivantes ont été adoptées :

- Les noms des concepts des domaines métiers sont donnés en anglais.
- Les noms des métaclasses et des stéréotypes sont écrits en upper camel case (ex. : ClientServerPort ).
- Les noms des stéréotypes sont présentés entre guillemets français (ex. : « ReconfigurableUnit »), remplaçant les chevrons habituellement utilisés dans la spécification de MARTE.
- Les noms des métaclasses et des stéréotypes définis dans MARTE sont donnés en gras (ex. : **ClientServerPort** et **« ClientServerPort »**).
- Les noms des attributs des métaclasses et des stéréotypes sont donnés en italique et en lower camel case (ex. : *nbPins*).
- Les attributs booléens des métaclasses et des stéréotypes sont systématiquement nommés en commençant par « is » (ex. : *isDynamic*).
- Les noms des métaclasses et des stéréotypes du domaine SystemC/TLM sont écrits en snake case conformément au manuel de référence du langage SystemC (ex. Sc_Prim_Channel et TLM_Bw_Transport_If).
- Les noms des métaclasses et des stéréotypes du domaine SystemC commencent toujours par « Sc » (ex. : Sc_Module et « Sc_Module »).
- Les noms des métaclasses et des stéréotypes du domaine TLM commencent toujours par « TLM » (ex. : TLM_Target_Socket et « TLM_Target_Socket »).

# 1

# Eléments pour la modélisation des SoCs reconfigurables

Sommaire

## 1.1 Introduction

L'essor rapide des systèmes embarqués et des applications à forte intensité de calcul en temps réel a favorisé l'émergence de nouvelles architectures matérielles capables de concilier, de manière efficace, les exigences croissantes en termes de performance, de flexibilité fonctionnelle, de maîtrise énergétique, ainsi que de réduction des délais de mise sur le marché. Ainsi, les systèmes sur puce reconfigurables, reposant sur des circuits logiques programmables de type FPGA (Field Programmable Gate Arrays), s'imposent progressivement comme une solution technologique privilégiée dans de nombreux domaines d'application. Dans ce contexte, la modélisation des systèmes à base de FPGA joue un rôle crucial pour maîtriser la complexité croissante des conceptions matérielles. Elle permet de représenter de manière abstraite et formelle le comportement fonctionnel et temporel du système, facilitant ainsi l'exploration de l'espace de conception, la validation, l'optimisation et l'analyse des performances dès les premières phases du cycle du développement.

Le langage SystemC s'impose comme un outil de choix pour la modélisation de haut niveau des architectures matérielles complexes. Basé sur le langage C++, SystemC permet de décrire à la fois des comportements matériels synchrones et des interactions systèmes, tout en supportant différents niveaux d'abstraction.

Le présent chapitre traite en premier lieu les architectures reconfigurables et leur classification, avec un accent particulier sur les systèmes sur puce et les circuits logiques programmables de type FPGA. Il examine ensuite divers aspects relatifs à la reconfiguration dynamique partielle, notamment le mécanisme de contrôle, le processus de reconfiguration et le flot de conception associé. La modélisation des systèmes sur puce à différents niveaux d'abstraction est également abordée. Enfin, avant de conclure, un aperçu est consacré à l'utilisation du langage SystemC pour la modélisation des systèmes matériels, en mettant en évidence les concepts de la modélisation au niveau transactionnel (TLM).

## 1.2 Les architectures reconfigurables

Une architecture reconfigurable représente un paradigme informatique hybride, alliant la flexibilité des solutions logicielles à la performance intrinsèque des systèmes matériels. Elle s'appuie sur une matrice matérielle configurable, spécifiquement conçue pour exécuter avec une grande efficacité des traitements intensifs en données, à l'instar du traitement d'images ou de la reconnaissance de motifs, en adoptant temporairement le comportement d'un circuit spécialisé. Une fois la tâche réalisée, cette infrastructure peut être reconfigurée dynamiquement afin d'accomplir une autre fonction, sans modification physique du matériel. Ce mécanisme confère à l'architecture reconfigurable une polyvalence remarquable, en lui permettant d'atteindre un compromis optimal entre adaptabilité fonctionnelle propre aux solutions logicielles et vitesse d'exécution caractéristique des systèmes matériels [Moghaddam et al., 2017].

Tel que montré dans la figure 1.1, les processeurs à usage général (General Purpose Processors - GPPs), bien qu'offrant une grande programmabilité, demeurent relativement limités en termes de performances, notamment en ce qui concerne le temps d'exécution, lorsqu'ils sont comparés aux circuits intégrés spécifiques à une application (Application Specific Integrated Circuits – ASICs). Ces derniers sont conçus pour exécuter des tâches précises avec un degré élevé de parallélisme, permettant ainsi une implémentation particulièrement efficace en matière de performance. En effet, un ASIC peut être optimisé en intégrant uniquement les unités fonctionnelles nécessaires à une application cible, ce qui le rend à la fois rapide, compact et énergétiquement efficient. L'évo-

lution des technologies de fabrication a considérablement augmenté la complexité des ASICs, faisant passer leur capacité de quelques milliers à plusieurs millions de portes logiques, augmentant ainsi leur potentiel fonctionnel. Toutefois, malgré leurs avantages, les ASICs présentent des inconvénients majeurs dans le contexte des systèmes embarqués : des coûts de développement non récurrents (Non Recurring Engineering - NRE) très élevés et des délais de mise sur le marché prolongés, les rendant économiquement viables uniquement pour des productions à très grande échelle.

Dans cette perspective, les architectures de calcul reconfigurables, telles que les FPGA (Field Programmable Gate Arrays) et les CGRA (Coarse Grained Reconfigurable Architectures), émergent comme des alternatives intermédiaires prometteuses, combinant à la fois performance et flexibilité. La performance résulte du parallélisme structurel de ces architectures, tandis que la flexibilité provient de leur capacité à être reconfigurées pour exécuter différentes tâches. Les FPGA offrent une reconfiguration fine, au niveau des portes logiques, alors que les CGRA permettent une reconfiguration à grain plus grossier, généralement au niveau du transfert de registre. L'ASIP (Application-Specific Instruction-set Processor) conserve la programmabilité d'un processeur tout en intégrant des optimisations matérielles ciblées, ce qui améliore son efficacité pour une classe d'applications donnée, au détriment d'une flexibilité plus réduite que celle d'un DSP (Digital Signal Processor) ou d'un GPP (General Purpose Processor).

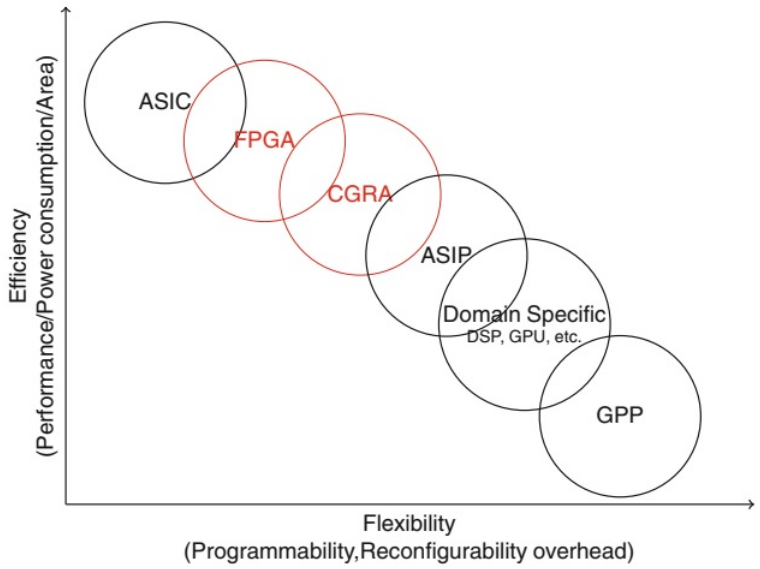



FIGURE 1.1 – *Comparaison des environnements matériels d'implémentation [Moghaddam et al., 2017].*

### 1.2.1 Classification des architecture reconfigurables

[Compton et Hauck, 2002] classifient les architectures reconfigurables selon la granularité de la structure interne des éléments reconfigurables, le degré de couplage entre le matériel reconfigurable et un microprocesseur traditionnel, le type du réseau d'interconnexion et le type de la reconfiguration.

**Granularité** : La granularité d'un système reconfigurable est déterminée par la taille de sa plus petite unité fonctionnelle. Une granularité fine signifie que de petites parties du système peuvent être reconfigurées individuellement, ce qui offre une flexibilité et une efficacité accrues. Comme exemple, la série AMD Virtex [1] ou la série Intel Stra-

1. https ://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html

tix [2]. En revanche, une granularité grossière implique que des parties plus grandes du système doivent être reconfigurées en même temps, ce qui peut limiter la flexibilité et augmenter les temps de reconfiguration. Le dispositif Matrix de la société QuickSilver Technology en est un exemple. Dans [Zamacola et al., 2020], les auteurs ont proposé une classification des systèmes reconfigurables basée sur la granularité des zones reconfigurables, telle que illustré dans la figure 1.2. Premièrement, la reconfiguration à gros grain représente la granularité la plus simple, visant à échanger des accélérateurs monolithiques couvrant généralement de vastes régions de ressources reconfigurables. Deuxièmement, la reconfiguration à grain moyen est employée pour reconfigurer des accélérateurs modulaires en modifiant les modules qui varient entre deux versions successives d'accélérateurs. Enfin, la reconfiguration à grain fin se concentre sur le changement rapide de primitives spécifiques du FPGA, telles que les LUTs (Look-Up Tables) ou les FFs (Flip-Flops).

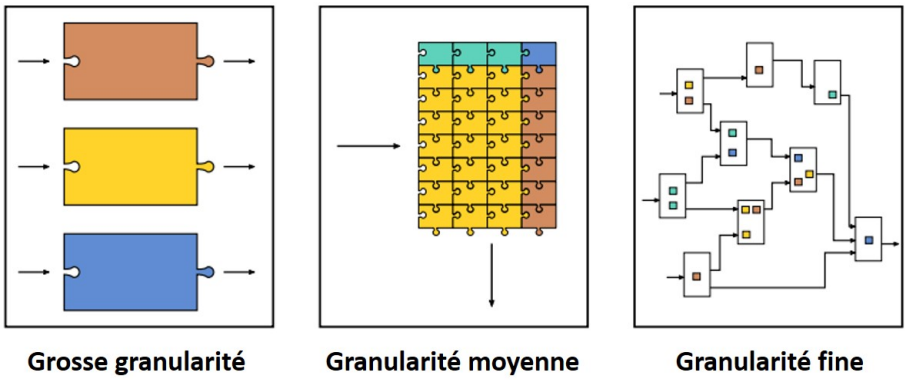



FIGURE 1.2 – *Classification des architectures reconfigurables selon leur granularité [Zamacola et al., 2020].*

**Degré de couplage** : Pour un système hybride qui intègre à la fois un microprocesseur et une logique reconfigurable, il existe plusieurs façons de coupler ces composants matériels. La logique reconfigurable peut être incorporée à l'intérieur d'un processeur hôte. Ce type offre un environnement de programmation classique avec l'ajout de matériel reconfigurable pouvant être modifié pour exécuter des opérations personnalisées. Dans ce cas, l'unité reconfigurable agit comme un sous-système auxiliaire du processeur principal et ne peut pas agir de manière indépendante. Une unité reconfigurable peut être utilisée comme coprocesseur. Dans ce cas, le coprocesseur, qui est généralement plus grand qu'une unité fonctionnelle, peut effectuer des calculs de manière indépendante, sans la supervision constante du processeur hôte. Cependant, le processeur déclenche toujours l'unité reconfigurable et envoie les données initiales pour démarrer une fonction sur le matériel reconfigurable ou fournit des informations sur l'emplacement de ces données en mémoire. Une unité de traitement reconfigurable conjointe agit comme un processeur supplémentaire dans un cadre multiprocesseur. En conséquence, il existe un délai plus élevé dans la correspondance entre le processeur hôte et le matériel reconfigurable. Par exemple, lors de la transformation des informations de configuration ou des données d'entrée et de sortie. Ce type d'architecture de communication offre une connexion à large bande passante entre le CPU et le FPGA. Enfin, une unité matérielle reconfigurable autonome peut être couplée à un processeur hôte via leurs connexions périphériques. Dans ce type d'architecture de connexion, l'unité autonome basée sur un FPGA aura occasionnellement des interactions avec le CPU. Ce modèle fonctionne de manière analogue aux stations de travail au sein d'un réseau,

2. https ://www.intel.fr/content/www/fr/fr/products/details/fpga/stratix/10.html

où le traitement des données constitue l'essentiel du temps d'exécution, réduisant ainsi considérablement le besoin d'une communication permanente avec le CPU.
[Todman et al., 2005] ont ajouté un modèle supplémentaire consistant à embarquer un processeur softcore ou hardcore au sein même d'une fabrique reconfigurable.
La latence de communication peut représenter le principal inconvénient pour les applications traitant un grand nombre de transactions, pouvant dans certains cas réduire ou annuler complètement les avantages d'accélération attendus avec l'utilisation de ce type de matériel reconfigurable.

**Type du réseau d'interconnexion** : Un réseau d'interconnexion fixe est prédéfini et ne peut pas être modifié après la fabrication du dispositif reconfigurable. Il est utilisé pour connecter les éléments de logique programmables entre eux et avec d'autres composants du système. Contrairement au réseau fixe, un réseau reconfigurable peut être modifié dynamiquement pour répondre aux besoins changeants de l'application. Il permet une flexibilité accrue en permettant aux éléments de logique programmables de se connecter différemment en fonction des exigences de l'application.

**Type de la reconfiguration** : Le processus de reconfiguration peut se manifester sous deux formes : statique (passive) ou dynamique (active). Ces variations, qu'elles soient complètes ou partielles, dépendent des caractéristiques de l'architecture reconfigurable et des spécifications de conception.

Une autre classification dite *Olymp* présentée dans [Radunovic et Milutinovic, 1998] et [Radunovic, 1999] décrit une architecture reconfigurable en termes de :

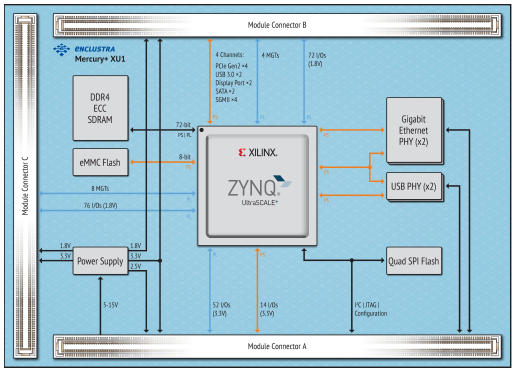
— Objectif de la reconfiguration : améliorer les performances en matière de vitesse d'exécution ou assurer une tolérance aux pannes.
— Granularité : fine, moyenne ou à gros grains.
— Intégration : lâchement couplée ou étroitement couplée.
— Reconfigurabilité du réseau d'interconnexion externe : architectures avec un réseau externe reconfigurable ou architectures avec un réseau externe fixe.

### 1.2.2 Les Systèmes sur puce

Un produit dont une partie ou la totalité de la fonctionnalité est réalisée sous forme de système VLSI (Very Large Scale Integration) sur un substrat de silicium est appelé système sur puce ou SoC (System on Chip). Lorsqu'un SoC est associé à un système logiciel utilisant un système d'exploitation en temps réel, il est alors appelé système embarqué [V. S. Chakravarthi, 2023]. Un SoC est généralement composé des éléments suivants :

— Processeurs et sous-systèmes de processeurs : Ce sont les unités de calcul principales du SoC, comprenant souvent des cœurs de processeurs, des unités de traitement graphique (Graphics Processing Unit - GPU) et d'autres unités de traitement spécialisées.
— Mémoires intégrées : Cela comprend la mémoire vive (Random Access Memory - RAM), la mémoire cache et parfois la mémoire morte (Read-Only Memory - ROM), ainsi que d'autres types de mémoires spécifiques au SoC.
— Sous-systèmes périphériques : Ces sous-systèmes gèrent les entrées/sorties et les interactions avec les périphériques externes tels que les ports USB, les ports HDMI, les ports Ethernet, etc.
— Cœurs de communication standard : Il s'agit des composants permettant la communication entre les différents éléments du SoC ainsi qu'avec des périphériques externes. Cela peut inclure des cœurs de communication tels que UART (Univer-


(a) *Vue de face du Mercury+ XU1*




(b) *Diagramme de blocs du Mercury+ XU1.*

FIGURE 1.3 – *Mercury+ XU1 : un exemple de système sur puce.*

sal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), etc.

— Contrôleurs de périphériques et de mémoire : Ces contrôleurs gèrent l'accès aux périphériques externes et à la mémoire intégrée, garantissant une communication efficace et synchronisée.

— Blocs fonctionnels spécifiques à l'application : Ces blocs sont conçus pour répondre à des besoins spécifiques de l'application pour laquelle le SoC est destiné. Cela peut inclure des blocs de traitement de données, des fonctions de couche physique dans les processeurs de communication, des cœurs de traitement du signal, etc.

— Logique supplémentaire : Cette logique peut inclure des accélérateurs matériel, des interfaces de communication spécifiques à l'application, des fonctions de gestion de l'énergie, etc.

Typiquement, ces éléments sont reliés les uns aux autres grâce à un réseau sur puce (Network-on-Chip - NOC) ou à un bus d'interconnexion, garantissant ainsi une communication fluide et efficace entre les divers composants du SoC.

La figure 1.3 montre le Mercury+ XU1 Zynq UltraScale+ MPSoC, un SoC hautement intégré développé par la société Enclustra [3]. Il est basé sur la famille de produits Zynq UltraScale+ de Xilinx, qui combine des cœurs de processeur Arm Cortex-A53 ou Cortex-A72 avec des matrices de logique programmable FPGA.

#### 1.2.2.1 Propriétés des SoCs

Les SoCs sont conçus pour optimiser le rapports entre fonctionnalités, encombrement, performances et consommation d'énergie. Ils doivent généralement avoir plusieurs propriétés pour être considérés efficaces et fonctionnels. Ces propriétés peuvent varier en fonction de l'application spécifique du SoC, mais certaines propriétés générales comprennent :

— Fiabilité : Un SoC fiable doit répondre de manière cohérente et prévisible aux exigences de l'utilisateur dans divers scénarios d'application et conditions d'utilisation.

— Disponibilité : La disponibilité d'un SoC fait référence à sa capacité à être opérationnel et à répondre aux demandes de l'utilisateur dans les applications prévues, avec un minimum de temps d'arrêt ou d'interruption.

— Scalabilité : Un SoC scalable peut s'adapter à différentes charges de travail et à des demandes croissantes en ressources, permettant ainsi une évolutivité facile en fonction des besoins changeants de l'application.

3. https ://www.enclustra.com/en/home/

— Flexibilité : Une architecture SoC flexible peut être configurée pour prendre en charge différentes fonctionnalités et applications, et peut être adaptée aux besoins spécifiques de diverses industries ou marchés.
— Maintenabilité : La maintenabilité d'un SoC fait référence à sa facilité de maintenance et de gestion tout au long de son cycle de vie, ce qui inclut la capacité à effectuer des mises à jour logicielles et matérielles, ainsi qu'à résoudre les problèmes éventuels de manière efficace.
— Sécurité : Un SoC sécurisé intègre des mécanismes de protection des données et des systèmes contre les menaces potentielles telles que les cyberattaques et les violations de la vie privée.
— Efficacité énergétique : Un SoC efficace sur le plan énergétique optimise l'utilisation des ressources énergétiques pour minimiser la consommation d'énergie tout en maintenant des performances élevées, ce qui est particulièrement important pour les appareils mobiles et les dispositifs alimentés par batterie.

En intégrant ces facteurs dans la conception d'une architecture SoC, on peut créer un système robuste, flexible, sécurisé, efficace sur le plan énergétique et facilement maintenable, répondant ainsi aux besoins variés des applications modernes.

#### 1.2.2.2 Avantages et inconvénients de l'utilisation des SoCs

Les avantages et les inconvénients mettent en évidence les raisons pour lesquelles la technologie des systèmes sur puces est la plus recherchée par les industries travaillant sur la logique embarquée. En consolidant plusieurs composants sur une seule puce en silicium et en créant un produit électronique tout-en-un, les entreprises s'attendent à bénéficier considérablement en termes de fabrication, notamment sur les marchés où le prix et la taille des appareils sont d'une importance capitale. Selon [V. S. Chakravarthi, 2023], les avantages des systèmes sur puces se manifestent à travers les aspects suivants :

— Intégration complète : Les SoC intègrent plusieurs composants matériels et logiciels sur une seule puce, réduisant ainsi la complexité du système.
— Espace optimisé : Les SoC occupent moins d'espace que plusieurs composants discrets, permettant la conception de dispositifs plus compacts.
— Efficacité énergétique : L'intégration des composants sur une seule puce permet une meilleure optimisation de la consommation d'énergie.
— Coût réduit : les concepteurs de SoCs peuvent incorporer des blocs de conception existants, tels que des propriétés intellectuelles (Intellectual Property - IP) réutilisables, des cœurs de processeur, des interfaces standard, etc. Cette réutilisation permet de réduire les coûts de développement, de diminuer le temps de mise sur le marché et d'améliorer la fiabilité, tout en assurant une compatibilité et une interopérabilité avec d'autres systèmes.
— Fiabilité améliorée : Les SoCs ont moins de connexions et sont donc plus fiables qu'un système multipartite connecté via un substrat.
— Hautes performances : Les SoC peuvent offrir des performances optimisées en intégrant des composants spécifiquement conçus pour fonctionner ensemble de manière efficace.

Les inconvénients des systèmes sur puces comprennent :

— Coût élevé de développement : La conception et le développement de SoC personnalisés peuvent être coûteux en raison des ressources humaines, matérielles et temporelles nécessaires.
— Nécessité de réussir dès la première fois : Les SoCs personnalisés ont peu de marge d'erreur car toute défaillance ou erreur peut avoir un impact significatif sur le produit final, ce qui peut augmenter les risques et les coûts.

— Cycles de développement prolongés : Si les IPs nécessaires ne sont pas disponibles ou si des retards surviennent dans le processus de développement, cela peut entraîner des cycles de développement plus longs, ce qui peut retarder la mise sur le marché du produit.
— Besoin de ressources hautement qualifiées : La conception et le développement de SoCs nécessitent une expertise technique et des compétences spécialisées dans les domaines de l'architecture, de la conception de circuits intégrés et du développement logiciel, ce qui peut être difficile à trouver et à maintenir.

### 1.2.3 Les circuits FPGAs

Les FPGAs (Field Programmable Gate Arrays) sont des dispositifs semi-conducteurs basés sur une matrice de blocs logiques configurables reliés par des interconnexions programmables. Leur particularité réside dans leur reconfigurabilité. Les FPGA peuvent être reprogrammés après leur fabrication pour répondre aux exigences des applications ou fonctionnalités souhaitées.

Le premier FPGA commercial, le XC2064, a été lancé et livré en 1985 par la société technologique américaine Xilinx. Doté d'une capacité maximale de 800 portes logiques, ce dispositif utilisait une technologie à l'aluminium de 2 µm avec 2 niveaux de métallisation. Xilinx a été suivi peu après par son principal concurrent, Altera, qui a introduit en 1992 le FPGA FLEX 8000, offrant une capacité maximale de 15 000 portes logiques. Au fil du temps, les circuits FPGAs ont continué à évoluer avec l'avancement technologique, et de nombreux fabricants dont Xilinx, Altera, Actel, Lattice et Motorola, ont rejoint le marché. Cependant, Xilinx (rachetée par AMD en 2022) et Altera (rachetée par Intel en 2015) demeurent les principaux acteurs de ce secteur.

À partir des années 2000, les FPGAs sont devenus plus complexes, atteignant jusqu'à 6,8 milliards de transistors. Les fréquences d'horloge dépassent désormais le gigahertz, et les technologies les plus avancées atteignent 16 nm, comme le Xilinx Virtex UltraScale+. Ainsi, les FPGA permettent aujourd'hui de concevoir des systèmes sur puce complets pour des applications très complexes. Cette approche est préférable à la réalisation en ASIC, où les coûts et les risques sont plus élevés, sans la garantie de parvenir à concevoir un circuit entièrement fonctionnel à la fin du processus.

En raison de leur capacité intrinsèque de configuration et de leur coût de développement relativement bas, l'importance des circuits FPGA dans différents domaines de l'ingénierie a rapidement augmenté. Les FPGAs offrent des avantages considérables pour un large éventail d'applications, notamment dans les secteurs de l'imagerie médicale, de l'aérospatiale, de la défense, de l'automobile, ainsi que dans le traitement vidéo et le traitement d'images.

#### 1.2.3.1 Structure générale d'un circuit FPGA

Un FPGA se compose d'une matrice de blocs logiques configurables entourée de blocs d'entrée/sortie, l'ensemble est connecté par des ressources de routage programmables. Les blocs logiques sont interconnectés par une hiérarchie d'interconnexions reconfigurables et sont généralement composés de LUTs (Look Up Tables) pour réaliser des fonctions élémentaires et des bascules flip-flop pour la mémorisation. Dans le contexte de l'optimisation de la surface occupée et des délais, il a été déterminé que le nombre idéal d'entrées pour les LUTs est de quatre. Toutefois, ce nombre peut varier en fonction de l'application spécifique visée par le fabricant [V. S. Chakravarthi, 2023]. Comme le montre la figure 1.4, à cette architecture très régulière s'ajoutent depuis quelques années différents éléments permettant de rendre le système implanté

dans le FPGA beaucoup plus efficace en termes de performance ou d'énergie consommée. On trouve par exemple : des blocs mémoire RAM de tailles diverses, des blocs arithmétiques optimisés pour les principales fonctions du traitement du signal (Digital Signal Processor - DSP), des blocs de génération d'horloges programmables par l'utilisateur, des blocs contenant un périphérique spécifique et de plus en plus de processeurs enfouis dans le FPGA.

Le concepteur implémente la fonctionnalité requise à l'aide d'un langage de description matérielle (Hardware Description Language - HDL) tel que Verilog/VHDL, ou il pourrait aussi utiliser une synthèse de haut niveau pour convertir un langage de programmation plus abstrait (comme C++ ou OpenCL) en HDL. Par la suite, le design HDL est compilé via un processus complexe de conception assistée par ordinateur pour générer un fichier de configuration (bitstream), qui est ensuite utilisé pour programmer toutes les cellules SRAM (Static RAM) de configuration du FPGA.

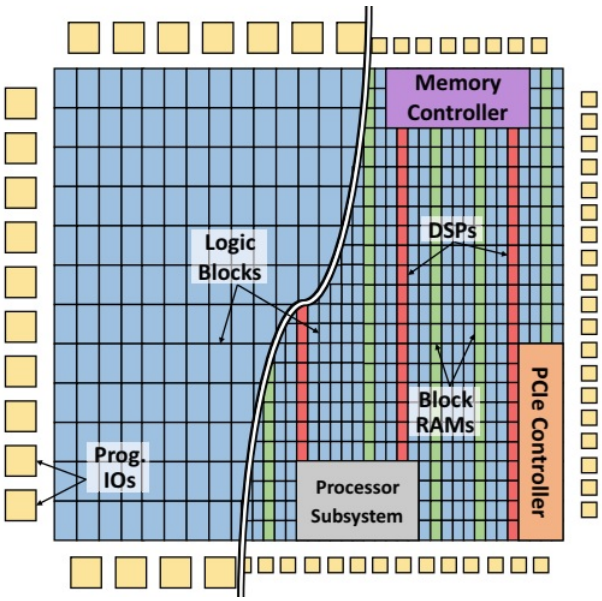



FIGURE 1.4 – *Première architecture FPGA avec logique programmable et E/S comparée à une architecture FPGA hétérogène moderne avec RAM, DSP et autres blocs matériels [Boutros et Betz, 2022].*

#### 1.2.3.2 Les Types de reconfiguration dans les FPGA

Les différents types de reconfigurations dans les FPGA peuvent être classés selon trois axes principaux : où, quand et comment la reconfiguration a lieu [Quadri, 2010].

**Où la reconfiguration a lieu :** Selon l'origine de l'initialisation et du contrôle, la reconfiguration peut être soit externe, soit interne au FPGA. Dans le cas d'une reconfiguration interne, un contrôleur intégré gère le processus de reconfiguration, ce qui réduit les délais causés par la communication avec un contrôleur externe. Par conséquent, cela limite l'impact du temps de reconfiguration sur les performances globales du système.

**Quand la reconfiguration a lieu :** La reconfiguration peut être classée en deux catégories : statique ou dynamique. Dans le premier cas, le FPGA doit être inactif pour être reconfiguré, tandis que dans le second, le FPGA est reconfiguré durant l'exécution. La reconfiguration dynamique offre une flexibilité et une adaptabilité accrues, tout en évitant les délais supplémentaires associés à la désactivation et à la réinitialisation en termes de temps d'exécution.

**Comment la reconfiguration a lieu :** la reconfiguration peut être complète ou partielle. Dans la reconfiguration complète, un bitstream décrivant l'intégralité du FPGA est chargé même si la reconfiguration n'apporte que des modifications infimes sur le système, ce qui constitue une perte de temps conséquente ayant un impact négatif sur les performances du système. La reconfiguration partielle permet de remédier à ce problème en définissant une ou plusieurs zones indépendantes du reste du design, pouvant être modifiées à n'importe quel moment sans interrompre l'exécution du reste du système. La reconfiguration partielle offre donc une meilleure gestion de la surface du FPGA en apportant des modifications en fonction des besoins de l'application. Il est également bon de souligner que les termes reconfiguration partielle et reconfiguration dynamique ont souvent été utilisés indifféremment dans la littérature spécialisée.

[Compton et Hauck, 2002] proposent une classification basée sur le mécanisme de gestion de la mémoire du FPGA et qui permet de distinguer trois modèles de reconfiguration : la reconfiguration à contexte unique, la reconfiguration multi-contexte et la reconfiguration partielle, comme illustrer dans la figure 1.5.

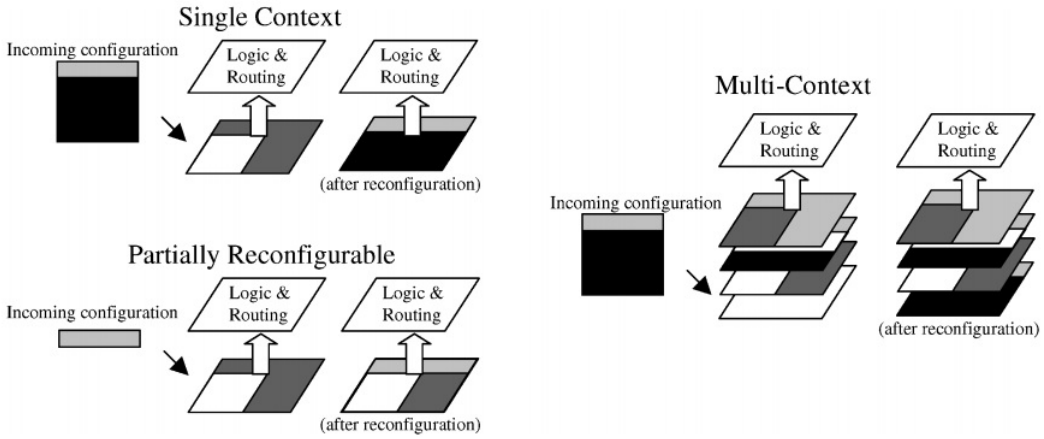



FIGURE 1.5 – *Les différents modèles de base d'un FPGA reconfigurable [Compton et Hauck, 2002].*

## 1.3 La reconfiguration dynamique partielle

La reconfiguration dynamique partielle (Dynamic Partial Reconfiguration - DPR) est une fonctionnalité des FPGA modernes qui permet la modification en temps réel d'un FPGA durant l'exécution. Des bitstreams partiels peuvent être chargés dans le FPGA pour reconfigurer des régions sélectionnées sans affecter le fonctionnement des autres parties du dispositif ; ce qui permet le partage temporel des ressources matérielles entre des tâches mutuellement exclusives. Conformément à la figure 1.6, un système qui utilise la DPR peut être conceptuellement divisé en deux parties principales : une partie statique configurée au démarrage avec un bitstream complet, et une partie reconfigurable à l'exécution, qui peut être composée de plusieurs régions reconfigurables indépendantes. Chacune de ces régions peut être reconfigurée plusieurs fois pendant l'exécution avec différents bitstreams partiels sans interférer avec le fonctionnement de la partie statique [Pezzarossa et al., 2018]. Le contrôleur de configuration est responsable de la gestion du processus de configuration globale du FPGA. Il contrôle le flux de données de configuration vers les différents éléments logiques programmables internes du FPGA en fonction du fichier de configuration fourni.

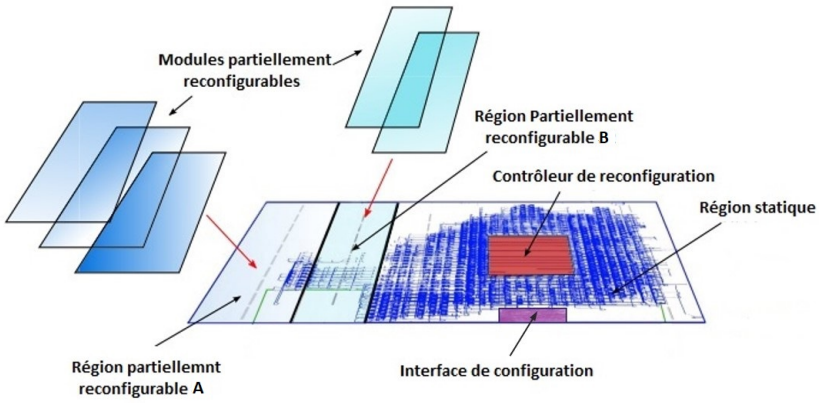



FIGURE 1.6 – *Concepts de base de la RDP.*

Comme illustré dans la figure 1.7, dans un téléphone mobile et pour des fonctions mutuellement exclusives, il est possible de partager les mêmes ressources en reconfigurant partiellement un module pour le dispositif actuellement nécessaire, le FPGA apparaîtra essentiellement plus grand que sa taille physique.

Ainsi, la reconfiguration dynamique partielle offre une solution flexible, efficace et évolutive pour répondre notamment aux besoins changeants des applications en temps réel, s'adapter aux changements environnementaux, surmonter les limitations de ressources, répondre aux exigences de qualité de service et respecter les contraintes de consommation d'énergie.

Un système dynamiquement reconfigurable doit posséder quelques propriétés spécifiques afin de pouvoir exécuter efficacement ses tâches. En voici quelques-unes :

— La granularité de la reconfiguration, permettant d'ajuster dynamiquement des parties allant de simples LUTs jusqu'à la reconfiguration de l'ensemble de la puce. Une granularité importante peut engendrer des surcoûts dans la zone partiellement reconfigurable, tandis qu'une granularité fine offre plus de flexibilité, bien qu'elle puisse entraîner des coûts architecturaux significatifs.
— La prise en charge de la relocalisation pendant l'exécution facilite l'utilisation d'un même bitstream pour configurer un circuit à différents emplacements sur le FPGA, similaire au mapping de mémoire virtuelle en mémoire physique dans les environnements logiciels.
— Un temps de reconfiguration minimal afin d'éviter de compromettre les performances du système. Avec des temps de reconfiguration prolongés, même si le reste du système fonctionne, l'attente du chargement d'un accélérateur par exemple risque d'impacter les performances globales. Une reconfiguration plus rapide permettrait de charger et décharger les accélérateurs aussi rapidement que le basculement des tâches dans les processeurs multi-cœurs.
— L'opération de reconfiguration devrait être transparente pour l'application, permettant au système de continuer à effectuer un travail utile pendant que la reconfiguration a lieu, sans qu'il ne soit nécessaire de connaitre les détails de mise en œuvre du code gérant la reconfiguration.
— Idéalement, un outil de conception de haut niveau d'abstraction devrait automatiser le processus de mapping d'une description d'application adaptative au niveau système en une implémentation spécifique, sans nécessiter une compréhension approfondie de l'architecture de bas niveau sous-jacente.

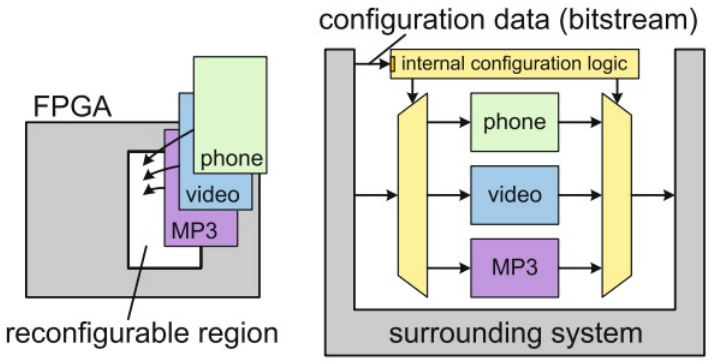



FIGURE 1.7 – *Exemple d'application de la DPR dans un téléphone mobile [Koch, 2013].*

### 1.3.1 Contrôle de la reconfiguration dynamique partielle

En règle générale, la mise en œuvre des systèmes à reconfiguration dynamique dépend d'une infrastructure comprenant des modules spécifiquement conçus pour le contrôle et l'exploitation du système. Le contrôleur de reconfiguration joue un rôle essentiel dans la gestion du processus de reconfiguration parmi ces modules. Chaque instant, il assure la supervision de l'insertion et du retrait des coeurs IP reconfigurables sur le dispositif reconfigurable en fonction des besoins du système. Il remplit un rôle similaire à celui d'un chargeur dans un système d'exploitation, en chargeant les configurations à exécuter sur le matériel reconfigurable, suivant un ordonnancement de tâches prédéfini [Carvalho et al., 2004].

La reconfiguration de la région dynamique est gérée par un contrôleur de reconfiguration qui peut être soit externe, soit interne au FPGA (auto-configuration). Il prend en charge le bitstream de reconfiguration partielle à charger dans la région modulaire reconfigurable. Toutefois, l'auto-configuration, réalisée via l'interface parallèle interne ICAP, permet de réduire le temps de reconfiguration par rapport à une reconfiguration externe.

D'un point de vue prise de décision, les modèles d'architectures de contrôle de la DPR peuvent être classés en des modèles d'architectures centralisées et des modèles d'architecture décentralisées tels que présentés dans [Trabelsi, 2013] et [Dalbouchi et al., 2023] (voir figure 1.8). Souvent logicielle, l'implémentation standalone peut se faire par un processeur hardcore ou softcore. Un processeur hardcore (ARM Cortex-R par exemples) présente l'avantage d'être implémenté directement sur la puce plutôt que par des slices, ce qui lui confère une personnalisation et une optimisation supérieures par rapport à un processeur softcore (tel qu'un MicroBlaze) implémenté sur du matériel générique (les slices). Cette mise en œuvre est adaptée aux problèmes de contrôle basiques où il n'est pas nécessaire d'avoir recours aux services avancés d'un système d'exploitation (Operating System OS). Les OSs pour les systèmes reconfigurables sur FPGA se distinguent des systèmes d'exploitation classiques par la nécessité de gérer la diversité des tâches (logicielles et matérielles) à ordonnancer et à faire communiquer. Un OS utilisé pour contrôler un système reconfigurable sur FPGA offre une grande souplesse par rapport à une solution standalone grâce à des services tels que l'attribution dynamique des tâches aux modules reconfigurables et la virtualisation de la communication entre les tâches matérielles et logicielles. Toutefois, le coût en termes de temps d'exécution peut être considérable, en particulier lorsque l'OS est implémenté en logiciel.

En raison de son approche globale, l'implémentation d'un contrôleur centralisé devient dépendante du système ciblé, ce qui réduit les possibilités de réutilisation et de

scalabilité. Pour y remédier, il est possible de répartir le contrôle entre plusieurs entités gérant chacune un problème de contrôle local. Le contrôle distribué permet d'éviter donc certains problèmes de communication, de favoriser la réutilisation des contrôleurs locaux et d'offrir une meilleure scalabilité du contrôle. Le contrôle complètement distribué répartit les décisions entre plusieurs nœuds autonomes, améliorant la résilience et la scalabilité certes, mais introduisant une complexité de communication et des risques d'incohérences. Deux modèles principaux de coordination entre contrôleurs ont été suggérés. La première approche repose sur la communication entre chaque contrôleur et tous les autres. Le modèle de coordination suivant consiste à ce que chaque contrôleur ne communique qu'avec ses voisins, ce qui réduit le nombre de messages échangés entre contrôleurs et la complexité de leurs échanges. Il reste néanmoins indispensable que chaque contrôleur comprenne le comportement des autres, ce qui peut entraîner une dépendance au système et rendre leur réutilisation plus complexe. Pour surmonter ces défis, un contrôle hiérarchique présente l'avantage de réduire le nombre d'échanges nécessaires entre les contrôleurs avant de parvenir à une décision commune satisfaisante. La distinction entre les problèmes de contrôle locaux et globaux dans un modèle hiérarchique favorise également la réutilisation des contrôleurs. Il est recommandé d'utiliser la négociation pour coordonner les contrôleurs lorsqu'une reconfiguration par l'un d'eux dépend de reconfigurations gérées par d'autres. Une négociation est alors initiée afin de parvenir à une configuration globale optimale. La coordination sans négociation repose uniquement sur l'échange d'informations entre les contrôleurs, sans recourir à des offres ou acceptations. Chaque contrôleur traite ces informations en suivant une stratégie prédéfinie pour orienter ses décisions de reconfiguration

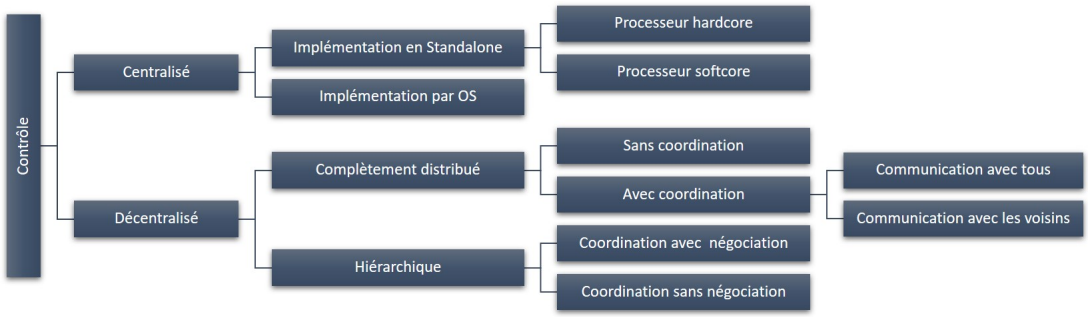



FIGURE 1.8 – *Classification des modèles de contrôle de la DPR.*

### 1.3.2 Processus de la reconfiguration dynamique partielle

La gestion du processus de reconfiguration est assurée par un contrôleur dédié. Il est essentiel de maintenir des performances optimales, en particulier en termes de bande passante, pour les systèmes reconfigurables. En effet, des procédures de reconfiguration plus rapides permettent de diminuer la latence liée au remplacement d'un module reconfigurable dans une région reconfigurable. Dans les dispositifs Xilinx, la mémoire de configuration peut être chargée avec un bitstream partiel via deux chemins principaux : le Processor Configuration Access Port (PCAP) et l'Internal Configuration Access Port (ICAP). La Figure 1.9 illustre ces deux méthodes pour les dispositifs Zynq-7000 les plus couramment utilisés [Xilinx Inc., 2018]. Le PCAP permet de reconfigurer le FPGA à partir d'un processeur externe. En revanche, l'ICAP est utilisé pour la reconfiguration partielle du FPGA via un bloc IP intégré directement dans le FPGA. Dans les deux cas, le bitstream partiel est stocké en mémoire et transféré lors de la reconfiguration partielle à l'aide d'un composant de DMA (Direct Memory Access).

Grâce à l'utilisation de bibliothèques logicielles dédiées, notamment PYNQ [4] et Baremetal [5], le chemin PCAP est le mieux pris en charge par la chaîne d'outils Xilinx et ses dispositifs [Duhamel, 2022]. Lors du chargement d'un bitstream partiel, le processeur envoie une requête à l'interface Device Configuration (DevC), permettant au moteur DMA intégré de récupérer le bitstream depuis la mémoire DDR. Au préalable, le bitstream doit avoir été transféré des mémoires locales vers la mémoire DDR. Ensuite, le bitstream est transmis au contrôleur PCAP, qui en vérifie l'intégrité et la validité avant de le charger dans la mémoire de configuration. L'interface de configuration spécifique, déjà incluse dans les SoCs modernes de Xilinx, facilite le processus de reconfiguration sans coût supplémentaire pour le concepteur. Cependant, les bibliothèques Python et C disponibles ont une bande passante inférieure à celle des solutions PCAP personnalisées, qui peuvent atteindre jusqu'à 145 Mbps [Vipin et Fahmy, 2018]. Le chemin PCAP utilise également le processeur pour configurer le moteur DMA de l'interface DevC, garantissant ainsi la récupération correcte des bitstreams depuis la mémoire (DDR ou carte SD).

Le chemin ICAP est une autre alternative pour réaliser des opérations de reconfiguration partielle dynamique. Ses principaux avantages incluent une bande passante plus élevée et la libération du processeur des tâches liées à la DPR. Cependant, cette méthode nécessite une logique dédiée au sein de la matrice FPGA pour effectuer la DPR. Les SoCs de Xilinx permettent à la logique programmable statique d'accéder à la ressource ICAP. Théoriquement, un design statique pourrait récupérer un bitstream depuis la mémoire DDR et reconfigurer une région reconfigurable sans l'intervention du processeur. En pratique, cependant, le processeur, ou un processeur logiciel comme le Microblaze, est souvent utilisé pour initier le processus de reconfiguration partielle dynamique DPR à l'aide de bibliothèques dédiées.

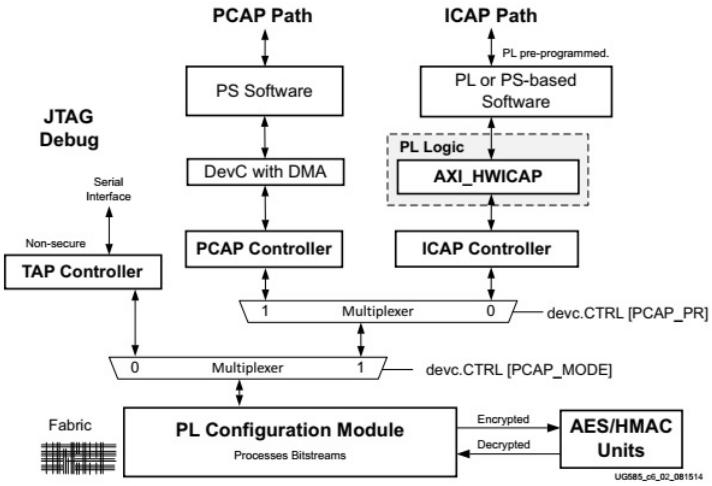



FIGURE 1.9 – *Reconfiguration dynamique via les ports ICAP et PCAP [Xilinx Inc., 2018].*

### 1.3.3 Avantages de la reconfiguration dynamique partielle

La DPR peut offrir plusieurs avantages aux concepteurs de SoCs reconfigurables à base de FPGAs. Tout d'abord, la densité logique effective de la puce peut être augmentée par le biais d'un multiplexage temporel des ressources matérielles entre des calculs mutuellement exclusifs, permettant ainsi d'inclure une application plus grande sur une

4. https ://www.pynq.io/
5. https ://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841745/Baremetal+Drivers+and+Libraries

puce de taille réduite. De plus, la DPR présente l'avantage d'un temps de reconfiguration réduit par rapport à une reconfiguration complète, car ce temps est directement proportionnel à la taille du fichier de configuration, qui, à son tour, est proportionnel à la superficie de la puce en cours de reconfiguration. Cela signifie que la reconfiguration peut être appliquée dans des systèmes ayant des exigences temporelles critiques. La DPR est bénéfique dans les systèmes matériels adaptatifs, car ils peuvent adapter les calculs à un environnement changeant tout en continuant à traiter les données. De plus, la DPR est utile dans des scénarios où une interface doit rester active pendant que la fonctionnalité évolue. Par exemple, dans un système FPGA interfacé avec un ordinateur hôte via un bus PCI Express (PCIe), une reconfiguration complète du FPGA interrompt le lien de communication, ce qui peut même nécessiter un redémarrage de l'ordinateur hôte pour le rétablir. La DPR permet de maintenir le lien en gardant la circuiterie d'interface active tandis que la reconfiguration est en cours pour la partie accélérée [Vipin et Fahmy, 2018]. En outre, la DPR constitue un levier pour améliorer le démarrage des systèmes en réduisant les délais, en optimisant l'utilisation de la mémoire, en accélérant les phases critiques grâce à des accélérateurs matériels, et en offrant une flexibilité pour ajuster les fonctionnalités dès le démarrage [Koch, 2013].

### 1.3.4 Coût de la reconfiguration dynamique partielle

Le coût de la DPR pour les SoCs dépend de plusieurs facteurs :

1. **Temps de reconfiguration** : Le calcul du temps de reconfiguration (temps nécessaire pour reconfigurer le système) a évolué dans la littérature : d'abord considéré comme nul ou constant, il a ensuite été affiné en fonction de la taille des bitstreams, de l'interface de configuration utilisée et de la méthode de transfert des données. Plus ce temps est long, plus les performances sont impactées. [Papadimitriou et al., 2011] estiment que le principal surcoût provient du transfert des bitstreams de la mémoire externe vers la mémoire interne ainsi que du processeur vers l'ICAP via le bus. Les FPGA modernes nécessitent plusieurs millisecondes, voire plusieurs secondes pour effectuer une opération de reconfiguration.
2. **Utilisation des ressources** : Les ressources FPGA peuvent être sous-utilisées en raison des contraintes sur la forme et l'emplacement des régions reconfigurables, ce qui entraîne une inefficacité dans l'utilisation des ressources disponibles.
3. **Consommation d'énergie** : La DPR peut entraîner une consommation d'énergie supplémentaire, surtout si elle est fréquente. Cela peut être un facteur important dans les systèmes embarqués et les dispositifs à faible consommation.
4. **Complexité de conception** : La mise en œuvre de la reconfiguration dynamique ajoute de la complexité à la conception et à la vérification du système. Cela peut augmenter le temps de développement et les coûts associés.
5. **Fiabilité et maintenance** : Les processus de reconfiguration peuvent introduire des risques de fiabilité supplémentaires. La maintenance et la gestion des erreurs dans un environnement reconfigurable peuvent aussi entraîner des coûts additionnels.
6. **Outillage de développement supplémentaire** : Les outils logiciels nécessaires pour la conception, la simulation et la mise en œuvre de la DPR peuvent induire des surcoûts à travers des licences coûteuses, une complexité accrue de conception, des temps de compilation et de simulation prolongés, des besoins en formation et des coûts de mises à jour et de maintenance des outils eux-mêmes.

Les travaux liés à la DPR dans leur majorité abordent l'optimisation de la DPR (réduire le surcout) par le biais de contrôleurs matériels ou bien en traitant le problème de

relocation des bitstreams. En effet, les contrôleurs DPR proposés par les fournisseurs de FPGA sont conçus pour des architectures à usage général, ce qui les amène à répondre à une grande variété de besoins, entraînant ainsi des implémentations matérielles importantes. De plus, leur gestion de la reconfiguration dépend souvent de bibliothèques logicielles et du support du processeur, ce qui peut entraîner des délais imprévisibles et une reconfiguration plus lente.

Xilinx propose un contrôleur DPR propriétaire (en closed-source) dans sa bibliothèque de cœurs IP. Cependant, les caractéristiques de synchronisation et de vitesse ne sont pas spécifiées, ce qui transforme essentiellement le contrôleur en une boîte noire inadaptée aux systèmes temps réel stricts. Dans [Pezzarossa et al., 2017], RT-ICAP un contrôleur léger spécialement développé pour prendre en charge la DPR dans les systèmes à temps réel propose différents modes de fonctionnement pour effectuer la reconfiguration dans un laps de temps délimité et prévisible. Egalement, les auteurs ont présenté un outil logiciel pour la conversion et la compression des bitstreams, ainsi que pour l'analyse du temps de reconfiguration. L'utilisation du bus AXI (Advanced eXtensible Interface) dans le contrôleur de reconfiguration proposé par [Kirchhoff et al., 2019] fournit une solution flexible et performante pour la DPR avec un faible surcout en ressources FPGA. Le contrôleur dont la structure de base est implémentée en VHDL est composé de trois machines à états, chacune responsable d'un canal d'information : le canal d'adresse AXI, le canal de données AXI et le canal ICAP. Ainsi, la séparation de l'accès et du transfert des bitstreams permet d'exploiter au mieux la bande passante et la capacité de lecture en rafale du contrôleur de mémoire DDR. Le flot de conception proposé dans [Dalbouchi et al., 2023] procure la possibilité de concevoir à la fois un contrôleur centralisé et décentralisé, en fonction des exigences de contrôle du système cible. La conception d'un contrôleur décentralisé, en répartissant le problème de contrôle entre des contrôleurs autonomes, chacun associé à un composant reconfigurable, peut s'avérer bénéfique pour les systèmes reconfigurables complexes. Dans cette approche, un coordinateur ayant une visibilité limitée aux exigences et contraintes globales du système peut être employé, laissant la gestion des données de contrôle locales aux contrôleurs distribués. Cela réduit la complexité des contrôleurs, facilite leur vérification et leur réutilisation, et améliore l'évolutivité du modèle de contrôle. Le framework ZyPR présenté dans [Bucknall et Fahmy, 2023] offre un contrôleur de reconfiguration asynchrone haute performance utilisant l'interface ICAP de ZynqMP avec un débit proche du débit théorique (augmenté à 757 MiB/s). La reconfiguration se fait via une interface DMA modifiée pour Linux, permettant des transferts de mémoire non bloquants.

Il est important de souligner que la DPR a évolué en passant de l'utilisation d'interfaces FPGA externes et d'interfaces internes spécialisées, telles que l'ICAP sur les dispositifs Xilinx, à l'utilisation de cœurs de contrôleur pour offrir des fonctions de gestion permettant des conceptions partiellement reconfigurables et autonomes, ainsi qu'une reconfiguration dynamique via le port de reconfiguration dynamique (Dynamic Reconfiguration Port DRP) [Kornaros et al., 2024].

D'autre part, la relocalisation de bitstream est une technique qui permet à un bitstream partiel d'être utilisé pour reconfigurer une partie du FPGA pour laquelle il n'a pas été initialement généré. Grâce à cette technique, un seul bitstream partiel est nécessaire pour implémenter un module dynamique dans plusieurs régions reconfigurables. Cela peut entraîner des temps de conception plus courts, car chaque module dynamique n'a besoin que d'une seule implémentation physique, quel que soit le nombre de régions relocalisables. Cela peut également réduire l'espace mémoire nécessaire pour stocker les bitstreams partiels, puisqu'un seul bitstream suffit pour l'implémenter dans n'importe quelle région relocalisable [Lalevée et al., 2016].

Plusieurs travaux de recherche ont proposé des méthodologies de conception reposant sur la relocalisation des bitstreams. De telles approches peuvent offrir certains avantages en termes de flexibilité. Cependant, étant donné que le flux standard de Xilinx ne prend pas en charge la relocalisation des bitstreams, les outils tiers peuvent ne pas offrir le même niveau de fiabilité. De plus, comme les techniques de relocation des bitstreams sont généralement spécifiques au matériel, elles peuvent limiter la portabilité du flux proposé [Seyoum et al., 2021]. La technique de relocation partielle des bitstreams sur puce (Partial Bitstream Relocation), permet de réduire le temps de configuration des modules dans les FPGA tout en minimisant les ressources mémoire nécessaires pour stocker les bitstreams partiels des modules reconfigurables. Ainsi, des outils comme OORBIT [Touiza et al., 2013], RePaBit [Rettkowski et al., 2016], IMPRESS [Zamacola et al., 2018] et Reloc [Gottschall et al., 2018] permettent chacun de traiter un certain nombre de contraintes imposées par les outils commerciaux telles l'empilement de partitions reconfigurables dans une région d'horloge, la reconfiguration hiérarchique, la communication entre modules reconfigurables et le découplage de l'implémentation des modules statiques et reconfigurables.

L'adoption de la DPR implique un *compromis* entre les avantages liés à la virtualisation matérielle, à l'optimisation et à la flexibilité d'une part, et les coûts supplémentaires induits que les concepteurs doivent prendre en compte lors du développement de systèmes reconfigurables.

### 1.3.5 Flot de conception des systèmes reconfigurables

La génération d'une architecture sur FPGA s'effectue en plusieurs étapes, allant de la spécification des exigences jusqu'à la génération du bitstream en passant par la description architecturale en HDL, la synthèse, le placement et le routage. Ces étapes sont généralement exécutées en utilisant une chaîne d'outils propriétaire du constructeur du FPGA.

Le flot de conception DPR pour les plates-formes Xilinx comprend quatre étapes majeures, à savoir : le partitionnement, le floorplanning, la génération de la partie statique et enfin le routage et la génération du bitstream [Seyoum et al., 2021] . Sur la figure 1.10, pour l'outil de conception Vivado [6], les rectangles pleins indiquent les étapes automatisées par le biais de scripts, tandis que les rectangles en pointillés indiquent les phases manuelles nécessitant une intervention de l'utilisateur.

— **Le partitionnement :** L'ensemble des tâches matérielles (chacune correspondant à un module reconfigurable) est partitionné et alloué à des régions reconfigurables distinctes sur la structure FPGA. Chaque région reconfigurable peut héberger plusieurs modules reconfigurables, qui seront exécutés de manière multiplexée dans le temps. Les applications considérées ont un nombre total $N^{sw}$ et $N^{hw}$ de tâches logicielles et de tâches matérielles respectivement. Les tâches matérielles sont hébergées sur $N^{RR}$ régions reconfigurables du FPGA, où $N^{RR} \leq N^{hw}$. Les modules reconfigurables sont alloués aux régions reconfigurables en mode « plusieurs à un » ou « un à un » et aucun module reconfigurable ne peut être alloué à plus d'une région reconfigurable ; le premier mode correspond au cas où la région reconfigurable est gérée sous DPR, tandis que le second correspond au cas particulier où un module reconfigurable est statiquement alloué à une région reconfigurable. Une région reconfigurable doit également satisfaire les exigences en terme de ressources CLB (Configurable Logic Block), BRAM (Block RAM), DSP (Digital Signal Processor) de tous les modules qui lui sont alloués.

6. https ://www.xilinx.com/products/design-tools/vivado.html

— **Le floorplanning :** Cette étape consiste en la génération de placements physiques pour les régions reconfigurables sur la structure FPGA. Les placements générés doivent satisfaire les besoins en ressources des modules reconfigurables qui leur sont alloués, doivent respecter l'ensemble des contraintes technologiques et structurelles de la DPR définies par le fournisseur de FPGA, et en fin, ne doivent pas avoir d'impact sur les performances de la conception, c'est-à-dire que les emplacements ne doivent pas être la cause d'une violation de contraintes temporelles du point de vue routage. À titre d'exemple, certaines des contraintes technologiques de floorplaning posées par Xilinx Vivado spécifient que les régions reconfigurables ne doivent pas se chevaucher, ne doivent pas partager de tuiles dans la même région d'horloge, doivent être rectangulaires et les limites verticales gauche et droite des régions reconfigurables ne peuvent pas être placées entre des tuiles d'interconnexion connectées dos à dos.
— **La génération de la partie statique :** La partie statique d'une conception DPR comprend la portion de la surface qui n'est pas soumise à la DPR ; les régions reconfigurables y sont généralement définis comme des boîtes noires, c'est-à-dire des modules sans implémentation logique à l'intérieur, dont les interfaces correspondent à l'union des interfaces des modules reconfigurables qui leur sont alloués. La génération de la partie statique est une étape de conception qui suit l'étape de partitionnement, car le nombre de régions reconfigurables et l'allocation des modules reconfigurables aux régions reconfigurables ne sont pas connus a priori. La partie statique d'une conception basée sur la DPR peut être générée à partir d'un modèle au moment de la conception (sur la base des sorties de l'étape de partitionnement) ou le concepteur peut préparer un ensemble de points de contrôle statiques pré-synthétisés avec un nombre différent de régions reconfigurables, qui peuvent ensuite être sélectionnées en fonction des résultats de l'étape de partitionnement pour la génération finale du bitstream. La partie statique est généralement spécifique à la conception et son contenu peut aller d'une simple chaîne hiérarchique d'interconnexions AXI (Advanced eXtensible Interface) - pour connecter les régions reconfigurables à la mémoire et aux processeurs - à des implémentations matérielles complexes telles que des moteurs cryptographiques, des cœurs de traitement vidéo, des encodeurs HDMI, des ports d'acquisition d'E/S parallèles, etc.
— **Le routage et la génération du bitstream :** L'étape du routage dans le flot de conception Vivado consiste à créer des connexions physiques entre les éléments de conception sur le FPGA, alors que la génération du bitstream convertit des informations de conception en un fichier binaire qui configure effectivement le FPGA pour implémenter le design spécifié. Les outils de conception des fournisseurs automatisent généralement cette dernière étape (voir la section 3.2).

## 1.4 Modélisation des SoCs

Le processus de conception d'un SoC implique souvent plusieurs niveaux d'abstraction, allant de la spécification initiale jusqu'au produit final. Ces niveaux présentent divers compromis entre la rapidité d'analyse et de synthèse et la précision des résultats obtenus. À mesure que la modélisation passe d'un niveau élevé à un niveau plus détaillé, les spécifications architecturales deviennent de plus en plus précises, tandis que l'espace pour ces solutions devient de plus en plus restreint. Le processus de conception démarre au niveau le plus élevé, considéré comme le plus abstrait et le moins précis, puis évolue soit de manière automatique, soit par une intervention manuelle, vers le niveau le plus bas, classifié à l'inverse comme le plus concret et le plus précis. La

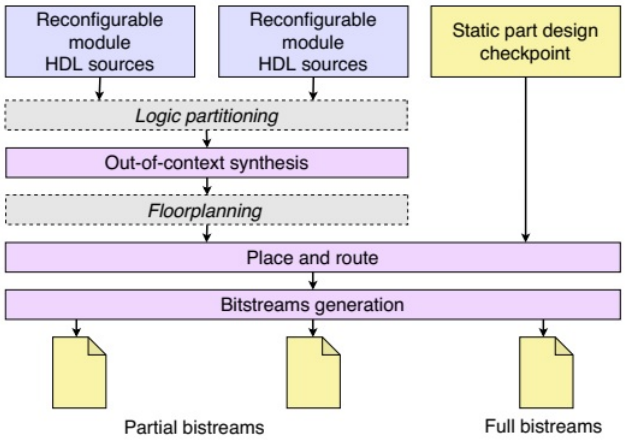



FIGURE 1.10 – *Flot de conception de la DPR pour les plates-formes Xilinx [Seyoum et al., 2021].*

conception des SoCs au niveau RTL (Register Transfer Level) et aux niveaux inférieurs, caractérisés par leur précision jusqu'au niveau CABA (Cycle Accurate Bit Accurate), rencontre des défis en raison du temps nécessaire à la production et à la vérification du système complet ; cette complexité découle de la gestion d'un nombre très élevé d'états du système. Pour résoudre cette problématique, les concepteurs se tournent de plus en plus vers le niveau système (ESL), offrant une abstraction plus élevée dans la conception. Au niveau ESL, la conception de SoC devient plus rapide et plus aisée, parfois même sans compromettre la précision. Cette approche permet à l'équipe de conception de simuler et de valider rapidement les spécifications initiales, tout en explorant un large éventail d'alternatives d'implémentation. Dans [Thorsten et al., 2003], les auteurs ont défini un ensemble de critères permettant de mesurer la précision d'un modèle par rapport à son implémentation. Il convient de souligner que ces critères ne garantissent pas toujours une évaluation complète de la précision d'un modèle dans son ensemble. Dans le contexte d'une description modulaire, les mesures peuvent se concentrer sur un module spécifique, voire même sur un sous-module.

— **La précision structurelle (Structural accuracy) :** ce critère évalue le degré de fidélité de la structure du modèle par rapport à celle de l'implémentation. Cela englobe l'identification des composants et leur division en modules matériels ou logiciels. Dans le cas d'un module matériel, la précision structurelle concerne la description de sa composition interne et de son interface. Pour un composant logiciel, la précision consiste à identifier les différentes tâches et à les faire correspondre avec le jeu d'instructions du processeur lors de l'implémentation.

— **La précision temporelle (Timing accuracy) :** ce critère évalue le degré de précision temporelle du modèle par rapport à son implémentation. Il est déterminé par la présence ou l'absence d'annotations temporelles permettant de simuler les latences de transmission, les délais de traitement et de communication, ainsi que le mode de synchronisation (basé sur l'horloge ou les appels de fonctions). Lorsque le modèle ne contient aucune notion de temps, il est considéré comme non-temporisé (untimed). Le niveau de granularité temporelle maximale est atteint lorsque les valeurs temporelles sont exprimées en termes de cycles d'horloge, on parle alors du niveau cycle accurate.

— **La précision fonctionnelle (Functional accuracy) :** c'est un critère qui s'inscrit dans une perspective fonctionnelle du modèle. Il vise à évaluer la fidélité des fonctionnalités du modèle par rapport à son implémentation, qu'elle soit partielle ou totale. Dans le cas des modèles abstraits, certaines fonctionnalités complexes peuvent volontairement être omises.

— **La précision de l'organisation des données (Data organisation accuracy)** : elle se rapporte à l'évaluation de la fidélité des types de données et donc des interfaces par rapport à l'implémentation. Dans les modèles au niveau système, les données peuvent être représentées par des valeurs entières. La précision est maximale lorsque les données sont décrites par des vecteurs de bits (bit-accurate).
— **La précision du protocole de communication (Communication protocol accuracy)** : c'est une évaluation du degré de précision du support des communications par rapport aux protocoles adoptés dans l'implémentation. Ce critère tient compte des niveaux de précision structurel et temporel dans la description des communications.

Dans le cadre de notre travail, nous nous sommes intéressés particulièrement au niveau système ESL incluant les différents niveaux d'abstraction qui se trouvent au-dessus du niveau RTL. La figure 1.11 illustre les différents niveaux d'abstraction dans un flot typique de conception d'un SoC.

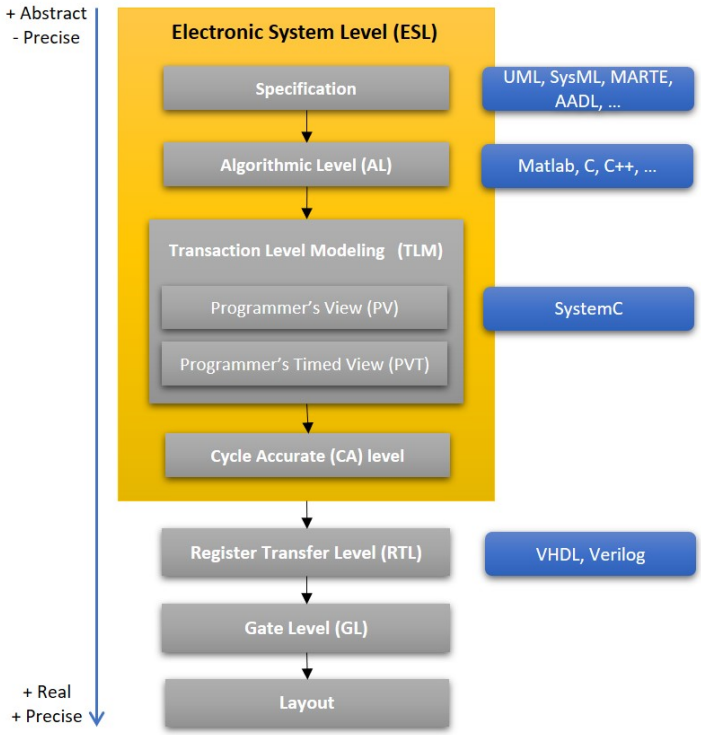



FIGURE 1.11 – *Les différents niveaux d'abstraction pour la modélisation d'un SoC.*

### 1.4.1 Modélisation au niveau système

ESL (Electronic System level) est une méthodologie de conception qui consiste à utiliser un niveau d'abstraction approprié afin d'améliorer la compréhension du système et l'implémentation efficace des fonctionnalités à moindre coût [Martin et al., 2007]. L'adoption de la conception ESL est essentielle pour réduire l'écart de productivité; ESL aborde l'écart de productivité en augmentant l'abstraction du modèle initial et en introduisant des activités de conception automatisées autour de ce modèle au niveau du système [Soonhoi et Jürgen, 2017].

La spécification ESL est transformée automatiquement en une conception RTL grâce à des outils de synthèse de haut niveau (High Level Synthesis - HLS). Cependant, dans

de nombreux cas, il existe encore une différence significative entre la qualité des résultats obtenus par les outils HLS et ceux obtenus par une optimisation manuelle d'une conception RTL [Bijan et Masoud, 2023].

#### 1.4.1.1 Niveau Algorithme (Algorithmic Level - AL)

À ce stade, l'application est décrite sous forme algorithmique en se basant sur une spécification standard ou une documentation existante. Les modèles à ce niveau sont formulés à l'aide d'un langage de description de haut niveau tel que Matlab, C ou C++. Ensuite, ils sont soumis à une analyse fonctionnelle précoce et à des vérifications afin de partitionner l'application de manière efficace entre les tâches matérielles et logicielles.

#### 1.4.1.2 Niveau transactionnel (Transaction Level Modeling - TLM)

TLM est une approche de modélisation basée sur des transactions et fondée sur des langages de programmation de haut niveau tels que SystemC. Elle met en avant le concept de séparation de la communication et du calcul au sein d'un système [Frank, 2005]. Une transaction, cœur de la modélisation TLM, est une unité de données échangée entre les composants d'un système ; ces derniers sont modélisés sous forme de modules comprenant un ensemble de processus concurrents. Les modules échangent des communications sous forme de transactions à travers un canal abstrait.

Pour combler le fossé entre les descriptions purement fonctionnelles et le niveau RTL, l'objectif principal de TLM est de simplifier la représentation des fonctionnalités complexes du système en éliminant les détails matériels inutiles aux premières étapes du processus de conception ce qui permet de réduire considérablement la vitesse de simulation en minimisant le nombre d'événements et d'informations à traiter. Ainsi, les modèles transactionnels sont jusqu'à 1000 fois (voire 10 000x) plus rapides à simuler que leurs équivalents RTL. Comme illustré sur la figure 1.12, les différentes connexions élémentaires en RTL sont représentées par une seule connexion en TLM, orientée de l'initiateur de la communication vers la cible.

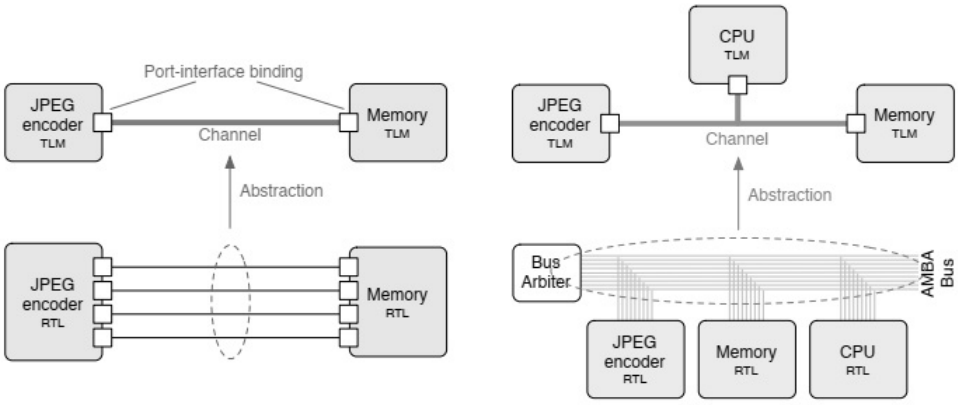



FIGURE 1.12 – *Exemples de conceptions TLM et RTL [Klingauf, 2008].*

Par exemple, la performance obtenue d'une plateforme CODEC MPEG4 sur une machine SUN-Ultra10 d'une vitesse de 330 MHz et d'une mémoire de 256 Mo en termes de taille du code et de vitesse de simulation sur différents environnements complémentaires est représentée dans la figure 1.13. Cette amélioration significative de la vitesse de simulation permet une vérification fonctionnelle plus approfondie et plus efficace, surmontant les limitations inhérentes à la vérification des modèles RTL. Un autre avantage est que TLM nécessitent beaucoup moins d'efforts de modélisation que RTL ou qu'un


(a) *Nombre de lignes de code RTL vs TLM.* (b) *Vitesse de simulation RTL vs TLM.*

FIGURE 1.13 – *Performances obtenues pour une plateforme CODEC MPEG4 [Frank, 2005].*

modèle au cycle précis ; cet effort de modélisation est encore réduit lorsqu'il existe déjà un code fonctionnel C/C++ à réutiliser pour les tâches de traitement effectuées par le bloc matériel à modéliser.

Tel que montré sur la figure 1.11, TLM est décrit comme une hiérarchie de plusieurs sous-niveaux. Le sous niveau Loosely-Timed (LT) appelé aussi Programmer's View (PV) et le sous niveau Approximately-Timed (AT) qualifié de programmer's View plus Time (PVT) diffèrent principalement par leur gestion du temps et la granularité des détails temporels qu'ils offrent pour la modélisation des systèmes.

Egalement, il faut noter que la modélisation au cycle précis (voir la sous section 1.4.1.3) dépasse actuellement les capacités de TLM-2.0. Bien qu'il soit possible de créer des modèles au cycle près en utilisant SystemC et TLM-1 dans leur forme actuelle, la standardisation d'un style de codage spécifique pour cette précision reste une question non résolue, qui pourrait être traitée par une future norme de l'Accellera Systems Initiative [IEEE, 2023].

**Loosely-Timed (LT)** A ce niveau d'abstraction, les mécanismes de communication implémentés dans les interfaces des composants sont simplifiés et ne prennent pas en compte la notion de temps. il est principalement utilisé pour valider, au niveau algorithmique, le système complet en exécutant l'application finale sur un modèle transactionnel d'architecture. Cependant, certaines propriétés extra-fonctionnelles telles que le temps d'exécution et la consommation d'énergie sont soit omises, soit approximées. Certains détails architecturaux tels que l'activité des caches et l'arbitrage des bus ne sont pas pris en compte. Bien que ces détails aient un impact significatif sur la précision des modèles de simulation et les mesures de performance, les inclure dans les modèles de simulation ralentirait le processus. Ces aspects sont plutôt pris en compte dans le niveau inférieur. Un modèle LT peut être affiné pour devenir un modèle AT afin d'obtenir une meilleure précision, bien que cela entraîne une dégradation des performances de simulation.

**Approximately-Timed (AT)** Avec des annotations temporelles plus précises, différents détails architecturaux sont inclus pour les parties de traitement et de communication. L'interconnexion est modélisée et une forme d'arbitrage de la communication est appliquée. Par conséquent, les modèles AT sont généralement utilisés pour l'évaluation des performances aux premiers stades de la conception, notamment pour l'exploration et la vérification de l'architecture de conception. Les styles de codage TLM seront développés dans la section 1.5.2.2.

### 1.4.1.3 Le niveau Cycle Précis (Cycle Accurate Level - CAL)

Le niveau Cycle Précis offre une représentation détaillée de l'état du modèle à chaque cycle d'horloge, se positionnant entre la simulation traditionnelle basée sur des événements et les modèles de transaction de haut niveau. Dans ce niveau, les transitions

entre les états sont explicitées à chaque cycle d'horloge, contrairement aux modèles de transaction où les états restent inchangés entre deux transactions successives sur le bus. En général, le comportement interne d'un composant implémente et calcule les différentes sorties en fonction des entrées actuelles et passées à l'aide d'un langage de programmation standard, tel que C ou SystemC. Sur le plan du traitement, une description détaillée de la micro-architecture interne du processeur est effectuée, incluant le pipeline, la prédiction de branchement, le cache, etc. Concernant la communication, un protocole précis au bit près est adopté. Cette précision des modèles améliore la qualité de l'estimation des performances et facilite la détection des erreurs de comportement. Cependant, ces modèles ont une vitesse de simulation limitée, généralement plus rapide que celle du niveau de transfert de registres RTL, mais nécessitent un effort de modélisation important sans pour autant fournir de description synthétisable.

### 1.4.2 Le niveau transfert de registres (Register Tranfert Level - RTL)

La modélisation au niveau RTL consiste à décrire l'implémentation physique du système en utilisant des blocs élémentaires tels que des bascules, des multiplexeurs, des UALs, des registres, etc. La logique combinatoire est ensuite utilisée pour connecter les entrées/sorties de ces blocs. C'est le premier niveau d'abstraction suffisamment précis pour être synthétisé automatiquement et efficacement. Il offre une précision au niveau du bit et du cycle, détaillant tous les composants internes. La conceptions à ce niveau sont généralement réalisées en utilisant les langages VHDL ou Verilog. Cependant, en raison de la nature hautement parallèle du matériel, la simulation d'une conception RTL de grande taille est extrêmement lente.

### 1.4.3 Le niveau porte logique (Gate Level - GL)

Le niveau porte logique décrit le comportement sous forme d'équations booléennes et abstrait beaucoup de détails (par exemples ceux liés au placement/routage) en se concentrant uniquement sur la description des portes logiques (ET, OU, bascules, etc.) et de leurs connexions. Une telle description forme la netlist de sortie à partir de l'outil de synthèse RTL, généralement implémentée à l'aide des primitives de niveau de porte Verilog, puis importée dans un outil EDA (Electronic Design Automation) de placement et de routage. Le niveau de porte est utilisé pour implémenter des modules de plus bas niveau dans une conception, tels que des additionneurs complets, des multiplexeurs et d'autres circuits numériques .

### 1.4.4 Le niveau dessin de masque (Layout Level)

A ce dernier niveau d'abstraction, le comportement électrique est modélisé à l'aide d'équations différentielles ou de fonctions de transfert. Une représentation structurelle divise le circuit en une interconnexion de transistors, tandis qu'une représentation géométrique détaille le placement et le routage de ces transistors sur le substrat de silicium. Ce niveau fournit la description la plus précise de la puce, où l'emplacement et la conception de chaque transistor sont précisément connus et doivent être minutieusement vérifiés. Enfin, les transistors et leurs connexions sont transposés sur des masques utilisés dans le processus de fabrication. Ces masques sont très coûteux à produire, à titre indicatif, pour les technologies de pointe les plus avancées avec une finesse de gravure inférieure à 10 nm, le coût de fabrication d'un jeu de masques peut atteindre plusieurs millions de dollars, pouvant dépasser les 10 millions de dollars dans certains cas (3 nm peut coûter jusqu'à 40 millions de dollars)[7]. Par conséquent, les erreurs de

7. https ://www.semianalysis.com/p/the-dark-side-of-the-semiconductor

conception matérielles peuvent être économiquement désastreuses à corriger en raison de la nécessité de reconstruire ces jeux de masques. Il est donc essentiel d'éviter ces erreurs dès les premières étapes du processus de conception pour prévenir des conséquences financières et des retards dans la production.

## 1.5 Aperçu sur SystemC/TLM

Alors que l'industrie électronique développe des systèmes de plus en plus complexes impliquant un grand nombre de composants, y compris des logiciels, il existe un besoin croissant d'un langage de modélisation capable de gérer la complexité et la taille de ces systèmes. SystemC offre un mécanisme pour gérer cette complexité grâce à sa capacité à modéliser le matériel et le logiciel ensemble à plusieurs niveaux d'abstraction, parmi lesquels figure le niveau TLM. Cette capacité n'est pas disponible dans les langages de description matérielle traditionnels. Dans cette section, nous visons non pas à définir de manière exhaustive les fonctionnalités de SystemC, mais plutôt à mettre l'accent sur certaines constructions indispensables à notre travail.

### 1.5.1 SystemC

SystemC est un langage de conception et de vérification de systèmes qui englobe à la fois le matériel et le logiciel. Il s'agit d'une bibliothèque de classes C++ adaptée pour modéliser le partitionnement d'un système, évaluer et vérifier l'affectation des blocs à des implémentations matérielles ou logicielles, et concevoir et mesurer les interactions entre les blocs fonctionnels. Il propose également un environnement de simulation complet avec un ordonnanceur non-déterministe et non préemptif. SystemC utilise un compilateur C++ standard qui produit un exécutable contenant à la fois le modèle et le simulateur associé ; C++17 (ISO/IEC 14882 :2017) étant la référence.

Les entreprises leaders dans les domaines de la propriété intellectuelle IP, de la conception électronique EDA, des semi-conducteurs, des systèmes électroniques et des logiciels embarqués utilisent actuellement SystemC pour l'exploration architecturale, la création de blocs matériels haute performance à différents niveaux d'abstraction et le développement de plates-formes virtuelles pour la co-conception matériel/logiciel. SystemC a été normalisé par l'Open SystemC Initiative (OSCI) et l'Accellera Systems Initiative, et ratifié en tant que norme IEEE Std 1666-2023 [IEEE, 2023].

Bien que SystemC imite délibérément les langages de description matérielle VHDL et Verilog à certains égards, il est plus justement décrit comme un langage de modélisation au niveau ESL. SystemC est utilisé pour la modélisation système, l'exploration architecturale, l'évaluation des performances, le développement logiciel, la vérification fonctionnelle et la synthèse de haut niveau.

En termes simples, SystemC permet à un utilisateur d'écrire un ensemble de fonctions C++ (processus) qui s'exécutent sous le contrôle d'un ordonnanceur dans un ordre qui imite le passage du temps simulé. Ces processus sont synchronisés et communiquent de manière utile pour la modélisation de systèmes électroniques contenant du matériel et du logiciel embarqué. Ils sont encapsulés dans une hiérarchie de modules qui capture les relations structurelles et la connectivité du système. La communication inter-processus utilise un mécanisme d'appel de méthodes d'interfaces, qui facilite l'abstraction et le raffinement indépendant des interfaces au niveau du système.

#### 1.5.1.1 Propriétés du langage SystemC

Un certain nombre de propriétés font de SystemC un langage puissant et flexible pour la modélisation et la simulation de systèmes complexes.

- Simulation basée sur des événements : SystemC permet une modélisation basée sur des événements, où les processus sont déclenchés par des événements et exécutés en fonction de ceux-ci. Cela permet une simulation précise et efficace des systèmes réactifs.
- Modélisation multi-niveaux : SystemC prend en charge la modélisation à différents niveaux d'abstraction, de la modélisation comportementale à la modélisation basée sur les transactions. Cela permet aux concepteurs de choisir le niveau approprié de détail pour leur application.
- Support matériel et logiciel : SystemC permet la modélisation de systèmes combinant à la fois du matériel et du logiciel. Cela en fait un outil idéal pour la conception de systèmes embarqués, où le matériel et le logiciel interagissent étroitement.
- Communication entre processus : SystemC offre des mécanismes de communication entre les différents processus, tels que les signaux et les canaux, facilitant l'interaction et la synchronisation entre les composants du système.
- Mécanismes d'ordonnancement et de synchronisation : SystemC fournit des mécanismes de synchronisation tels que des événements, des listes de sensibilité et des sémaphores pour l'ordonnancement des processus.
- Emulation de la concurrence : SystemC est considéré comme un langage mono-thread ; cela signifie que le code SystemC est exécuté de manière séquentielle, un seul thread à la fois. Cependant, il émule le parallélisme qui se produit au niveau matériel en permettant l'exécution de plusieurs processus en parallèle à l'aide d'un mécanisme de simulation d'événements discrets. Ainsi, bien qu'il soit mono-thread, SystemC peut modéliser des systèmes qui impliquent une concurrence et des interactions parallèles.
- Hiérarchie de modules : SystemC permet l'organisation des différents composants du système dans une structure hiérarchique de modules. Cela facilite la conception modulaire et la réutilisation des composants. Les modules sont généralement composés de processus, d'interfaces, d'événements, de données/fonctions membres et d'instances d'autres modules.
- Capitalisation sur les atouts de C++ : SystemC est basé sur le langage de programmation C++, ce qui permet aux concepteurs de profiter des fonctionnalités avancées de ce langage tout en modélisant leurs systèmes. Cela simplifie également l'intégration avec d'autres outils de développement logiciel.
- Extensibilité : SystemC est conçu pour être facilement extensible, ce qui permet aux utilisateurs d'ajouter de nouvelles fonctionnalités et de personnaliser le langage selon leurs besoins spécifiques en utilisant les mécanismes fournis par C++. Cela garantit que SystemC peut être adapté à une grande variété d'applications et d'environnements de conception.

#### 1.5.1.2 Structure d'un modèle SystemC

La structure d'un modèle SystemC peut varier en fonction de la complexité et des besoins spécifiques de la conception. Cependant, de manière générale, un modèle SystemC est organisé selon une architecture modulaire où chaque composant est encapsulé dans un module distinct et interagit avec les autres composants via des interfaces et des mécanismes de communication, comme montré sur le figure 1.14.

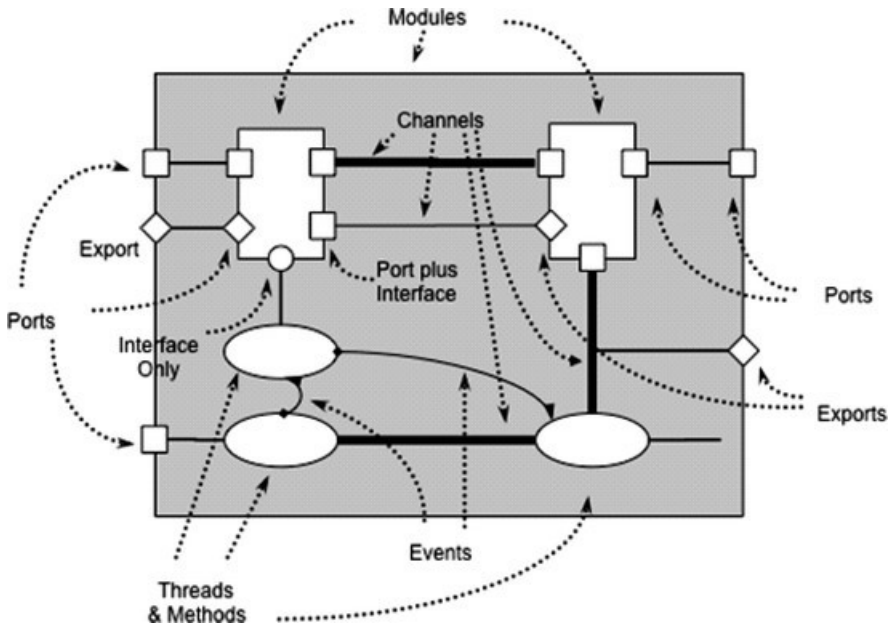



FIGURE 1.14 – *Structure d'un modèle SystemC [Black et al., 2010].*

**Module** Les modules sont les principaux éléments de construction structurels de SystemC. La classe sc_module est la classe de base pour les modules. Les objets de la classe sc_module ne peuvent être construits que pendant l'élaboration. La communication inter-modules est généralement réalisée en utilisant des appels de méthode d'interface ; c'est-à-dire qu'un module devrait communiquer avec son environnement via ses ports.

**Port** Lorsqu'un module souhaite communiquer le résultat d'un calcul vers l'extérieur, il utilise une méthode d'un de ses ports. Cette méthode transmet automatiquement l'appel au canal externe auquel le port est relié. Autrement dit, un port, instance de la classe sc_port redirige les appels de méthode d'interface vers le canal auquel le port est lié. Un port définit un ensemble de services (relatifs à son type) requis par le module contenant le port. Si un module doit appeler une fonction membre appartenant à un canal extérieur au module lui-même, cet appel doit être effectué en utilisant une méthode d'interface via un port du module. Cependant, un appel à une fonction membre appartenant à un canal instancié dans le module actuel peut être effectué directement. Cela est connu sous le nom d'accès au canal sans port (portless channel access). Si un module doit appeler une fonction membre appartenant à une instance de canal dans un module enfant, cet appel doit être effectué à travers un second type de port appelé export du module enfant. La classe sc_export permet alors à un module enfant de fournir une interface à son module parent. Les méthodes de l'interface d'exportation transmettent les appels au canal lié. Les instances de ports peuvent être liées à des instances de canaux, à d'autres instances de ports ou à des exports. Les instances exports peuvent être liées à des instances de canaux ou à d'autres instances exports, mais pas à des instances de ports. La liaison d'un port est une relation asymétrique, tout comme la liaison d'un export. Si un port est lié à un canal, cela ne signifie pas que le canal est lié au port. La liaison des ports/exports peut se produire uniquement durant la phase d'élaboration (voir la sous-section 1.5.1.3).

**Canal** Les canaux implémentent la communication entre les modules. Ils doivent hériter soit de la classe sc_prim_channel, soit de la classe sc_channel, ce qui les définit respectivement comme primitifs ou hiérarchiques. Cette distinction entre ces deux

classes de base SystemC réside dans leurs capacités et fonctionnalités distinctes. Les canaux primitifs sont conçus pour fournir des communications simples et rapides. Ils ne contiennent pas de hiérarchie ni de ports, et ils n'intègrent pas de processus de simulation. En revanche, les canaux hiérarchiques peuvent avoir leurs propres ports et processus, et ils peuvent contenir une hiérarchie, comme leur nom l'indique. En réalité, les canaux hiérarchiques sont simplement des modules qui implémentent une ou plusieurs interfaces. Ils sont destinés à modéliser des bus de communication complexes tels que PCI (Peripheral Component Interconnect), HyperTransport [8] , AMBA (Advanced Microcontroller Bus Architecture) ou AXI (Advanced eXtensible Interface).

**Processus** Le comportement d'un module SystemC est spécifié par un ou plusieurs processus SystemC. Une instance de processus est un objet d'une classe définie par l'implémentation, dérivée de la classe sc_object et créée par l'une des trois macros SC_METHOD, SC_THREAD ou SC_CTHREAD, ou en appelant la fonction sc_spawn. Le but des macros de processus est d'enregistrer la fonction associée auprès du noyau de manière à ce que l'ordonnanceur puisse rappeler cette fonction membre pendant la simulation. Le terme processus fait référence soit à une instance de processus, soit à la fonction membre qui est associée à une instance de processus lors de sa création. Le sens est précisé par le contexte. Chaque processus dispose d'une liste de sensibilité qui contient des événements SystemC. Un événement représente quelque chose qui se produit pendant l'exécution, par exemple un changement de valeur sur un port d'entrée. La sensibilité d'une instance de processus correspond à l'ensemble des événements et des délais qui peuvent potentiellement provoquer la reprise ou le déclenchement du processus. La sensibilité statique d'une instance de processus non engendrée est déterminée pendant l'élaboration. Pour une instance de processus engendrée, la sensibilité statique est fixée lors de l'appel à la fonction sc_spawn. La sensibilité dynamique d'une instance de processus peut changer au fil du temps sous le contrôle du processus lui-même. Une instance de processus est dite sensible à un événement si cet événement a été ajouté à sa sensibilité statique ou dynamique.

- **Processus SC_METHOD :** il est démarré par le noyau de simulation SystemC chaque fois qu'un événement auquel il est sensible se produit. Il s'exécute intégralement avant de rendre le contrôle au noyau de simulation. De plus, il s'exécute en "temps nul", ce qui signifie que le temps global de simulation reste inchangé après son exécution. Un processus de méthode ne peut être terminé qu'en appelant la méthode *kill* sur un hundle associé à ce processus.
- **Processus SC_THREAD :** il n'est lancé qu'une fois par le noyau de simulation et ne se termine qu'à la fin complète de la simulation. Un processus SC_THREAD peut être suspendu en des points statiquement définis dans le code SystemC, identifiables par l'appel de la fonction d'attente wait(). Le noyau de simulation reprend le processus SC_THREAD lorsque l'un des événements de sa liste de sensibilité se produit ou lorsque le délai spécifié dans la fonction wait() est écoulé. Une fois repris, le processus s'exécute jusqu'à atteindre le prochain appel à la fonction wait(), où il est alors suspendu de nouveau.
- **Processus SC_CTHREAD :** pour clocked thread process, est un type spécial de SC_THREAD qui ne réagit qu'à un certain type d'événement provenant d'un port d'entrée horloge (sensibilité statique). Il doit être statique et ne peut être engendré.
- **Fonction sc_spawn :** elle est utilisée pour créer une instance de processus engendré, statique ou dynamique. Le processus ou le module à partir duquel sc_spawn

8. HyperTransport a été largement utilisé dans les systèmes informatiques haut de gamme avant l'adoption généralisée de l'interface PCIe (Peripheral Component Interconnect Express).

est appelé est le parent du processus engendré. La fonction sc_spawn peut être appelée pendant l'élaboration, auquel cas le processus engendré est un fils de l'instance du module au sein de laquelle la fonction sc_spawn est appelée. Si la fonction sc_spawn est invoquée à partir de la fonction sc_main, alors le processus engendré est un objet de niveau supérieur. Egalement, la fonction sc_spawn peut être invoquée durant la simulation à partir d'une méthode, d'un thread ou d'un thread cadencé. Ainsi, le processus créé est un fils du processus qui a fait appel à la fonction sc_spawn.

Un processus est dit **statique** s'il est créé pendant la construction de la hiérarchie des modules ou à partir de la fonction de rappel (callback) *before_end_of_elaboration*. La hiérarchie des modules est l'ensemble total des instances de modules construites pendant l'élaboration. Le terme est parfois utilisé pour inclure tous les objets instanciés à l'intérieur de ces modules pendant l'élaboration. Par opposition, un processus est dit **dynamique** s'il est créé à partir de la fonction de rappel *end_of_elaboration* ou pendant la simulation.

Un processus **non-engendré (unspawned)** est un processus créé en invoquant l'une des trois macros SC_METHOD, SC_THREAD ou SC_CTHREAD. Un processus non-engendré est généralement un processus statique, mais il serait un processus dynamique s'il était invoqué à partir de la fonction de rappel end_of_elaboration. Un processus **engendré (spawned)** est un processus créé en appelant la fonction sc_spawn (voir l'annexe B). Un processus engendré est généralement un processus dynamique, mais il serait un processus statique si la fonction sc_spawn est appelée avant la fin de l'élaboration.

Donc, un processus dynamique démarre son exécution à un moment donné pendant la simulation, plutôt que lors de la construction statique de la hiérarchie des modules. Cela offre une flexibilité supplémentaire pour la création et la gestion des processus, en permettant la génération de processus en fonction de certaines conditions ou événements au cours de la simulation. Nous portons un grand intérêt à ce type de processus pour l'implémentation des composants reconfigurables dynamiquement.

#### 1.5.1.3 Sémantique d'exécution d'un modèle SystemC

Le processus d'exécution d'une application SystemC se déroule en deux phases : l'élaboration et la simulation. L'objectif principal de l'élaboration est de créer, au sein du noyau, les structures de données internes nécessaires pour supporter la sémantique de la simulation. Au cours de cette phase, les éléments de la hiérarchie des modules (modules, ports, canaux primitifs et processus) sont construits, et les ports ainsi que les exports sont liés aux canaux. Etant donné que ces actions ne peuvent se produire que pendant l'élaboration, SystemC ne prend pas en charge la création ou la modification dynamique de la hiérarchie des modules pendant la simulation, bien qu'il prenne en charge les processus dynamiques. La simulation, quant à elle, concerne l'exécution de l'ordonnanceur, qui est une partie du noyau. L'objectif principal de l'ordonnanceur est de déclencher ou de reprendre l'exécution des processus fournis par l'utilisateur dans le cadre de l'application. L'ordonnanceur est piloté par les événements, ce qui signifie que les processus sont exécutés en réponse à la survenance d'événements. Les événements se produisent (sont notifiés) à des moments précis du temps de simulation. Le temps de simulation est une quantité entière initialisée à zéro au début de la simulation et augmente de manière monotone pendant la simulation. Le temps de simulation et les intervalles de temps sont représentés par la classe sc_time.

Les phases d'élaboration et de simulation doivent se dérouler dans l'ordre suivant :

1. Elaboration : construction de la hiérarchie des modules.

2. Elaboration : appels à la fonction *before_end_of_elaboration*.

   Le but de cette fonction membre est de permettre à une application d'exécuter des actions durant l'élaboration qui dépendent des propriétés globales de la hiérarchie des modules et qui nécessitent également de modifier cette hiérarchie. Des exemples incluent entre autre l'instanciation de modules de niveau supérieur (top-level) pour surveiller des événements enfouis dans la hiérarchie ; l'instantiation des classes sc_module, sc_port, sc_export, sc_prim_channel ; liaison des ports/exports ; utilisation des macros SC_METHOD, SC_THREAD, and SC_CTHREAD ; appels à la fonction sc_spawn pour la création de processus statiques ; les appels aux fonctions membres de contrôle de processus *suspend, resume, disable, enable, sync_reset_on* et *sync_reset_off* de la classe sc_process_handle ; etc.

3. Elaboration : appels à la fonction *end_of_elaboration*.

   Cette fonction membre est appelée à la toute fin de l'élaboration, après que tous les appels à la fonction before_end_of_elaboration aient été terminés et après l'achèvement de toute instanciation ou liaison de ports effectuée par ces appels et avant de commencer la simulation. Son but est de permettre à une application d'exécuter des actions de maintenance à la fin de l'élaboration et qui n'ont pas besoin de modifier la hiérarchie des modules. Des exemples incluent la vérification des règles de conception ; des actions dépendants du nombre de fois qu'un port est lié ; l'impression de messages de diagnostic concernant la hiérarchie des modules ; l'instanciation d'objets de classes dérivées de la classe sc_object, à l'exception des classes sc_module, sc_port, sc_export et sc_prim_channel ; l'utilisation des macros SC_METHOD et SC_THREAD ; les appels à la fonction sc_spawn pour créer des processus dynamiques engendrés ; les appels aux fonctions membres de contrôle de processus *suspend, resume, disable, enable, sync_reset_on* et *sync_reset_off* de la classe sc_process_handle ; etc. Les constructions suivantes ne doivent pas être utilisées directement ou indirectement dans le rappel end_of_elaboration : l'instanciation d'objets des classes sc_module, sc_port, sc_export et sc_prim_channel ; la liaison des ports/export ; l'utilisation des macros SC_CTOR et SC_CTHREAD ; etc.

4. Simulation : appels à la fonction *start_of_simulation*.

   Le but de la fonction membre start_of_simulation est de permettre à une application d'exécuter des actions de maintenance au début de la simulation. Des exemples incluent l'ouverture de fichiers de stimuli et de réponses, ainsi que l'impression de messages de diagnostic ; l'instanciation d'objets de classes dérivées de la classe sc_object, à l'exception des classes sc_module, sc_port, sc_export et sc_prim_channel ; les appels à la fonction sc_spawn pour créer des processus dynamiques engendrés ; les appels aux fonctions membres de contrôle de processus *suspend, resume, disable, enable, sync_reset_on* et *sync_reset_off* de la classe sc_process_handle ; etc. Les constructions suivantes ne doivent pas être utilisées directement ou indirectement dans le rappel start_of_simulation : l'instanciation d'objets des classes sc_module, sc_port, sc_export, sc_prim_channel ; la liaison des ports/export ; l'utilisation des macros SC_CTOR, SC_METHOD, SC_THREAD, and SC_CTHREAD ;

5. Simulation : phase d'initialisation.

   Les trois étapes suivantes sont exécutées dans l'ordre : l'exécution de tous les appels en attente à la fonction *update* ; l'ajout de chaque instance des processus sc_method et sc_thread présente dans la hiérarchie des objets à l'ensemble des processus exécutables, mais en excluant les instances de processus pour lesquelles

la fonction *dont_initialize* a été appelée, ainsi que les processus sc_cthread ; l'exécution de la phase de notification delta [9].

6. Simulation : évaluation, mise à jour, notification de delta et phase de notification temporisée (répétées).

   L'évaluation commence à la fin de la phase de notification delta, elle permet à partir de l'ensemble des processus exécutables, de sélectionner une instance de processus, de la retirer de l'ensemble, et seulement ensuite de déclencher ou reprendre son exécution. La phase de mise à jour permet d'exécuter tous les appels en attente à la fonction *update* qui découlent des appels à la fonction *request_update* effectués lors de la phase d'évaluation précédente ou lors de l'élaboration. S'il n'y a plus d'appels en attente à la fonction *update*, la phase de notification delta est exécutée. Si, à la fin de la phase de notification delta, l'ensemble des processus exécutables n'est pas vide, on revient à la phase d'évaluation. Si des notifications temporisées en attente ou des délais d'expiration existent alors il faut : le temps de simulation est avancé jusqu'au moment de la notification temporisée en attente la plus proche ou du délai d'expiration ; les instances de processus qui sont sensibles aux événements notifiés et aux délais d'expiration sont déterminées et ajoutées à l'ensemble des processus exécutables ; enfin, toutes ces notifications et délais sont retirées de l'ensemble des notifications temporisées et des délais d'expiration. Si, à la fin de la phase de notification temporisée, l'ensemble des processus exécutables n'est pas vide, on revient à la phase d'évaluation. Si aucune notification temporisée en attente ou aucun délai n'est présent, cela signifie que la fin de la simulation a été atteinte. Par conséquent, il faut sortir de l'ordonnanceur.

7. Simulation : appels à la fonction *end_of_simulation*.

   L'implémentation doit appeler la fonction membre end_of_simulation au moment où l'ordonnanceur s'arrête à cause de l'appel de la fonction *sc_stop* pendant la simulation ou à la toute fin de la simulation si celle-ci est initiée sous le contrôle direct du noyau. Les rappels end_of_simulation ne doivent être appelés qu'une seule fois, même si la fonction *sc_stop* est appelée plusieurs fois. La fonction end_of_simulation est utilisée entre autre pour la fermeture des fichiers de stimuli et de réponses, ainsi que l'impression de messages de diagnostic. L'intention est qu'une implémentation qui initie l'élaboration et la simulation sous le contrôle direct du noyau (en l'absence des fonctions *sc_main* et *sc_start*) effectue les rappels à end_of_simulation à la toute fin de la simulation, que la fonction *sc_stop* ait été appelée ou non.

8. Simulation : destruction de la hiérarchie des modules.

Plus de détails concernant l'utilisation des constructions SystemC et le déroulement des phases d'élaboration et de simulation sont fournis dans [IEEE, 2023].

### 1.5.2 Concepts de la modélisation TLM

Nous avons précédemment introduit le langage SystemC, utilisable pour modéliser des systèmes matériels à différents niveaux d'abstraction. Dans ce qui suit, nous nous concentrons sur la modélisation au niveau transactionnel.

TLM met en avant le concept de séparation entre la communication et le calcul au sein d'un système. En effet, les composants sont modélisés comme des modules avec un ensemble de processus concurrents qui calculent et représentent leur comportement. Ces modules échangent des données sous forme de transactions via un canal abstrait.

9. La notification delta est un mécanisme dans les simulateurs de systèmes qui permet de traiter les événements de manière instantanée dans le même cycle de simulation sans avancer le temps.

Les interfaces TLM sont implémentées au sein des canaux pour encapsuler les protocoles de communication. Pour établir la communication, un processus doit simplement accéder à ces interfaces via les ports des modules. Essentiellement, l'interface est la partie qui sépare la communication du calcul au sein d'un système TLM.

Les classes TLM-2.0 sont superposées à la bibliothèque de classes SystemC comme illustré à la figure 1.15 . TLM-2.0 a une structure en couches, les couches inférieures étant plus flexibles et générales, tandis que les couches supérieures sont spécifiques à la modélisation des bus. Pour une interopérabilité maximale, et en particulier pour la modélisation des bus mappés en mémoire, il est recommandé d'utiliser ensemble les interfaces principales, les sockets, la charge utile générique (generic payload) et le protocole de base de TLM-2.0. Ces classes sont collectivement connues sous le nom de couche d'interopérabilité.

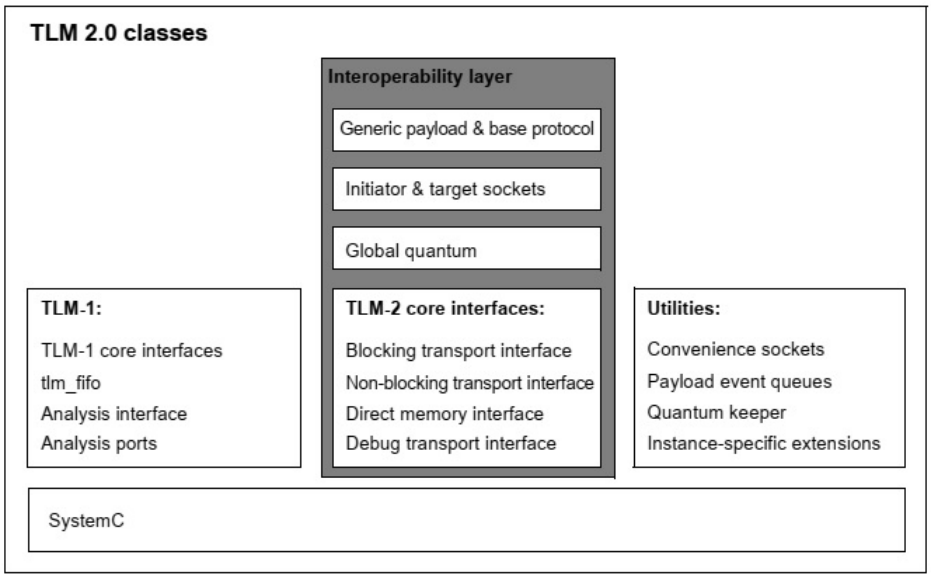



FIGURE 1.15 – *Les classes TLM 2.0 [IEEE, 2023].*

TLM-2.0 reconnaît explicitement la diversité des cas d'utilisation pour la modélisation au niveau des transactions : développent logiciel, analyse de l'architecture matérielle, évaluation des performances logicielles, et vérification matérielle. Cependant, plutôt que de définir un niveau d'abstraction pour chaque cas d'utilisation, TLM-2.0 distingue les interfaces (API) des styles de codage. La norme TLM-2.0 propose un ensemble d'interfaces considérées comme des mécanismes de programmation de bas niveau pour implémenter des modèles au niveau des transactions et décrit plusieurs styles de codage adaptés à divers cas d'utilisation, sans y être restreints.

Les interfaces principales de TLM-2.0 permettent de transférer des transactions entre initiateurs et cibles. Un **initiateur** est un module capable de lancer des **transactions**, c'est-à-dire de créer de nouveaux objets de transaction et de les transmettre en appelant une méthode de l'une des interfaces principales. Une **cible** est un module qui sert de destination finale pour une transaction. Par exemple, dans une transaction d'écriture *write(adress, data)*, un initiateur (un processeur par exemple) écrit des données vers une cible (une mémoire). Dans une transaction de lecture *read(adress, data)*, un initiateur lit des données à partir d'une cible. Un **composant d'interconnexion** est un module qui accède à une transaction sans agir comme initiateur ou cible pour cette transaction ; des exemples typiques incluent les arbitres et les routeurs. Les rôles d'initiateur, de module d'interconnexion et de cible peuvent changer dynamiquement. Par exemple, un même

composant peut servir d'interconnexion pour certaines transactions et de cible pour d'autres.

#### 1.5.2.1 Interfaces de base TLM-2.0

Outre les interfaces principales de TLM-1, TLM-2.0 intègre des interfaces de transport bloquantes et non bloquantes, une interface de mémoire directe (Direct Memory Interface - DMI), ainsi qu'une interface de transport de débogage.

**Interfaces de transport** Les interfaces de transport sont les interfaces principales utilisées pour transporter des transactions entre les initiateurs, les cibles et les composants d'interconnexion. Deux types d'interfaces de transport existent : **bloquantes** et **non bloquantes**. Dans le cadre de notre travail, nous avons porté une attention particulière à ces deux types d'interfaces, car elles sont fondamentales pour la mise en œuvre des styles de codage TLM. L'interface de transport bloquante ne peut modéliser que le début et la fin d'une transaction, la transaction étant achevée au cours d'un seul appel de fonction. L'interface de transport non bloquante permet à une transaction d'être décomposée en plusieurs points de synchronisation, et nécessite généralement plusieurs appels de fonctions pour une seule transaction.

Les interfaces de transport bloquantes et non bloquantes prennent en charge l'annotation temporelle et le découplage temporel, mais seule l'interface de transport non bloquante prend en charge plusieurs phases au cours de la durée de vie d'une transaction et retourne une valeur indiquant si le chemin de retour a été utilisé ou non.

L'interface de transport bloquante est destinée à supporter le style de codage *loosely-timed*. Elle est appropriée lorsqu'un initiateur souhaite réaliser une transaction avec une cible au cours d'un seul appel de fonction, les seuls points d'intérêt temporel étant ceux qui marquent le début et la fin de la transaction, l'interface de transport bloquante n'utilise que le forward path de l'initiateur à la cible. La méthode de transport bloquante **b_transport()** peut retourner immédiatement (c'est-à-dire, dans la phase d'évaluation actuelle de SystemC) ou peut céder le contrôle à l'ordonnanceur et ne retourner à l'initiateur qu'à un moment ultérieur du temps de simulation, comme le montre la figure 1.16.

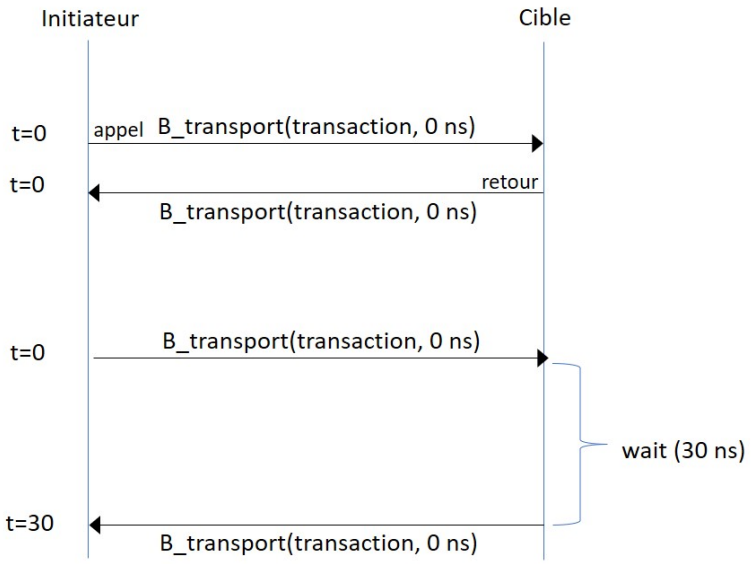



FIGURE 1.16 – *Communication via une interface de transport bloquante.*

L'interface de transport non bloquante est conçue pour prendre en charge le style de codage *approximately-timed*. Elle est appropriée lorsqu'il est nécessaire de modéliser la séquence détaillée des interactions entre l'initiateur et la cible pendant chaque transaction. Autrement dit, elle permet de décomposer une transaction en plusieurs phases, chaque phase étant associée à un point de synchronisation. Associées au protocole de base, quatre phases sont à distinguer : BEGIN_REQ (début de l'appel), END_REQ (fin de l'appel), BEGIN_RESP (début du retour), et END_RESP (fin du retour). L'interface de transport non bloquante utilise à la fois le forward path de l'initiateur à la cible et le backward path de la cible à l'initiateur ; il existe donc deux interfaces distinctes : **tlm_fw_nonblocking_transport_if** et **tlm_bw_nonblocking_transport_if** fournissant les fonctions membres **nb_transport_fw** et **nb_transport_bw** respectivement. Les données passées à et renvoyées depuis nb_transport sont représentées avec la notation <return, phase, delay>. Ici, return correspond à la valeur renvoyée de l'appel de fonction à savoir : TLM_ACCEPTED, TLM_UPDATED ou TLM_COMPLETED ; phase représente la valeur de l'argument de phase, et delay indique la valeur de l'argument sc_time. La figure 1.17 représente un exemple de communication en utilisant une interface de transport non bloquante avec chemin de retour.

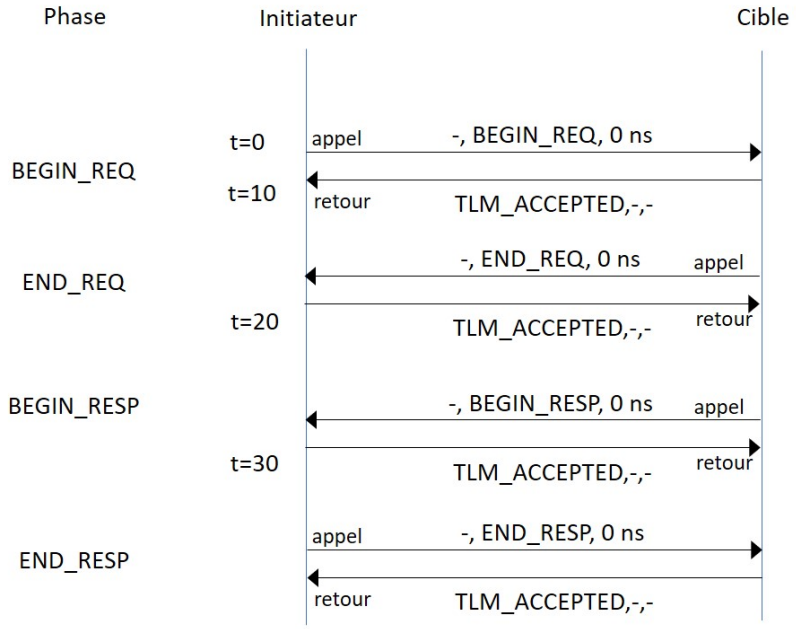



FIGURE 1.17 – *Communication via une interface de transport non bloquante avec chemin de retour.*

**Direct Memory Interface DMI** L'interface DMI permet à un initiateur d'accéder directement à une zone de mémoire appartenant à une cible à l'aide d'un pointeur au lieu de passer par une interface de transport. Cette fonctionnalité offre un potentiel significatif d'accélération de la simulation lors des accès mémoire entre l'initiateur et la cible, car une fois établie, elle évite le parcours habituel des multiples appels b_transport ou nb_transport de l'initiateur à travers les composants d'interconnexion jusqu'à la cible.

**Interface de transport de débogage** Cette interface fournit un moyen de lire et d'écrire dans la mémoire d'une cible, sur le même chemin d'appel de l'initiateur à la cible que celui utilisé par l'interface de transport, mais sans délais, attentes wait, notifications d'événements ou autres effets associés à une transaction régulière. En d'autres termes,

l'interface de transport de débogage est non intrusive. Par exemple, elle pourrait permettre à un initiateur de prendre une image instantané du contenu de la mémoire système pendant la simulation à des fins de diagnostic, ou d'initialiser une zone de la mémoire système à la fin de l'élaboration.

#### 1.5.2.2 Styles de codage

Un style de codage est un ensemble de pratiques de programmation qui fonctionnent bien ensemble, plutôt qu'un niveau d'abstraction spécifique ou une interface de programmation logicielle. Deux styles de codage sont pris en compte dans TLM-2.0 : Loosely-Timed et Approximatively-Timed.

**Style de codage Loosely-Timed** Ce style de codage fait appel à l'interface de transport bloquante comme il a été mentionné précédemment. Cette interface permet d'associer seulement deux points temporels à chaque transaction, qui correspondent à l'appel et au retour de la fonction de transport bloquante. Dans le contexte du protocole de base, le premier point temporel signale le début de la requête, et le second indique le début de la réponse. Le style de codage est approprié pour le développement logiciel utilisant un modèle de plateforme virtuelle d'un MPSoC [10], où le contenu logiciel peut inclure un ou plusieurs systèmes d'exploitation. Il supporte également le *découplage temporel*, où il est permis à chaque processus SystemC d'avancer dans une *distorsion temporelle* locale sans faire progresser le temps de simulation jusqu'à ce qu'ils atteignent un point où ils doivent se synchroniser avec le reste du système. Ce découplage temporel peut conduire à une simulation extrêmement rapide pour certains systèmes, car il améliore la localité des données et du code et diminue la surcharge de l'ordonnancement du simulateur. Chaque processus a la possibilité de s'exécuter pendant un quantum de temps (défini par l'application) avant de passer au suivant, ou peut choisir de céder le contrôle lorsqu'il atteint un point de synchronisation explicite.

**Style de codage Approximatively-Timed** Le style de codage à synchronisation approximative est pris en charge par l'interface de transport non bloquante, ce qui le rend adapté à des cas d'utilisation comme l'exploration architecturale et l'analyse des performances. Cette interface permet l'annotation temporelle et la gestion de plusieurs phases et points de synchronisation tout au long d'une transaction. Dans ce style, une transaction est divisée en plusieurs phases, avec un point de synchronisation explicite marquant chaque transition. Pour le protocole de base, il y a exactement quatre points de synchronisation : le début et la fin de la requête, ainsi que le début et la fin de la réponse. Certains protocoles spécifiques peuvent nécessiter l'ajout de points de synchronisation supplémentaires, ce qui pourrait entraîner une incompatibilité directe avec la charge utile générique. Comme les processus cèdent régulièrement le contrôle à l'ordonnanceur pour permettre l'avancement du temps de simulation, le style de codage à synchronisation approximative est censé simuler beaucoup plus lentement que le style de codage à synchronisation lâche.

#### 1.5.2.3 Sockets initiateur et cible

Un socket combine un port et un export ; un socket initiateur comporte un port pour le forward path et un export pour le backward path, tandis qu'un socket cible dispose

10. Un MPSoC (Multiprocessor System on Chip) est un circuit intégré qui regroupe plusieurs processeurs (CPU, GPU, DSP, etc.) ainsi que d'autres composants matériels (mémoire, accélérateurs matériels, interfaces de communication, etc.) sur une seule puce.

d'un export pour le forward path et d'un port pour le backward path. Les sockets surchargent également les opérateurs de liaison des ports SystemC afin de lier à la fois le port et l'export à l'export et au port du socket opposé. Les sockets initiateurs et cibles sont tous deux implémentés en utilisant une hiérarchie d'héritage en C++. En général, seules les classes les plus spécialisées, **tlm_initiator_socket** et **tlm_target_socket**, sont utilisées directement par les applications. Les sockets ne peuvent être liés entre eux que s'ils partagent le même type de protocole. Par défaut, le type de protocole est défini par la classe *tlm_base_protocol_types*. Les sockets présentent l'avantage de regrouper les interfaces de transport, de mémoire directe et de débogage pour les forward path et backward path dans un seul objet en offrant des méthodes permettant de lier le port et l'export en une seule opération.

#### 1.5.2.4 Generic payload

**tlm_generic_payload** est le type de transaction par défaut pour les classes de socket. il joue un rôle important pour assurer l'interopérabilité entre les modèles au niveau des transactions. Le generic payload a deux principales fonctions étroitement liées : premièrement, il peut être utilisé comme un type de transaction général pour la modélisation abstraite de bus mappés en mémoire, lorsqu'on ne s'intéresse pas aux détails spécifiques d'un protocole de bus particulier, permettant ainsi une interopérabilité immédiate entre les modèles standardisés ; deuxièmement, le generic payload peut servir de base pour modéliser une large gamme de protocoles spécifiques de manière plus détaillée permettant de créer des passerelles entre différents protocoles lorsqu'ils sont tous basés sur le même type de generic payload. Le generic payload contient plusieurs attributs essentiels pour décrire une transaction, notamment :

- commande : indique le type de commande de la transaction (lecture, écriture) ;
- adresse : l'adresse mémoire cible de la transaction ;
- données : un pointeur vers les données à lire ou à écrire ;
- octets activés : spécifie quels octets dans les données sont valides ;
- largeur de streaming : indique la taille de la transaction pour le streaming de données ;
- statut de réponse : indique le résultat de la transaction (succès, erreur, etc.) ;
- indice DMI : indique la possibilité d'utilisation de l'accès direct à la mémoire ;
- extensions : permet d'ajouter des informations supplémentaires à la transaction.

Le generic payload propose également deux commandes : la lecture et l'écriture, implémentées par le module cible en transférant les données vers ou depuis une zone de données dans l'initiateur.

## 1.6 Conclusion

Dans ce chapitre, nous avons mis en lumière les spécificités des architectures reconfigurables et des SoCs dynamiquement reconfigurables en particulier, en soulignant leur potentiel à combiner flexibilité, performance et adaptabilité, notamment grâce à l'exploitation des architectures FPGA. Nous avons également abordé des aspects importants de la reconfiguration dynamique et partielle, en particulier le mécanisme de contrôle, le processus de reconfiguration proprement dit, ainsi que le flot de conception associé, tout en mettant en évidence les avantages significatifs qu'offre cette fonctionnalité. Par ailleurs, l'usage du langage SystemC a été exploré en tant qu'approche de modélisation haut niveau permettant de décrire, simuler et valider efficacement les systèmes reconfigurables. SystemC, de par sa capacité à être utilisé pour la modélisation ESL, l'exploration architecturale, l'évaluation des performances, etc. s'impose comme

un outil central dans les phases amont du cycle de conception des SoCs, en particulier pour le prototypage rapide et la validation fonctionnelle avant la synthèse matérielle.

Toutefois, la complexité croissante des systèmes reconfigurables impose une évolution des méthodologies de conception traditionnelles vers des approches plus abstraites, automatisées et structurées. C'est dans cette perspective que s'inscrit l'ingénierie dirigée par les modèles (IDM), qui propose une formalisation systématique de la conception à travers l'utilisation de modèles à différents niveaux d'abstraction et des transformations automatisant le processus de conception. Le chapitre suivant présentera ainsi les principes fondamentaux de l'IDM, tout en explorant une relecture de la DPR à travers le prisme de cette approche.

# 2

# L'Ingénierie Dirigée par les Modèles

Sommaire

## 2.1 Introduction

Face à une évolution exponentielle de la taille et de la complexité des systèmes logiciels [Rousseau et al., 2020], il est fort opportun de s'appuyer sur des représentations du problème ainsi que de ses possibles solutions à différents niveaux d'abstraction et selon de nombreux points de vue permettant de couvrir efficacement toutes les préoccupations pertinentes. C'est l'essence même de la modélisation. Aujourd'hui, un système complexe est modélisé selon plusieurs aspects (structurel, comportemental, fonctionnel, etc.) pouvant varier en termes d'abstraction et de précision en faisant passer progressivement les modèles d'une simple ébauche à une définition rigoureuse de produits.

L'ingénierie dirigée par les modèles (IDM ou MDE en anglais pour Model Driven Engineering) apparait comme une solution potentielle au problème du développement des systèmes complexes. En effet, elle permet de capitaliser les savoir-faire de conception et de validation de tels systèmes par tissage et transformation de modèles dédiés. Grâce à des modèles indépendants des détails techniques des plateformes cibles, il devient possible de générer automatiquement une grande partie du code des applications et garantir ainsi un gain de productivité. En substance, l'IDM prône l'utilisation systématique des modèles pour mécaniser une partie du processus de développement que les concepteurs expérimentés suivent à la main [Jézéquel et al., 2012]. Une telle mise en œuvre fait passer les modèles d'un état purement contemplatif à un état pleinement productif, interprétable et exécutable par une machine.

Ce chapitre expose les principes généraux de l'IDM, en s'appuyant sur les concepts clés de système, de modèle et de métamodèle. L'approche Model-Driven Architecture (MDA), considérée comme une déclinaison de l'IDM, est ensuite présentée. Une taxonomie de la transformation de modèles ainsi que les principales approches associées sont également abordées. Par ailleurs, les langages spécifiques à la modélisation font l'objet d'une analyse approfondie. Enfin, la gestion de la DPR est traitée comme une préoccupation transverse, en soulignant la convergence entre l'adaptation dynamique des logiciels et la DPR, ainsi que les paradigmes de conception qui en découlent, avant de conclure le chapitre.

## 2.2 Principes généraux

L'IDM est un paradigme du génie logiciel qui promeut une approche centrée sur le modèle pour le développement logiciel, où les modèles sont utilisés pour spécifier, concevoir, tester et générer du code pour des systèmes logiciels complexes [Jiménez-Pastor et al., 2017]. Bien qu'elle ait été adoptée avec succès dans de nombreux domaines, notamment l'industrie automobile, l'aérospatial, les télécommunications et les systèmes d'information commerciaux, son utilisation n'est pas à son plein potentiel et l'usage de modèles pour générer automatiquement des systèmes est encore relativement rare [Mussbacher et al., 2014].

L'idée du « tout modèle » omniprésente dans l'IDM vise à fournir un grand nombre de modèles pour exprimer séparément chacune des préoccupations des utilisateurs, des concepteurs, des architectes, etc. selon leurs domaines de compétences ; rampant ainsi les liens avec le principe du « tout objet » dominant jusque-là l'ingénierie du logiciel.

### 2.2.1 Système

Un système est un ensemble de parties et de relations entre ces parties qui peuvent être organisées pour atteindre un objectif donné. Un système peut inclure des parties matérielles, logicielles et humaines, une entreprise, une fédération d'entreprises, un

processus métier, une combinaison de parties de différents systèmes ou une fédération de systèmes. Dans le contexte de l'IDM, un système est défini comme un concept générique pour désigner une application logicielle, une plateforme logicielle ou tout autre artefact logiciel [da Silva, 2015].

### 2.2.2 Modèle

Un modèle est une abstraction d'un système, modélisé sous la forme d'un ensemble de faits construits dans une intention particulière. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé [Combemale, 2008].

Un modèle représente donc un système selon un certain point de vue, à un niveau d'abstraction facilitant par exemple la conception et la validation de cet aspect particulier du système [Jézéquel et al., 2012]. La relation « représentationDe » lie un modèle au système qu'il représente. Selon le principe de substituabilité, un modèle doit capturer les informations nécessaires et suffisantes pour permettre de répondre aux questions que l'on se pose sur un aspect du système qu'il est censé représenter, exactement de la même façon que le système lui-même aurait répondu. Un modèle doit posséder certaines propriétés le distinguant des autres artefacts [Muller et al., 2012] :

— Mapping : un modèle est basé sur un système d'origine ; tout objet/phénomène de ce dernier mappé dans le modèle doit être identifiable.
— Réduction : un modèle est une version simplifiée du système ne comprenant que les propriétés les plus pertinentes.
— Pragmatisme : un modèle doit être utile ; il devrait pouvoir remplacer le système d'origine dans un certain but.

D'autres propriétés peuvent être vérifiées, notamment : l'abstraction, la compréhensibilité, la précision, la prédictibilité et le coût moindre [Selic, 2003].

Pour qu'un modèle soit productif (manipulable par une machine), le langage dans lequel ce modèle est exprimé doit être explicitement défini. Un tel langage est défini par un modèle appelé métamodèle.

### 2.2.3 Métamodèle

Un métamodèle est un modèle qui définit la structure d'un langage de modélisation [da Silva, 2015]. Il permet de capitaliser un domaine de connaissance en définissant des concepts et des relations entre concepts. La relation liant le modèle au langage de modélisation est appelée « conformeA ». Un modèle est dit conforme à un métamodèle s'il est possible d'instancier tous les éléments du modèle à partir d'éléments du métamodèle en respectant l'ensemble des propriétés exprimées sur le métamodèle. Ce dernier est exprimé dans un langage appelé méta-métamodèle. La figure 2.1 illustre les relations entre système, modèle et métamodèle.

## 2.3 L'approche MDA

L'architecture dirigée par les modèles, ou Model Driven Architecture (MDA) en anglais, est une initiative de l'OMG dont l'idée est de fournir à la fois une architecture et une démarche de développement logiciel respectant les principes de l'IDM. Le MDA vise à mettre en valeur les qualités intrinsèques des modèles, telles que pérennité, productivité et prise en compte des plateformes d'exécution [Jézéquel et al., 2012]. L'architecture est basée sur plusieurs standards de modélisation, notamment : UML, MOF (Meta-Object Facility), CWM (Common Warehouse Meta-mode) et XMI (XML Metadata Interchange). Le long du cycle de développement, les modèles sont décrits en UML en

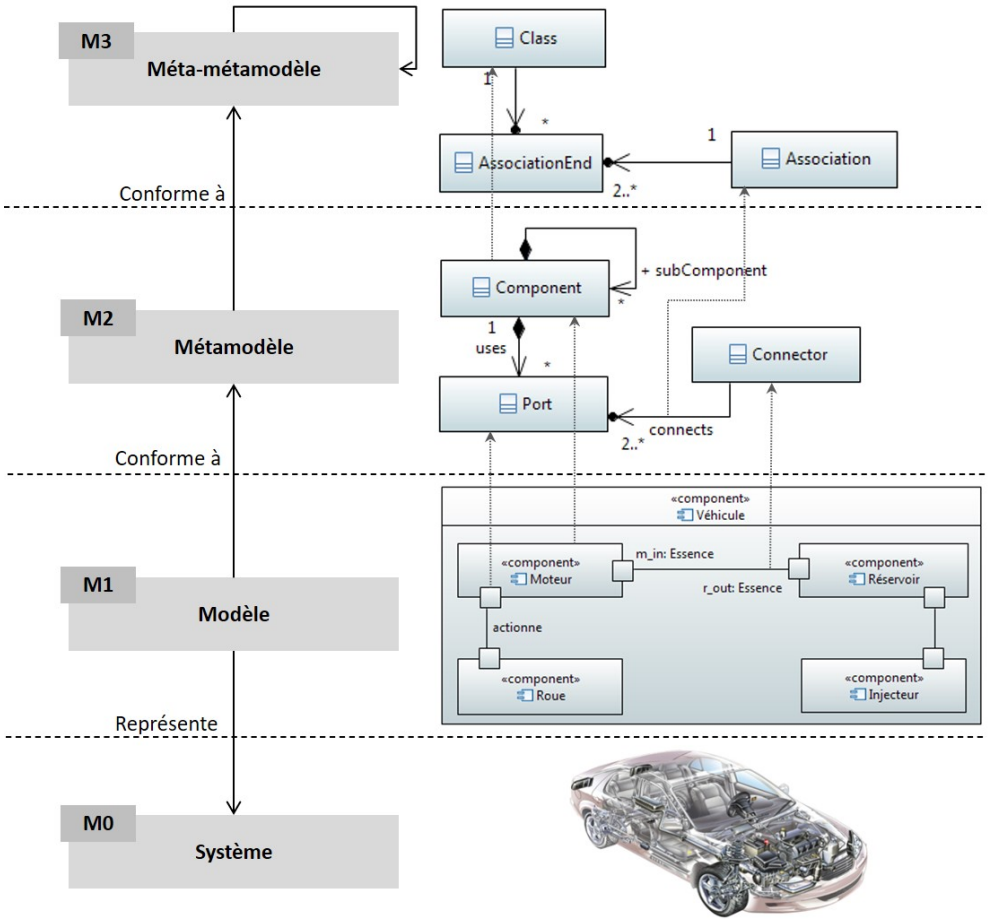



FIGURE 2.1 – *Relations entre système, modèle, métamodèle et méta-métamodèle.*

séparant les spécifications fonctionnelles d'un système des détails de son implémentation sur une plate-forme donnée. Pour cela, le MDA préconise l'élaboration de modèles selon trois niveaux d'abstraction :

— Exigences : le modèle CIM (Computational Independant Model) littéralement « modèle indépendant de la programmation », permet de décrire les besoins fonctionnels de l'application dans son environnement en terme de services offerts et requis sans la moindre considération informatique. Une fois validés par le client de l'application, les modèles d'exigences sont les premiers modèles pérennes fournissant une base contractuelle variant peu. Ainsi à travers des liens de traçabilité avec d'autres modèles, un lien peut être établi depuis les exigences jusqu'à l'implémentation. En UML, un diagramme de cas d'utilisation peut être utilisé pour représenter un modèle CIM.

— Analyse et conception : le modèle PIM (Platform Independent Model) ou modèle indépendant des plates-formes d'implémentation, qualifié aussi de modèle d'analyse et de conception abstraite, permet de donner une vision structurelle et dynamique de l'application sans connaissance aucune des techniques d'implémentation. Le rôle des modèles PIM est de faire le lien entre le modèle d'exigences et le code de l'application. En effet, ils doivent être pérennes et productifs pour constituer le socle de tout le processus de génération de code défini par le MDA. La productivité des PIM signifie qu'ils doivent être suffisamment précis et riches en information pour qu'une génération automatique de code soit faisable. Indépendamment des plates-formes d'exécution, les diagrammes de classes UML permettent de décrire l'aspect statique d'une application alors

que les diagrammes de séquences ou d'activités permettent de modéliser l'aspect dynamique.

— Code : le modèle PSM (Platform Specific Model) ou modèle spécifique aux plates-formes d'implémentation, dit aussi modèle de code, permet de projeter les spécifications du PIM sur une plateforme cible afin de faciliter la génération de code. Les modèles de code sont essentiellement productifs mais ne sont pas forcément pérennes, en effet, il est possible de déployer un même modèle PIM sur plusieurs plates-formes cibles (PSMs). Le MDA propose l'utilisation de profils UML pour l'élaboration de PSM, par exemple : CORP (UML Profile for CORBA) et CCMP (UML Profile for CCM).

Le MDA établit des liens de traçabilité automatiquement grâce à l'exécution de transformations de modèles CIM vers PIM et PIM vers PSM et des transformations inverses : code vers PSM, PSM vers PIM et PIM vers CIM. Il est également possible d'effectuer des opérations de raffinements (mapping horizontal) de PIM à PIM ou de PSM à PSM. Le passage de PIM à PSM fait intervenir des mécanismes de composition et de transformation de modèles avec un modèle de description de plateforme (Platform Description Model - PDM) donnant lieu à un cycle de développement en Y (voir figure 2.2).

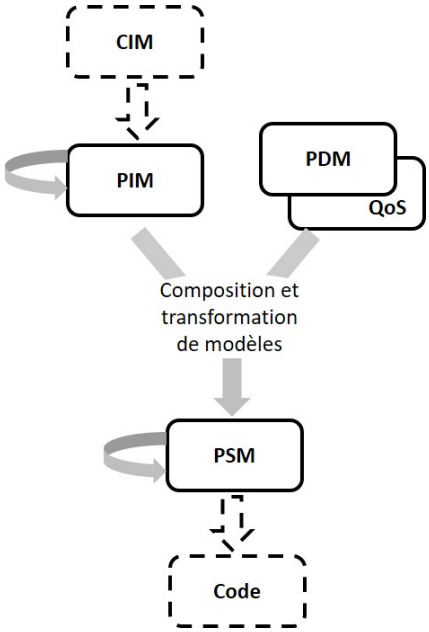



FIGURE 2.2 – *Le cycle de développement en Y du MDA.*

En résumé, le MDA permet aux entreprises de capitaliser sur leur métier grâce à des modèles pérennes du savoir-faire, de gagner en termes de productivité en automatisant une partie du processus de développement et enfin faciliter la migration d'une plateforme d'exécution à une autre.

## 2.4 Transformation de modèles

La transformation de modèles joue un rôle essentiel dans l'IDM afin de rendre les modèles opérationnels. D'après [Czarnecki et Helsen, 2006], la transformation permet la manipulation automatique des modèles par des outils à un niveau d'abstraction élevé à des fins de :

— Génération de modèles de bas niveau à partir de modèles de niveau supérieur ; la génération de code, de test et de documentation en font partie.
— Mapping et synchronisation entre des modèles de même niveau d'abstraction ou de niveaux différents.
— Création de vues basées sur des requêtes pour un système donné.
— Evolution de modèles, telle que la restructuration (refactoring).
— Rétroingénierie de modèles de haut niveau à partir de modèles de niveau inférieur ou de code.

Nous avons retenu les définitions suivantes fournies dans [Kleppe et al., 2003] et généralisées par [Mens et Van Gorp, 2006] :

**Définition** ***(Transformation de modèles)*** Une transformation de modèles est un processus qui génère un ou plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources suivant une définition de transformation.

**Définition** ***(Définition de transformation)*** Une définition de transformation est un ensemble de règles de transformation qui spécifient comment un modèle décrit dans un langage source, peut être transformé en un modèle décrit dans un langage cible.

**Définition** ***(Règle de transformation)*** Une règle de transformation est une description de la façon dont une ou plusieurs constructions en langage source peuvent être transformées en une ou plusieurs constructions en langage cible.

Comme le montre la figure 2.3, une transformation définie par un modèle de transformation Mt consiste donc à faire exécuter par un moteur de transformation un ensemble de règles assurant la production d'un ou de plusieurs modèles de sortie Mb à partir d'un ou de plusieurs modèles d'entrée Ma tout en garantissant la conformité avec les métamodèles MMt, MMb et MMa respectivement. De même que ces derniers doivent être conformes à leur méta-métamodèle MMM.

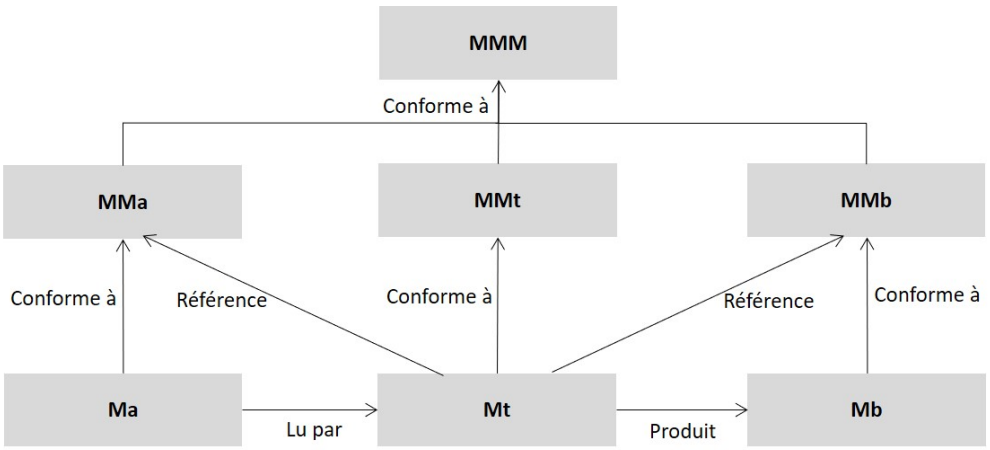



FIGURE 2.3 – *Principes de la transformation de modèles.*

### 2.4.1 Taxonomie de la transformation de modèles

Plusieurs classifications des types de transformation de modèles regroupant des outils, des techniques ou des formalismes ont été proposées dans la littérature [Czarnecki et Helsen, 2006], [Mens et Van Gorp, 2006].

En fonction du langage dans lequel sont exprimés les modèles source et cible, [Mens et Van Gorp, 2006] font la différence entre les transformations **endogènes** et **exogènes**. En effet, une transformation est dite endogène si les modèles source et cible sont conformes au même métamodèle et exogène si les modèles source et cible sont conformes à des métamodèles différents. En outre, le niveau d'abstraction auquel appartiennent les modèles source et cible permet de distinguer entre les transformations horizontale et verticale. Si les modèles source et cible appartiennent au même niveau d'abstraction, la transformation est dite **horizontale** ; exemples typiques : le refactoring et la migration logicielle. Si les modèles source et cible se situent à des niveaux d'abstraction différents, la transformation est qualifiée de **verticale** ; exemple : le raffinement, où une spécification est progressivement raffinée en une implémentation à part entière en y ajoutant des détails plus concrets. Ces différents types de transformation sont repris sur la figure 2.4.

Une dernière distinction peut être faite entre des transformations de modèles ne prenant en compte que l'aspect syntaxique et des transformations plus sophistiquées traitant la sémantique du modèle. Comme exemple de transformation syntaxique, un analyseur qui transforme la syntaxe concrète d'un modèle dans un certain langage de modélisation en une syntaxe abstraite sur laquelle des transformations sémantiques plus complexes (refactoring ou optimisation) peuvent être appliquées.

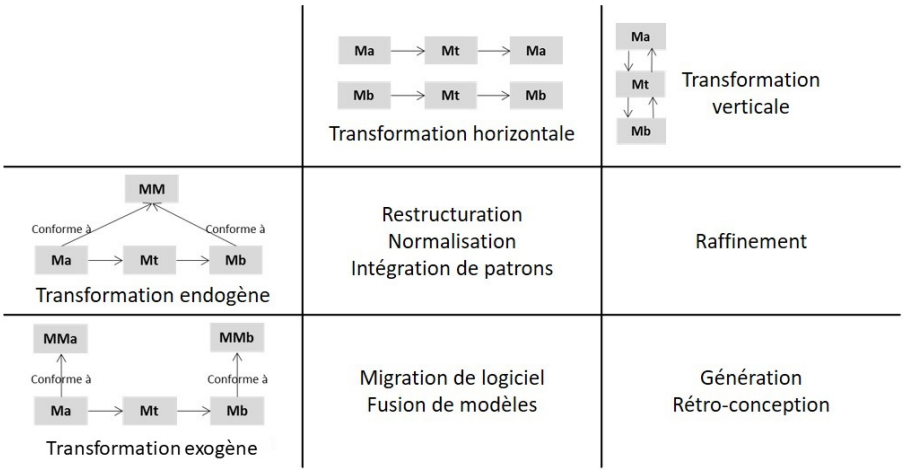



FIGURE 2.4 – *Types de transformation et leurs principales utilisations [Combemale, 2008].*

### 2.4.2 Approches de transformation de modèles

Dans [Czarnecki et Helsen, 2006] et selon la nature du modèle cible, nous distinguons entre les approches de transformation de modèle à texte (model to text M2T) et les approches modèle à modèle (model to model M2M). Les transformations M2T sont utilisées pour générer des artefacts textuels tels que le code et la documentation ; elles sont aussi considérées comme un cas particulier de transformations M2M.

#### 2.4.2.1 Approches modèle vers texte

Nous rapportons les approches suivantes :

— **Approche basée sur le visiteur :** elle consiste à parcourir la représentation interne d'un modèle donné en entrée à l'aide d'un visiteur et écrire du texte (code) sur un flux de sortie. Ce principe a été utilisé dans le projet Jamda (Java Model Driven Architecture) [1], un framework orienté objet qui prend en entrée un modèle UML de classes métiers, le transforme en ajoutant de nouvelles classes pour prendre en charge l'implémentation, puis génère du code.

— **Approche basée sur les templates :** Préconisée par l'OMG, elle permet de produire du code à partir de spécifications de haut niveau appelées templates. Un template est une représentation abstraite et généralisée de la sortie textuelle qu'il décrit. Il a une partie statique composée de fragments de texte qui apparaissent dans la sortie tels quels et une partie dynamique embarquée dans des espaces réservés de méta-code qui implémentent la logique de génération. Comme illustré dans la figure 2.5, l'exécution du template par un moteur permet de calculer la partie dynamique et remplacer donc les méta-codes par du texte statique en fonction du modèle d'entrée (run-time input) et ce conformément au métamodèle d'entrée (design-time input). Parmi les outils les plus utilisés actuellement, nous citons : Acceleo, Xpand, XSLT, Velocity, Jet, Fujaba, EGL et Simulink TLC [Syriani et al., 2018].

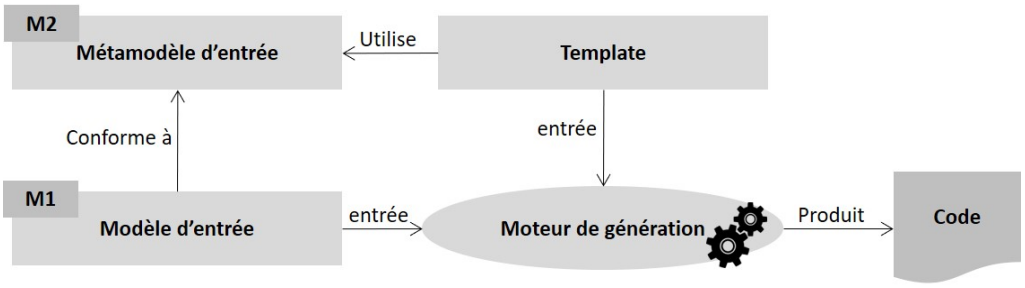



FIGURE 2.5 – *Processus de génération de code à partir de templates*

#### 2.4.2.2 Approches modèle vers modèle

Il est possible de distinguer entre les approches suivantes :

— **Approche par manipulation directe :** elle fournit une représentation interne du modèle et quelques APIs (Application Programming Interface) pour les manipuler. Souvent implémentée en tant que framework orienté objet, elle offre une infrastructure minimale pour organiser les transformations. Cependant, il revient aux utilisateurs d'implémenter des règles de transformation, ordonnancement et traçabilité dans un langage de programmation tel que Java.

— **Approche dirigée par la structure :** une transformation se déroule en deux phases ; la première concerne la création de la structure hiérarchique du modèle cible, alors que la seconde renseigne les attributs et les références dans la cible. Il incombe ainsi aux utilisateurs de fournir uniquement les règles de transformation. OptimalJ et QVT sont des exemples pratiques d'implémentation de cette approche.

— **Approche opérationnelle :** comparable à la manipulation directe mais offre un support plus spécifique à la transformation de modèles. Une solution typique

1. https ://jamda.sourceforge.net/

consiste à étendre le méta-métamodèle avec des moyens d'expression de calculs. Par exemple, la combinaison du MOF avec le langage d'expression de contraintes OCL mène à un environnement de programmation orienté objet à part entière. Cette approche a été mise en œuvre dans QVT Operational, XMF-Mosaic et Kermeta.

— **Approche relationnelle :** basée sur le concept clé de relations mathématiques, l'idée de base est de lier les éléments sources et cibles en spécifiant leurs relations à l'aide de contraintes. Les relations sont purement déclaratives et leur spécification n'est pas exécutable. QVT Relations, Kent MTL et Tefket sont des exemples d'implémentations de cette approche. Afin de rendre exécutables les spécifications, la programmation logique était un choix naturel pour apporter une sémantique exécutable avec l'utilisation des prédicats pour la description des relations.

— **Approche basée sur la transformation de graphes :** des règles de réécriture et de transformation de graphes sont appliquées sur un modèle en entrée sous forme de graphe orientée étiqueté pour le transformer en un modèle de sortie. Cette approche est supportée dans AGG (Attributed Graph Grammar), Atom3, BOTL (Bidirectional Object-oriented Transformation Language) et UMLX.

— **Approche hybride :** permet de combiner différentes techniques des approches précédentes à différents niveaux de granularité ; essentiellement au niveau des composants et des règles de production. QVT est un exemple d'utilisation d'une approche hybride par composants alors que ATL combine des approches déclaratives et impératives à un niveau plus fin (règle de transformation).

## 2.5 Les langages dédiés à la modélisation

Les langages de modélisation permettent une expression directe et formelle des concepts liés au monde réel grâce à des constructions de modélisation de base. La conceptualisation du domaine permet donc de décrire un modèle mental (abstraction du domaine) lié à l'état de certaines parties du monde réel. Le niveau de dépendance de la conceptualisation à un domaine spécifique permet de classer les langages en deux catégories : les langages de modélisation à usage général (GPML), applicables dans n'importe quel domaine et les langages de modélisation spécifiques au domaine (DSML [2]), qui sont conçus pour un domaine, un contexte ou une industrie spécifique [Zečević et al., 2017].

### 2.5.1 GPML (General Purpose Modeling Language)

Ce sont des langages conçus pour couvrir un large éventail de domaines à des fins de modélisation. Il s'agit de concepts de modélisation génériques qui n'incluent aucun aspect spécifique à un domaine particulier. Le meilleur exemple étant le langage UML. Bien que facilitant la communication entre les différentes communautés, l'aspiration générale de ces langages crée un écart conceptuel entre les domaines de problèmes et les domaines de solutions engendrant une complexité supplémentaire. En conséquence, des langages de modélisation spécifiques au domaine ont été créés pour aider les utilisateurs à spécifier la solution directement à l'aide de concepts de domaine du problème et pallier ainsi aux limites des langages généralistes [Hölldobler et al., 2017].

2. Dans la littérature, un DSML est appelé aussi DSL (Domain Specific Language) graphique.

### 2.5.2 DSML (Domain Specific Modeling Language)

Les DSMLs sont des langages de modélisation dédiés à un domaine particulier, ils offrent aux utilisateurs des concepts propres à leur métier afin de modéliser un aspect particulier (préoccupation) d'un système. Un DSML définit le vocabulaire de base du domaine métier et les relations entre ses différents concepts, il est petit de taille par le fait qu'il contient un nombre réduit de concepts familiers aux experts du domaine, facile à apprendre, à manipuler et à combiner. Ainsi, les connaissances d'un domaine métier peuvent être capitalisées par la définition de langages spécifiques et outillés permettant d'automatiser certaines tâches du processus de développement.

### 2.5.3 Définition d'un langage de modélisation

Un langage de modélisation $(L_m)$ est défini selon le tuple $(AS, CS^*, M^*_{ac}, SD^*, M^*_{as})$ où $AS$ est la syntaxe abstraite, $CS^*$ est la (les) syntaxe (s) concrète (s), $M^*_{ac}$ est l'ensemble des mappings de la syntaxe abstraite vers la (les) syntaxe (s) concrète (s), $SD^*$ est le (s) domaine (s) sémantique (s) et $M^*_{as}$ est l'ensemble des mappings de la syntaxe abstraite vers le (s) domaine (s) sémantique (s) [Jézéquel et al., 2012].

La syntaxe abstraite $AS$ occupe une place centrale dans la description d'un DSML, décrite en premier, elle exprime la structure du langage en termes de concepts et leurs relations. Une telle syntaxe peut être exprimée au travers d'un métamodèle dont les constructions élémentaires sont fournies par un langage de métamodélisation tel que le standard MOF, Ecore d'Eclipse, KM3, Xcore ou Kermeta. Les langages de métamodélisation s'inspirent du paradigme objet pour définir les concepts d'un DSML ; par convention, la représentation graphique du diagramme de classes UML est utilisée à cet effet. Les syntaxes concrètes $CS^*$ mettent à disposition de l'utilisateur un ou plusieurs formalismes (textuels et/ou graphiques) pour manipuler les concepts de la syntaxe abstraite et en créer des modèles par instanciation. Définir une syntaxe concrète revient à définir un des mappings de $M^*_{ac} : AS \leftrightarrow CS$, ainsi chaque construction définie dans AS est annotée par une ou plusieurs décorations de CS directement manipulables par l'utilisateur du langage et ce avec différents formalismes moyennant les mêmes constructions et la même représentation abstraite. De nombreux projets visent à définir des modèles de syntaxe concrète graphique tels que GMF (Generic Modeling Framework), Obeo Designer, Sirius et MetaEdit+, ou textuelle comme : EMFText et Xtext. Par ailleurs, la sémantique est exprimée à partir des constructions de la syntaxe abstraite par un lien vers un domaine sémantique définissant l'ensemble des états atteignables par le système $M^*_{as} : AS \leftrightarrow SD$, ce qui permet donc de définir de manière précise la signification des constructions du langage selon les besoins de vérification, simulation, compilation, etc.

### 2.5.4 Limites des DSMLs

De par leur grande expressivité, les DSMLs offrent des solutions plus simples et plus concises comparés aux GPMLs. En effet, l'utilisation de concepts et de notations adaptés à un domaine d'application particulier à un niveau d'abstraction élevé permet de réduire les couts de développement des systèmes et gagner en matière de productivité et de qualité du produit final, ainsi que l'inclusion directe des utilisateurs finaux dans le processus de développement logiciel.

Cependant, les DSMLs sont souvent perçus comme une arme à double tranchant ; bien qu'ils fournissent un support solide pour la communication au sein des communautés, permettant aux experts de s'exprimer en utilisant des concepts adaptés à leurs

besoins exacts, ils sont un mauvais support pour la communication entre les communautés en raison de leur manque de concepts communs et transcendants. Ainsi, la spécificité même du domaine qui rend les DSMLs si puissants pour exprimer des concepts particuliers à un domaine donné réduit également leur capacité à communiquer des informations (c'est-à-dire des connaissances) entre les domaines. En d'autres termes, les DSML ont tendance à avoir un effet tour de Babel ; ils créent des communautés distinctes de concepteurs avec un grand nombre de modèles difficiles à gérer et induisant des problèmes d'interopérabilité [Atkinson et al., 2012].

Une autre conséquence à leur aspect spécifique fait que le nombre de DSMLs soit en constante augmentation. Une augmentation favorisée d'un côté par l'élargissement incessant de la gamme des domaines traités par les systèmes logiciels et d'autre part par la taille et la complexité de ces derniers [Vallejo, 2015]. Ainsi, des langages partageant des caractéristiques communes (concepts, structure, outils) forment une famille de langages.

Outre le fait d'accentuer le phénomène tour de Babel, la conception d'un nouveau DSML n'est pas une tâche triviale ; elle demande généralement un investissement initial augmentant par conséquent les couts de développement et de maintenance du logiciel comparée à l'utilisation d'un GPML ou d'un DSML existant. De plus, sans outils et générateurs appropriés, les capacités du langage deviennent limitées, ce qui nécessite l'implémentation et la maintenance d'outils supplémentaires qui existent déjà dans le contexte des GPML [Zečević et al., 2017].

Une approche alternative à la création de nouveaux DSML ex-nihilo consiste à étendre ou à personnaliser des langages de modélisation existants. Les modèles ainsi créés offrent l'avantage d'utiliser des outils, des technologies et des expériences déjà éprouvés par les concepteurs dans le cadre d'une démarche IDM. Des mécanismes de spécification d'extensions ont déjà été intégrés dans certains langages de modélisation où un langage embarqué utilise la syntaxe, les bibliothèques existantes, ainsi que les outils associés d'un langage hôte. Dans de nombreux travaux de recherche, le langage généraliste UML a fait l'objet de customisation afin de répondre aux exigences d'un domaine d'application particulier. La création d'un DSML à partir de UML peut se faire soit en modifiant le métamodèle UML (heavyweight extension), soit en faisant appel au mécanisme d'extension du langage UML en l'occurrence le profil ; ce dernier permet d'ajouter des contraintes et de nouveaux éléments sans modifier le métamodèle de base, ainsi l'extension est dite légère (lightweight extension) [Kraas, 2019].

### 2.5.5 Profils UML

La possibilité de personnaliser UML pour un domaine spécifique est l'une des grandes fonctionnalités d'UML. La création de personnalisations permet de tirer parti des outils et conventions de modélisation existants définis par la spécification UML tout en rendant la modélisation plus facile pour l'utilisateur final (et éventuellement moins abstraite). Le type de personnalisations requises dépend souvent de la nature du domaine et de la manière dont le modèle étendu est destiné à être utilisé. Comme le souligne la spécification de UML [Obj, 2017], il y a plusieurs raisons pour lesquelles on peut vouloir personnaliser un méta-modèle :

— Donner une terminologie adaptée à une plate-forme ou à un domaine particulier.
— Donner une syntaxe pour les constructions qui n'ont pas de notation (comme dans le cas des actions).
— Donner une notation différente pour les symboles déjà existants (comme pouvoir utiliser une image d'un ordinateur au lieu du symbole de nœud ordinaire pour représenter un ordinateur dans un réseau).

— Ajouter une sémantique supplémentaire à UML ou à des métaclasses spécifiques.
— Ajouter des types qui n'existent pas dans UML (comme la définition d'un temporisateur, d'une horloge ou d'un temps continu).
— Ajouter des contraintes qui restreignent la façon dont les constructions d'UML sont utilisées (comme interdire l'héritage multiple).
— Ajouter des informations pouvant être utilisées lors de la transformation de modèles.

Une caractéristique clé de ces extensions est qu'elles doivent être compatibles, tant sur le plan syntaxique que sémantique, avec le concept UML de base qu'elles étendent. Cela assure la compatibilité d'un profil avec la norme UML et les outils qui la supportent, ce qui est un avantage majeur du mécanisme de profil. Cela signifie que les outils UML standard peuvent être réutilisés et que les concepteurs familiers avec l'UML standard peuvent appliquer leurs connaissances existantes, n'ayant qu'à apprendre les ajouts spécifiques du profil, réduisant ainsi les coûts de formation et d'outillage.

L'extensibilité de première classe est gérée via le MOF, où il n'y a aucune restriction au niveau du métamodèle : il est possible d'ajouter des sous-classes et des associations si nécessaire. Si l'on veut faire des personnalisations simples, en ajoutant de nouvelles propriétés aux méta-classes UML existantes, alors une extension légère est la voie à suivre. Même s'il n'y a pas de réponse simple pour savoir quand créer un nouveau métamodèle et quand créer un nouveau profil, un choix néanmoins peut être fait en analysant l'espace de domaine du DSML afin de constater s'il y a beaucoup de chevauchements entre les concepts d'UML et ceux du DSML ; si tel était le cas alors il faudrait privilégier une extension légère d'UML par la création de profils, comme le montre la figure 2.6.

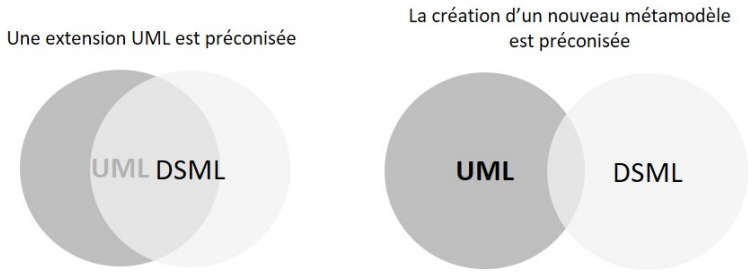



FIGURE 2.6 – *Extension d'UML vs. Création de métamodèle*

#### 2.5.5.1 Définitions de quelques concepts liés aux profils UML

Dans ce qui suit, nous présentons des définitions émises par l'OMG [Obj, 2017] concernant quelques concepts clés associés aux profils UML.

**Définition *(Profil)*** Un profil est une forme restreinte du métamodèle de référence qui peut être utilisé pour étendre UML avec des constructions spécifiques à un domaine, une plate-forme ou une méthode particulière . Dans UML, le diagramme de profil est un diagramme de structure qui décrit un mécanisme d'extension léger en définissant des stéréotypes, des valeurs étiquetées et des contraintes. Un profil utilise la même notation qu'un package, avec en plus le mot-clé « Profile » affiché avant ou au-dessus du nom du package. Un ou plusieurs profils peuvent être appliqués à un modèle (package) ; l'application d'un profil signifie qu'il est possible d'appliquer les stéréotypes, les valeurs étiquetées et les contraintes définis dans le profil à des éléments de modèle spécifiques, tels que les classes, les attributs, les opérations et les activités. Plusieurs profils complémentaires peuvent être appliqués au même package afin de

répondre à différentes préoccupations ; une telle opération s'appelle intégration de profils. Un profil peut également réutiliser tout ou une partie d'un autre et d'étendre d'autres profils. Ainsi, des stéréotypes peuvent être directement réutilisés en étant référencés ou spécialisés dans d'autres profils. Toutefois, les stéréotypes abstraits ne peuvent pas être spécialisés.

**Définition** ***(stéréotype)*** Un stéréotype définit une extension pour une ou plusieurs métaclasses et permet l'utilisation d'une terminologie ou d'une notation spécifique à la place ou en plus de celles utilisées pour les métaclasses étendues. Un stéréotype est un type limité de métaclasse qui ne peut pas être utilisé seul, mais doit toujours être utilisé en conjonction avec l'une des métaclasses qu'il étend. Chaque stéréotype peut étendre une ou plusieurs métaclasses par association (extension) plutôt que par généralisation/spécialisation. De même, une métaclasse peut être étendue par un ou plusieurs stéréotypes. Un Stéréotype utilise la même notation qu'une classe en ajoutant le mot-clé « Stereotype » avant ou au-dessus du nom de la classe. La différence entre un stéréotype et une construction définie par l'utilisateur à partir des concepts du langage réside dans le fait qu'un stéréotype est considéré comme un concept de *première classe*. Ce terme signifie qu'il s'agit d'un concept intégré au langage et qui fait donc partie de la définition du langage, plutôt que d'être limité à un modèle utilisateur spécifique. Tout comme une classe, un stéréotype peut avoir des propriétés, qui sont traditionnellement appelées définitions d'étiquettes (tag definitions). Lorsqu'un stéréotype est appliqué à un élément du modèle, les valeurs des propriétés sont qualifiées de valeurs étiquetées (tagged values). Il est possible d'attacher une image à un stéréotype qui peut être utilisée à la place ou en plus de la notation normale d'un élément du modèle auquel le stéréotype est appliqué.

**Définition** ***(Extension)*** Une extension est un type particulier d'association liant un stéréotype à une métaclasse. Elle indique que les propriétés d'une métaclasse sont étendues via un stéréotype et donne la possibilité d'ajouter/supprimer de manière flexible des stéréotypes aux classes. Une extension est notée par une flèche pointant d'un stéréotype vers la métaclasse étendue, où la pointe de la flèche est représentée par un triangle plein. Il existe une distinction essentielle entre l'extension UML et la généralisation UML. Dans une généralisation, les relations sont conjonctives, ce qui implique qu'une instance d'une sous-classe avec plusieurs ancêtres hérite simultanément des caractéristiques de chacun de ses ancêtres. En revanche, dans le cadre d'une extension, l'interprétation est disjonctive : une instance de stéréotype n'étend qu'une seule instance de sa classe de base. Ainsi, chaque classe de base d'un stéréotype représente un cas d'utilisation distinct du concept du domaine.

**Définition** ***(Contrainte)*** Une contrainte est une assertion indiquant une restriction qui doit être satisfaite par toute réalisation valide du modèle contenant la contrainte. Une contrainte est attachée à un ensemble d'éléments (du modèle) contraints et représente des informations sémantiques supplémentaires sur ces éléments. Elle est notée entre accolades et peut être insérée au besoin dans une note graphique. La spécification d'une contrainte peut être exprimée dans un langage formel (tel que OCL pour Object Constraint Language), un langage de programmation (tel que Java) ou tout simplement dans un langage naturel.

La syntaxe abstraite représentée dans la figure 2.7 capture les concepts précédemment décrits liés aux profils UML.

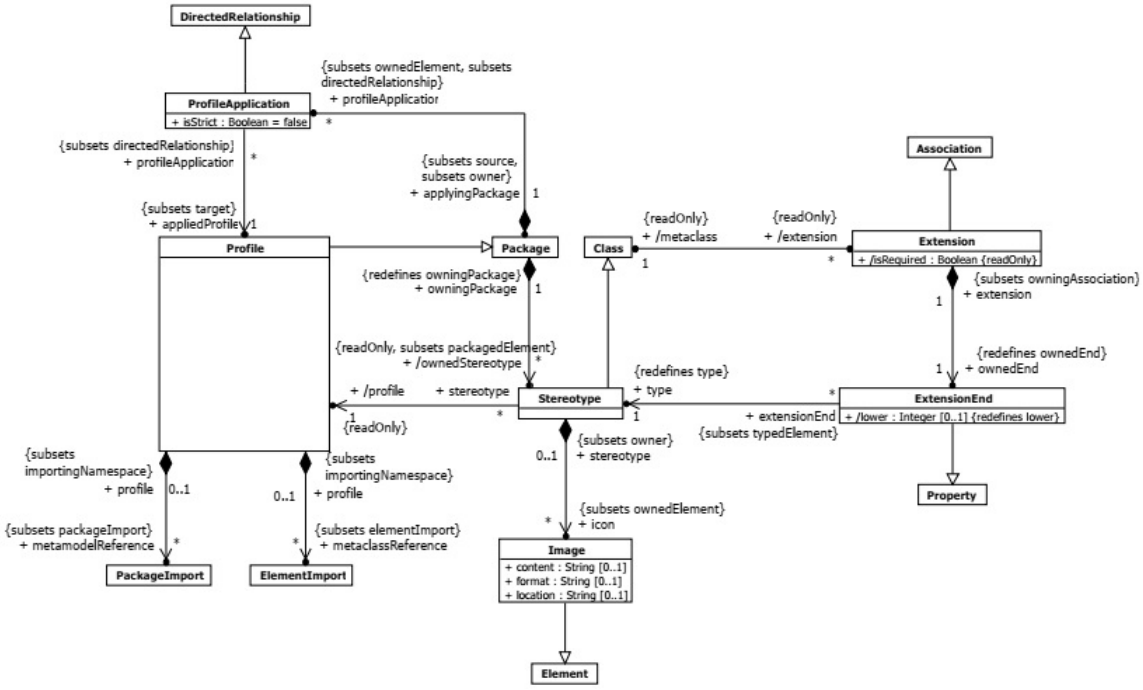



FIGURE 2.7 – *La syntaxe abstraite d'un diagramme de profil UML [Obj, 2017]*

### 2.5.5.2 Types de profils UML

Un profil UML permet d'enrichir les modèles UML en ajoutant des informations complémentaires non représentables directement avec le UML standard. Il peut être employé de deux façons : soit pour étendre les capacités du langage UML, soit pour associer des données supplémentaires aux modèles, utiles pour des tâches comme l'analyse de modèles ou la génération de code. Par conséquent, deux types de profils sont distingués : les profils de langage et les profils d'annotation.

**Profils de langage** Dans ce cas, le mécanisme de profil sert à créer soit un langage spécifique à un domaine, complet et autonome, tel que SysML [Obj, 2023a], soit un langage utilitaire, comme MARTE [Obj, 2023b], conçu pour étendre UML ou un autre profil de langage en y ajoutant de nouveaux concepts propres à un domaine particulier.

**Profils d'annotation** Les profils d'annotation, quant à eux, servent à ajouter des informations supplémentaires à un modèle et à ses éléments, particulièrement pour des concepts difficiles à exprimer avec le UML standard. Ils sont principalement destinés à l'analyse des modèles. Les deux sous-profils d'analyse de MARTE, à savoir le Schedulability Analysis Modeling (SAM) et le Performance Analysis Modeling (PAM), illustrent bien cet usage. Pour utiliser un profil d'annotation, il faut d'abord l'appliquer au modèle, ce qui rend ses concepts disponibles. Les annotations ainsi ajoutées fonctionnent comme une superposition transparente sur une diapositive projetée, créant une vue combinée du modèle initial et des informations supplémentaires. Il est également possible d'appliquer plusieurs profils en même temps, à condition qu'il n'existe pas de conflits entre eux. Une particularité intéressante des profils d'annotation réside dans leur capacité à être désappliqués d'un modèle. Cela élimine de manière efficace les annotations supplémentaires liées à ce profil, ce qui laisse le modèle sous-jacent dans son état initial, sans indications supplémentaires. Grâce à cette souplesse, il est possible

d'appliquer successivement plusieurs annotations au même modèle, ce qui facilite la réalisation de diverses analyses.

#### 2.5.5.3 Exemples de profils UML

L'OMG propose un certain nombre de profils couvrant chacun un domaine métier particulier et formant l'écosystème UML, illustré par la figure 2.8. Ainsi des profils peuvent être combinés en associant différents artefacts permettant de combler par conséquent le fossé de communication entre les différentes disciplines de l'ingénierie.


FIGURE 2.8 – *L'écosystème UML*

**MARTE (UML Profile for MARTE)** Le profil UML MARTE (Modeling and Analysis of Real-Time and Embedded systems) est destiné à supporter une approche d'IDM pour le développement des systèmes temps réel embarqués. Il permet de prendre en charge les étapes de spécification, de conception et de vérification/validation en offrant aux concepteurs des constructions de langage de première classe pour leur domaine d'application. MARTE fut appelé aussi à remplacer le profil SPTP (UML Profile for Schedulability, Performance and Time) dont le domaine d'application était restreint à l'analyse quantitative de modèles UML. Il s'agissait notamment de couvrir des besoins particuliers en termes de modélisation des plateformes logicielles et matérielles et aussi d'analyses de performance et d'ordonnançabilité.

**BPMNProfile (UML Profile for BPMN Processes)** Le profil UML pour les processus BPMN (Business Process Model and Notation) est assimilé à un mapping entre les concepts des spécifications BPMN et UML pour lesquelles la notation graphique BPMN dédiée à la modélisation des processus métiers est utilisée comme syntaxe concrète pour des modèles d'activités et de collaborations UML. En effet, les deux langages modélisent de manière semblable la transmission de l'information entre les étapes d'une procédure via les processus BPMN et les activités UML ainsi que les interactions entre procédures ou entités exécutant des procédures (collaborations). Le métamodèle UML est étendu de façon à garantir une équivalence sémantique avec BPMN, par conséquent les modèles sont exécutés de la même manière qu'ils soient capturés à l'aide du métamodèle BPMN ou du métamodèle UML étendu [Bocka et al., 2014].

**QFTP (UML Profile for Modeling QoS and FT)** Le profil UML QFTP (UML Profile for Modeling Quality of Service and Fault Tolerance) définit un ensemble d'extensions UML pour représenter les concepts de qualité de service et de tolérance aux pannes

des systèmes afin de combler les limites d'UML dans ce domaine. Les extensions sont intégrées dans deux frameworks de base : QoS Modeling Framework et FT Modeling Framework. Le premier permet d'associer des exigences et des propriétés aux éléments du modèle pour introduire des aspects extra-fonctionnels dans les modèles UML. Le second comprend des notations pour modéliser les évaluations des risques, en accordant une attention particulière à la description des dangers, des risques et des traitements des risques.

**UTP2 (UML Testing Profile 2)** Le profil UML UTP2 permet de concevoir, visualiser, spécifier, analyser, construire et documenter les artefacts couramment utilisés et requis pour diverses approches de test, en particulier les approches de test basé sur un modèle (MBT : Model Based Testing). UTP fait partie de l'écosystème UML, et en tant que tel, il peut être combiné avec d'autres profils de cet écosystème afin d'associer des artefacts liés aux tests avec d'autres artefacts système pertinents tels que les exigences, les risques, les cas d'utilisation, les processus métiers, les spécifications des systèmes, etc. Cela permet aux ingénieurs des exigences, aux ingénieurs système et aux ingénieurs de test de combler le fossé de communication entre les différentes disciplines d'ingénierie.

**NIEM-UML (UML Profile for NIEM)** Le profil UML pour NIEM (National Information Exchange Model) fournit un moyen de créer des échanges d'informations conformes à NIEM en UML plutôt que de coder directement des définitions de schéma XML (XSD : XML Schema Definition) en permettant aux modélisateurs et aux développeurs d'appliquer le profil avec un minimum d'effort afin de créer de nouveaux modèles ou de modifier des modèles existants et, finalement, de produire des artefacts NIEM MPD (Model Package Description). NIEM-UML adopte la norme MDA pour faciliter la séparation des préoccupations entre les besoins de l'entreprise et les technologies d'implémentation.

**TelcoML (UML Profile for Telecommunication Services)** Le profil UML TelcoML (UML Profile for Advanced and Integrated Telecommunication Services) est défini comme une spécialisation du profil UML SoaML (Service oriented architecture Modeling Language) pour la modélisation des services de communication temps réel et la prise en charge de nombreux autres services et normes d'architecture existantes. L'extension est nommé bibliothèque de modélisation des télécommunications (TelcoML : Telecommunication Modeling Library). TelcoML permet l'application de techniques dirigées par les modèles pour un développement agile des services de télécommunications avancés et intégrés. Ces derniers exploitent la convergence des réseaux de communication fixe, sans fil et voix, et en même temps tire parti de la pléthore d'installations accessibles depuis le World Wide Web. A cela s'ajoute la prise en charge de la sensibilité au contexte de l'utilisateur comme la présence, la localisation, les préférences de l'utilisateur et l'utilisation des moyens de communication tels que les SMS ou la messagerie vocale.

**UPR (UML Profile for ROSETTA)** Le profil UML de ROSETTA, en référence à son fondement mathématique sous-jacent, le framework Relational Oriented Systems Engineering Technology Tradeoff and Analysis, fournit une solution orientée objet pour une structuration précise et efficace des informations à l'appui d'une analyse basée modèle pour l'optimisation de l'architecture et la conception des systèmes. L'orientation relationnelle du framework prend en charge la transformation des modèles depuis la description de l'architecture jusqu'à la conception détaillée du système.

**SoaML (Service Oriented Architecture Modeling Language)** Le langage SoaML fournit un métamodèle et un profil UML pour la spécification et la conception de services au sein d'une architecture orientée services (SOA : Service Oriented Architecture). Il étend UML pour prendre en charge entre autres les capacités d'identification/spécification des services, leurs fonctionnalités, des exigences et leurs dépendances, définition des consommateurs et des fournisseurs de services, leurs connexions, leurs échanges, les protocoles/règles d'utilisation des services, etc.

**SysML (System Modeling Language)** SysML est un langage de modélisation à usage général pour l'ingénierie des systèmes (IS). Il réutilise un sous-ensemble d'UML 2.5 et fournit des extensions supplémentaires pour répondre aux exigences d'UML pour l'IS. Il est également possible de personnaliser SysML pour modéliser des applications spécifiques à un domaine, telles que les systèmes automobiles, aérospatiaux, de communication et d'information. SysML est conçu pour fournir des constructions simples mais puissantes pour la modélisation d'un large éventail de problèmes d'IS, il est particulièrement efficace pour spécifier les exigences, la structure, le comportement, les allocations et les contraintes sur les propriétés du système pour prendre en charge l'analyse technique.

## 2.6 La DPR comme préoccupation transverse

Le principe de séparation des préoccupations, tel que énoncé précocement par Dijkstra [Dijkstra, 1976], consiste à identifier les différentes préoccupations dans un système et à les séparer en les encapsulant dans des modules ou des parties appropriées afin de réduire la complexité globale. Même si nous disposons déjà d'un paradigme de décomposition, tel que l'orienté objet, certaines catégories de préoccupations ne peuvent pas être encapsulées dans des modules, ce qui leur vaut le nom de préoccupations transverses. Par exemple, certains types de fonctionnalités, comme l'authentification, sont par nature difficiles à encapsuler. Il en va de même pour un certain nombre d'exigences non fonctionnelles, qui sont intrinsèquement transverses, telles que la sécurité, la disponibilité, la répartition, la gestion des ressources ou les contraintes temps-réel. Dans ce contexte de séparation des préoccupations transverses, la programmation orientée aspect (Aspect Oriented Programming - AOP) a été utilisée pour gérer les comportements dynamiques et les reconfigurations sans altérer le code métier de base [Morin et al., 2009], [Kebir, 2012] et [Cardoso, 2012]. L'ingénierie dirigée par les modèles vise, de façon encore plus approfondie que l'approche AOP, à offrir une variété de modèles qui expriment distinctement les préoccupations des utilisateurs, des concepteurs et des architectes [Jézéquel et al., 2012]. Dans cette optique, la DPR peut être efficacement gérée comme une préoccupation transverse.

### 2.6.1 Convergence de domaines

A un niveau d'abstraction élevé, les distinctions entre l'adaptation dynamique des logiciels et la DPR s'estompent, révélant une convergence de principes fondamentaux et de méthodologies qui les rendent presque indissociables dans leur approche et leurs objectifs. L'utilisation de l'IDM met en avant l'idée que les différences spécifiques entre les deux domaines deviennent moins perceptibles à mesure que l'on s'élève dans l'abstraction, laissant place à une compréhension unifiée et cohérente des concepts sous-jacents décrits par des modèles.

**Reconfiguration Vs Adaptation** La DPR et l'adaptation dynamique des logiciels partagent l'objectif d'ajuster les ressources d'un système dynamiquement pour répondre à des exigences changeantes, que ce soit pour maximiser les performances, réduire la consommation d'énergie, ou s'adapter à des conditions d'exploitation variables. La DPR permet de modifier la configuration d'un circuit afin de réallouer les ressources matérielles ou de changer la fonction du matériel en fonction des besoins actuels du système. De manière similaire, l'adaptation logicielle ajuste les paramètres ou le comportement des logiciels pour optimiser leur fonctionnement en réponse à des variations dans l'environnement ou la charge de travail.

**IP vs Composant logiciel** Une définition formulée par [Szyperski, 2002] et largement référencée dans la littérature considère un composant logiciel comme étant une unité *binaire* de composition dotée d'une interface spécifiée contractuellement et de dépendances de contexte explicites. Il peut être déployé indépendamment et est sujet à la composition par des tiers. Un composant définit donc un contrat formel pour les services qu'il offre à ses clients ainsi que ceux qu'il demande à d'autres composants dans le système, en fonction de ses interfaces fournies et requises. Ainsi, l'analogie est facilement établie avec le concept de hw/sw IP (composant d'architecture ou tâche applicative) et le bitstream associé. En effet, vu que les composants peuvent être développés puis assemblés par différentes compagnies tierces, il devient nécessaire de fournir les composants comme des unités binaires évitant ainsi la publication du code source.

La réutilisation de composants en boîte blanche et en boîte grise compromet souvent leur substituabilité. De même, les IP dans un SoC reconfigurable peuvent être des boîtes noires, grises, ou blanches selon le niveau d'accès et de transparence offert par le fournisseur de l'IP.

**Contrôleur vs Conteneur** Le contrôle d'un système reconfigurable repose généralement sur trois tâches clés : la collecte des informations susceptibles de déclencher la reconfiguration, la prise de décision concernant la reconfiguration et l'initiation du processus de reconfiguration. Le conteneur dans une approche à composants orientés services (voir la section 2.6.3) se charge de traiter les interactions avec le registre de services et donc de masquer la gestion du dynamisme de l'application ; il prend en charge également le cycle de vie du composant, ses liaisons et ses dépendances tout en gérant les propriétés non fonctionnelles associées au composant. Dans les deux domaines, le contrôleur et le conteneur gèrent les préoccupations transversales de manière similaire, bien que le conteneur ait un champ d'action plus étendu.

### 2.6.2 Adaptation dynamique des logiciels

L'essentiel de ce que nous désignons aujourd'hui sous le terme d'ingénierie logicielle a émergé en réaction à la croissance de la complexité dans le domaine de la programmation. Les architectures logicielles sont maintenant considérées comme l'une des solutions les plus prometteuses pour réduire cette complexité [Taylor et van der Hoek, 2007]. Dans [ISO, 2022], l'architecture est définit comme étant l'ensemble des concepts fondamentaux ou des propriétés d'un système dans son environnement, incarnés dans ses éléments, ses relations, et les principes de leur conception et de leur évolution. Ainsi, cette définition met en avant l'importance des éléments constitutifs d'un système, de leurs interrelations et des principes directeurs qui influencent leur conception et leur évolution, tout en prenant en compte le contexte environnemental. Les architectures logicielles dynamiques introduisent un aspect de dynamisme dans les

architectures logicielles. Elles sont caractérisées par la possibilité d'ajouter ou supprimer un composant et/ou de modifier les connexions entre les composants au cours de l'exécution du système sans nécessiter l'arrêt ou la réinitialisation du système complet. Ce phénomène est appelé évolution ou adaptation à l'exécution [Taylor et al., 2009]. Plusieurs termes dans la littérature ont été employés pour désigner ce que l'on appelle aujourd'hui l'adaptation dynamique, à savoir : modification de programme à la volée [Fabry, 1976], mise à jour dynamique de programme [Segal et Frieder, 1988], gestion du changement dynamique [Kramer et Magee, 1990], changement de version en ligne [Gupta et al., 1996], évolution à l'exécution [Oreizy et al., 1998], évolution dynamique [Malabarba et al., 2000], évolution en ligne [Wang et al., 2006] ou encore disponibilité dynamique ; cette dernière désigne une situation où les services peuvent devenir disponibles ou indisponibles à tout moment pendant l'exécution d'une application, et cela est en dehors du contrôle de l'application [Cervantes et Hall, 2004]. Bien que de nombreux problèmes d'adaptation aient été largement étudiés et abordés par divers domaines de l'ingénierie logicielle, il est désormais reconnu que l'adaptation logicielle doit être considérée comme une discipline autonome [Canal et al., 2006].

De [Buckley et al., 2005], [Andersson et al., 2009] et [Salehie et Tahvildari, 2009] à [Alhozaimy et al., 2024], plusieurs taxonomies de l'adaptation logicielle ont été proposées. Ces dernières prennent en compte généralement quatre aspects de la reconfiguration du système : pourquoi reconfigurer, quand reconfigurer, quoi reconfigurer, et comment reconfigurer . Comme le montre la figure 2.9, le besoin de reconfiguration est généralement motivé par trois facteurs principaux [Siddiqi et de Weck, 2008] : *la multi-capacité*, le système réalise plusieurs fonctions distinctes à différents moments ; *l'évolutivité*, le système change facilement au fil du temps en supprimant, en remplaçant ou en ajoutant de nouveaux éléments et fonctions et enfin *la résilience* ou *la survivabilité*, le système reste fonctionnel, éventuellement dans un état dégradé, malgré quelques défaillances. Une reconfiguration proactive est un processus dans lequel un système ajuste ou modifie sa configuration de manière anticipée, avant que des problèmes ou des besoins ne se manifestent. Ce type de reconfiguration est basé sur des prédictions, des analyses de tendances, ou des règles préétablies, et vise à prévenir les défaillances, optimiser les performances, ou améliorer l'efficacité du système. En surveillant en continu des paramètres tels que l'état du système, les conditions environnementales, ou les besoins en ressources, la reconfiguration proactive permet d'apporter des ajustements avant que des perturbations ou des dégradations de service ne se produisent. Lorsque des spécifications d'architecture sont utilisées pour déterminer quand une reconfiguration doit avoir lieu et ce qui doit changer, la reconfiguration est dite programmée. Lorsque les changements sont générés automatiquement par le système en cours d'exécution et que celui-ci décide du moment de leur application, la reconfiguration est considérée comme non programmée. Une reconfiguration réactive est un processus dans lequel un système adapte sa configuration en réponse à des événements ou des perturbations qui surviennent de manière imprévue ou externe. Contrairement à une reconfiguration proactive, qui anticipe et prépare les changements à l'avance, la reconfiguration réactive se déclenche lorsqu'un problème, une défaillance, ou un changement dans l'environnement est détecté. L'objectif est de réagir rapidement pour maintenir la continuité du service, corriger les erreurs, ou adapter le système aux nouvelles conditions. Les reconfigurations réactive et proactive peuvent être combinées dans un schéma de reconfiguration hybride.

Ces systèmes, potentiellement auto-adaptatifs [3], sont capables de prendre des déci-

3. Dans la littérature, on considère que l'auto-configuration est une propriété des systèmes auto-adaptatifs tout comme l'auto-optimisation, l'auto-maintenance, l'auto-réparation et l'auto-protection. L'auto-configuration qui nous intéresse particulièrement est la capacité de se reconfigurer automatique-

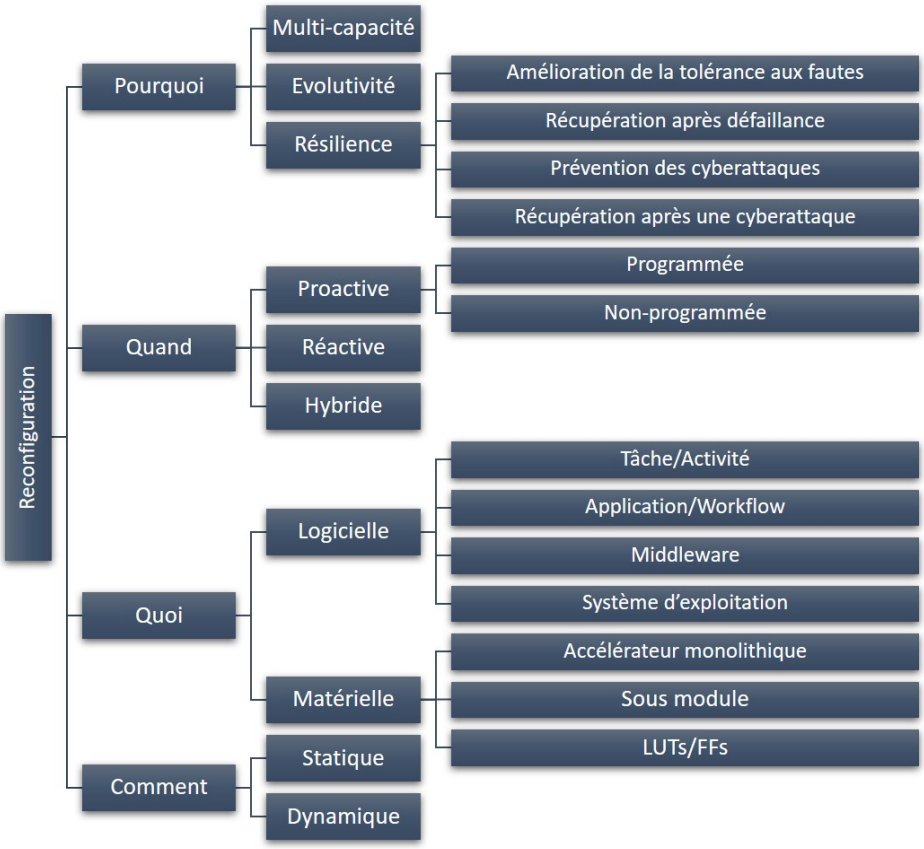



FIGURE 2.9 – *Taxonomie de la reconfiguration présentée dans [Alhozaimy et al., 2024] et enrichie.*

sions en temps réel pour déterminer le meilleur mode de fonctionnement et reposent généralement sur une boucle de contrôle bien connue, appelée MAPE-K (Monitor, Analyze, Plan and Execute based on Knowledge) [Kephart et Chess, 2003]. La boucle de rétroaction MAPE-K est le modèle de contrôle de référence le plus influent pour les systèmes autonomes et auto-adaptatifs [Arcaini et al., 2015]. La figure 2.10 illustre le processus de la boucle d'adaptation. L'étape de surveillance (Monitor) collecte des données des ressources gérées (utilisation du CPU, mémoire, température, charge réseau, etc.) à l'aide de capteurs et d'agents logiciels et détecte les conditions pouvant déclencher des adaptations. Ces données collectées sont ensuite analysées (étape Analyse) pour déterminer si un changement est nécessaire afin d'atteindre les objectifs du système. Des techniques d'analyse du seuil et d'identification des tendances ou des modèles prédictifs de machine learning peuvent être appliqués pour détecter des anomalies ou prévoir des comportements futurs. Si une adaptation est requise, l'étape de planification (Plan) définit une procédure visant à atteindre une nouvelle condition cible répondant aux objectifs, en incluant les étapes intermédiaires nécessaires à la transition d'un état à un autre. Enfin, la procédure planifiée est exécutée (étape Execute) sur les ressources gérées en activant les actions via des effecteurs, des API ou des commandes spécifiques au système. La base de connaissances (Knowledge), qui relie les différentes étapes de la boucle MAPE-K, peut être alimentée par diverses sources collectant l'information de manière automatique ou même manuelle. Elle peut inclure des historiques de fonctionnement, d'observations du comportement du système, de données collectées et stockées (comme les logs de capteurs), ainsi que des informations supplémentaires fournies par des ex-

ment et dynamiquement en réponse à des changements. Cela peut inclure l'installation, l'intégration, la suppression ainsi que la composition/décomposition des éléments du système.

perts humains pour définir les paramètres et les stratégies d'adaptation. Cependant, lorsque les systèmes sont vastes, complexes et hétérogènes, une seule boucle MAPE-K (centralisée) risque de ne pas suffire à gérer toutes les décisions d'adaptation. Dans ce cas, plusieurs boucles MAPE-K (décentralisées) peuvent être nécessaires. Dans [Porter et al., 2021], le projet de recherche (RASS : Resilient Autonomic Software Systems) a abordé la conception et le développement d'un framework pour systèmes logiciels distribués, à la fois hautement résilient et fiable, reposant sur une version décentralisée du modèle MAPE-K. Les auteurs dans [Andersson et al., 2023] ont introduit Decor, un framework de raisonnement basé sur des modèles pour la conception et l'évaluation du contrôle décentralisé MAPE-K destiné au systèmes auto-adaptatifs.

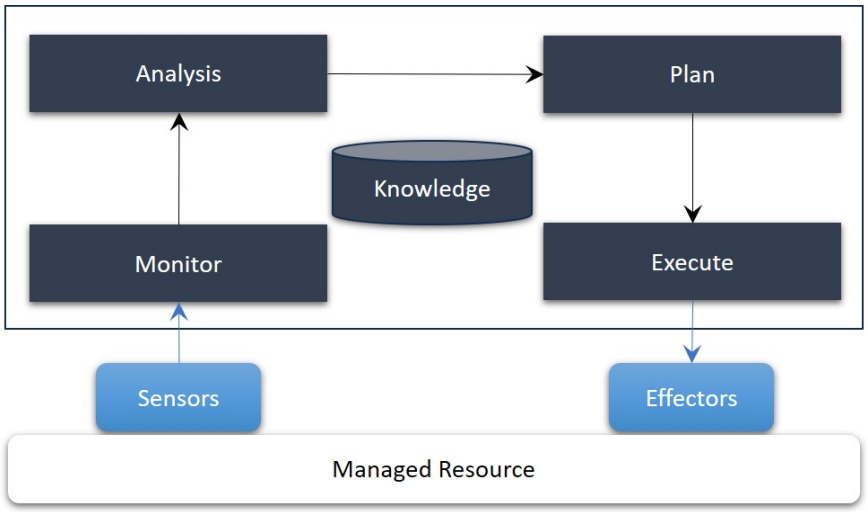



FIGURE 2.10 – *La boucle de contrôle MAPE-K.*

### 2.6.3 Paradigmes de conception

En génie logiciel, deux approches de conception des systèmes logiciels dynamiquement adaptatifs sont couramment utilisées : l'ingénierie logicielle basée composants (Component-Based Software Engineering CBSE) et l'ingénierie logicielle orientée services (Service-Oriented Software Engineering SOSE [4]).

CBSE et SOSE utilisent des approches et des technologies similaires, et partagent une architecture logicielle commune. Cependant, ils évoluent dans des directions différentes. Leur principale différence réside dans leur focalisation sur des acteurs distincts : la CBSE se concentre sur le point de vue du fournisseur, facilitant le déploiement de nouvelles fonctionnalités, tandis que la SOSE se concentre sur le point de vue du consommateur, fournissant des fonctions aux utilisateurs indépendamment des implémentations de service [Rouvoy et al., 2008].

#### 2.6.3.1 Ingénierie logicielle basée composants

En tant que paradigme, la programmation orientée objet a marqué une évolution significative par rapport à la programmation procédurale, en offrant un niveau d'abstraction et une modularité plus élevés, notamment grâce aux concepts d'héritage et de polymorphisme. Cela dit, l'approche orientée objet manquait d'abstraction et de mécanismes pour la composition des objets. Il était difficile de visualiser les relations entre

4. Appelée également Service Oriented Computing (SOC).

les objets dans une application lorsque leur nombre augmentait. En somme, la programmation orientée objet restait trop proche du code pour faciliter le développement d'applications exploitant pleinement des concepts tels que la réutilisation et la substitution d'objets [Meyer, 1997]. De ce besoin est née l'ingénierie logicielle basée composants [Szyperski, 2002], un domaine du génie logiciel qui vise à construire des systèmes logiciels à partir de composants faiblement couplés et réutilisables. Dans [Taylor et al., 2009], un composant logiciel est une entité architecturale qui encapsule un sous-ensemble des fonctionnalités et/ou des données du système, restreint l'accès à ce sous-ensemble via une interface explicitement définie et dépend explicitement de son contexte d'exécution requis. Dans la spécification de UML 2.5.1 [Obj, 2017], un composant est une unité substituable qui peut être remplacée à la conception ou à l'exécution par un composant offrant une fonctionnalité équivalente basée sur la compatibilité de ses interfaces. Tant que l'environnement est pleinement compatible avec les interfaces fournies et requises d'un composant, il pourra interagir avec cet environnement. De même, un système peut être étendu en ajoutant de nouveaux types de composants qui ajoutent de nouvelles fonctionnalités. Un composant possède une ou plusieurs interfaces fournies et/ou requises potentiellement exposées via des ports liés à des connecteurs.

Il existe une variété étendue de modèles de composants couvrant un large éventail de concepts (cycle de vie, type d'interface, mécanismes de liaison et d'interaction, etc.), nous citons comme exemples : Fractal [Bruneton et al., 2006], CCM [OMG, 2006], BIP [Basu et al., 2006], SOFA [Bures et al., 2006], Rubus [Hanninen et al., 2008] , FlowVR [Lesage et Raffin, 2012] et ROBOCOP [Maaskant, 2005]. Un framework de classification et de comparaison des modèles de composants est présenté dans [Crnkovic et al., 2011].

L'adaptation dynamique dans les systèmes basés sur les composants est possible grâce à des mécanismes de liaison tardive (late binding), qui permettent de coupler les composants à l'exécution, juste avant leur invocation, via des interfaces bien définies.

Les composants peuvent être vus comme des boîtes noires dont l'implémentation ou la structure interne est inconnue à l'avance. Cela favorise la réutilisation, la substituabilité et la séparation des préoccupations. D'après [Szyperski, 2002], la réutilisation de composants de type boîte blanche et boîte grise risque fort de compromettre la substituabilité des composants. Dans ces situations, il est nécessaire d'établir des conventions explicites concernant les informations relatives à l'implémentation et aux modifications apportées aux composants pour garantir leur substituabilité.

#### 2.6.3.2 Ingénierie logicielle orientée services

Dans l'approche orientée services [Papazoglou, 2003], un service est défini comme une unité logicielle qui effectue une fonction et dont les opérations et propriétés sont décrites dans un descripteur de service ou contrat de service. C'est au moyen de ces descripteurs que différents fragments de logiciel vont interagir les uns avec les autres. Un descripteur de service contient les opérations fournies par un service donné et la manière dont ces opérations peuvent être invoquées ; il peut inclure également les propriétés non fonctionnelles du service et sa sémantique. Ce paradigme vise à réduire les dépendances entre les unités fonctionnelles, favorisant ainsi la substitutabilité. En réduisant les dépendances, chaque élément peut évoluer séparément, rendant l'application résultante plus flexible que les applications monolithiques.

Un fournisseur de services publie sa description de service ainsi que la référence à l'implémentation du service en utilisant un registre de services (parfois appelé annuaire, répertoire ou courtier de services). Les consommateurs peuvent rechercher des services dans le registre, puis les invoquer une fois qu'ils ont une référence vers l'implémentation. Cela permet la découverte, la sélection, la liaison et la composition des

services. La Figure 2.11 illustre les interactions qui ont lieu dans une architecture orientée services. Le registre de services est un rôle toujours joué par le framework, tandis que le consommateur de services et le fournisseur de services sont des rôles joués par les composants. Les principes de la SOSE sont mis en œuvre au moyen d'une architecture orientée services (Service Oriented Architecture - SOA). Deux formes différentes de dynamisme peuvent être ajoutées aux SOA : la disponibilité dynamique des services, ce qui signifie que les services peuvent changer leur état de disponibilité à tout moment pendant l'exécution, sans le contrôle de l'application et la reconfiguration dynamique, qui désigne les services dont les propriétés (et donc la description du service) peuvent être modifiées pendant l'exécution. La disponibilité dynamique permet à un système d'évoluer sans interruption grâce à la publication, la mise à jour et la suppression dynamiques des services. Une différence subtile entre la SOA de base et une SOA dynamique concerne les notifications du registre de services vers les clients de services. Ces notifications sont indépendantes de la recherche et peuvent arriver à tout moment pour informer de l'enregistrement et du retrait des services. Les clients de services peuvent ainsi choisir dynamiquement le fournisseur de services le plus adapté. La substituabilité, base du dynamisme doit être garantie par un service faiblement couplé, lié tardivement, réutilisable, indépendant du langage de programmation et transparent par rapport à sa localisation [Buyya et al., 2013].

La mise en œuvre la plus courante de SOA repose sur les services web en incluant le protocole SOAP (Simple Object Access Protocol) pour la transmission des données, le langage de description des services Web (Web Services Description Language - WSDL) pour la définition des services, et le langage d'exécution des processus métier pour les services Web (BPEL4WS) pour orchestrer les services. Une autre implémentation possible des SOA se base sur un bus de services ESB (Enterprise Service Bus) construit sur des standards comme XML, JMS (Java Message Service) ou encore les services web.

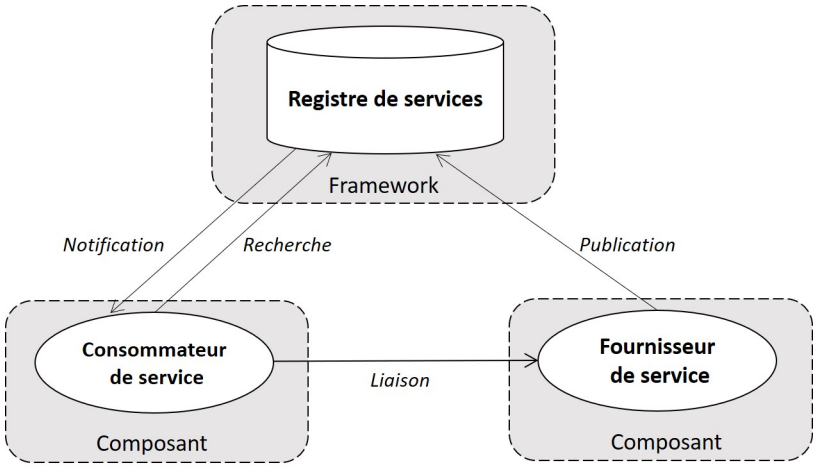



FIGURE 2.11 – *SOA : acteurs et interactions.*

### Discussion

Les concepts et principes de base de la CBSE et de la SOSE peuvent sembler similaires au premier regard, mais ils se distinguent par leurs mécanismes, leurs approches et leurs mises en œuvre. De nombreux travaux de recherche ont été consacrés à l'analyse comparative de ces paradigmes, notamment [Kim et Han, 2006], [Breivold et Larsson, 2007], [Aboud et al., 2012] et [Al-Hamed et al., 2018].

La figure 2.12 montre une comparaison qualitative présentée dans [Hock-koon et Oussalah, 2012] entre les paradigmes objet, composant et service selon six critères de qualité et trois niveaux d'importance, une comparaison au niveau conceptuel qui se révèle particulièrement intéressante pour nos travaux de thèse en raison du contexte et du niveau d'abstraction visé.

Un système basé sur des objets est constitué d'un ensemble de classes fortement couplées, tandis qu'un système basé sur des composants ou des services vise à être plus faiblement couplé ; ainsi, la question de l'adaptation dynamique est traitée de manière significative dans un contexte de CBSE et SOSE. En matière d'expressivité, l'approche orientée objet surpasse les deux autres en proposant de nombreux concepts tels que la granularité, l'héritage, le polymorphisme, le niveau d'abstraction, etc., qui pour certain n'ont pas atteint le niveau de maturité nécessaire dans la CBSE et encore moins dans la SOSE.

Grâce à un schéma de composition du comportement global (workflows et data-flows), la SOSE offre une meilleure abstraction de la communication comparée à la CBSE pour laquelle la communication est décrite de manière répartie à l'intérieur des connecteurs, ce qui compromet la compréhension de la collaboration globale de l'ensemble des composants.

En général, les systèmes basés sur des objets souffrent d'un manque d'architecture claire et aisément compréhensible. La CBSE a été créé entre autre pour remédier à ce problème et promouvoir le développement du concept d'architecture logicielle évolutive. La SOSE capitalise sur cette expérience, et ses divergences avec la CBSE sont peu significatives.

La propriété exprime le degré de liberté accordé par le fournisseur au client et dans ce contexte, la SOSE a poussé le concept de propriété à son extrême. Ainsi, le fournisseur de services est responsable de divers aspects tels que le développement, la qualité de service, la maintenance, le déploiement, l'exécution et la gestion. Cependant, dans le cadre de la CBSE, les responsabilités sont réparties au niveau du déploiement. Le client est alors responsable de l'instanciation du composant au sein de son application, ainsi que de son exécution et de sa gestion.

L'impact de ces critères peut fluctuer en fonction de la qualité spécifique visée du logiciel et du point de vue de l'utilisateur. Ainsi, pour la dynamicité du logiciel, le couplage faible, la capacité évolutive et l'abstraction de la communication ont un impact majeur par rapport aux autres critères. L'utilisation combinée des deux approches répondant aux besoins spécifiques des systèmes logiciels en tirant profit des avantages de chacune a conduit à l'émergence d'un nouveau paradigme appelé composants orientés services.

#### 2.6.3.3 Composants orientés services

Les composants orientés services résultent de l'application des principes de l'architecture orientée services aux modèles de composants. Ils ont émergé du besoin d'ajouter un dynamisme explicite et un support de disponibilité dynamique dans les modèles de composants. Dans cette approche, les composants sont liés par le biais de services ; les liaisons sont donc non explicites et réalisées aussi tard que possible (liaison tardive), permettant la substituabilité des composants (et par conséquent, des fournisseurs de services). Une application est formée par une composition dont l'architecture peut évoluer et s'adapter à l'exécution en fonction de la disponibilité des services. Le processus de sélection des composants orientés service se produit à l'exécution et l'application démarre lorsque les dépendances du composant principal sont satisfaites. Les dépendances dynamiques permettent une reconfiguration sans redémarrer le module ni

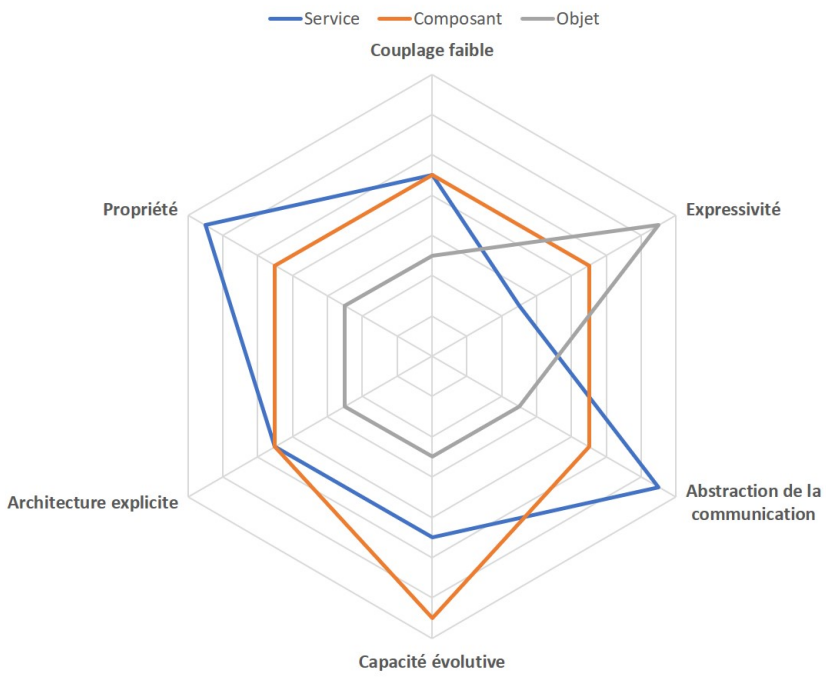



FIGURE 2.12 – *Comparaison qualitative entre paradigmes [Hock-koon et Oussalah, 2012].*

perdre son état. Elles se manifestent au niveau des services et tirent parti des principes de la SOSE. Les dépendances dynamiques influencent l'instance du composant, nécessitant une nouvelle liaison à un service compatible en cas de changement. Si aucune dépendance compatible n'est trouvée et que le service est indispensable, l'instance du composant est alors arrêtée, et ses services fournis sont retirés du registre jusqu'à ce que ses dépendances soient à nouveau résolues. L'adaptation dynamique des services peut se produire automatiquement, si elle s'exécute sur un framework de support, contrairement à l'approche basée sur les composants, où le support du dynamisme nécessite généralement plus de développement de code.

La disponibilité dynamique point fort de ce paradigme est également à l'origine du principal problème des modèles de composants orientés services : l'imprévisibilité (unpredictability). Les changements de disponibilité doivent être communiqués et pris en compte par d'autres compositions, leurs composants internes et les instances de ces composants. Lorsqu'un composant quitte le système, cela implique une recherche d'implémentations alternatives de services. Si aucune implémentation de remplacement n'est trouvée, à la fois le mécanisme de composition et le framework d'exécution doivent décider de la marche à suivre concernant la dépendance de service non satisfaite. Deux techniques sont couramment utilisées pour assurer la prévisibilité : l'analyse d'ordonnançabilité et la vérification formelle.

Dans leur projet Gravity, [Cervantes et Hall, 2004] ont exploré l'introduction de concepts orientés services dans le modèle de composants, afin de prendre en charge la liaison tardive et la disponibilité dynamique des composants. Le modèle qui en découle favorise la substituabilité des services, car les compositions et les dépendances sont exprimées en termes de spécifications. Cela permet de développer indépendamment les services constituants et d'avoir des implémentations interchangeables.

Plusieurs modèles de composants orientés services ont été proposés, la plupart étant basés sur Java, en particulier sur la plateforme de services OSGi (Open Service Gateway Initiative [Alliance, 2020] ; en voici quelques exemples : le modèle de composant iPOJO [Escoffier et al., 2007], Declarative Services (DS) [OSGi Alliance, 2019], REACT [Pfan-

nemüller et al., 2020] et Eclipse Gemini Blueprint [Eclipse Foundation, 2024]. Comme alternative, l'ensemble des spécifications du standard OASIS SCA (Service Component Architecture) [Marino et Rowley, 2009] offrent un modèle de composant orienté service agnostique du langage de programmation et du protocole de communication. L'un des aspects les plus remarquables de la spécification SCA réside dans sa capacité à permettre aux développeurs d'étendre son modèle d'assemblage pour prendre en charge de nouvelles implémentations, liaisons et interfaces [Américo et Donsez, 2012].

Le composant constitue l'unité de construction fondamentale pour SCA. Un composant est une instance configurée d'une implémentation, où l'implémentation représente la partie du code fournissant des fonctions métier. Ces fonctions métier sont proposées à d'autres composants sous forme de services. Les implémentations peuvent dépendre des services fournis par d'autres composants, et ces dépendances sont appelées références. De plus, les implémentations peuvent posséder des propriétés configurables, qui sont des valeurs de données influençant le déroulement de la fonction métier. Enfin, le composant configure l'implémentation en fournissant des valeurs pour les propriétés et en reliant les références aux services fournis par d'autres composants.

SCA décrit le contenu et les liens d'une application sous forme d'assemblages appelés composites. Comme le montre la figure 2.13, ces composites peuvent contenir des composants, des services, des références, des déclarations de propriétés, ainsi que les connexions qui décrivent les relations entre ces éléments. Les composites permettent de regrouper et de lier de manière dynamique ou statique à l'aide de liens prédéfinis appelés fils des composants construits à partir de différentes technologies d'implémentation, ce qui permet d'utiliser les technologies appropriées pour chaque tâche métier. En outre, les composites peuvent être utilisés comme des implémentations de composants complètes : ils fournissent des services, dépendent de références et possèdent des propriétés configurables. Ces implémentations de composites peuvent être utilisées dans d'autres composites, permettant ainsi une construction hiérarchique de solutions métier où les services de haut niveau sont mis en œuvre à l'aide d'ensembles de services de niveau inférieur. Les services composites impliquent la promotion d'un service de l'un des composants au sein du composite, ce qui signifie que le service composite est en fait fourni par l'un des composants du composite. Les références composites impliquent la promotion d'une ou de plusieurs références d'un ou de plusieurs composants. Diverses implémentations et frameworks pour la spécification SCA sont disponibles, qu'il s'agisse de solutions commerciales comme : Websphere d'IBM [5] et Tibco ActiveMatrix de Tibco Software [6], ou de solutions open source comme : FraSCAti [Seinturier et al., 2009; 2012].

## 2.7 Conclusion

Dans ce chapitre, les principes fondamentaux de l'IDM ont été présentés, mettant en évidence son objectif de modéliser de manière cohérente l'ensemble des aspects d'un système complexe, tout en fournissant des outils permettant d'en maîtriser la complexité. Il convient de souligner que l'originalité de l'IDM ne réside pas principalement dans l'introduction des concepts de modèle ou de transformation de modèles, lesquels étaient déjà bien établis avant l'émergence de cette approche. Ce qui distingue véritablement l'IDM, c'est l'adoption d'une modélisation à plusieurs niveaux d'abstraction, rendue possible par la méta-modélisation, ainsi que la capacité à automatiser les transitions entre ces différents niveaux. Cette caractéristique confère à cette approche une

5. https ://www.ibm.com/products/rad-for-websphere-software
6. https ://docs.tibco.com/products/tibco-activematrix-businessworks

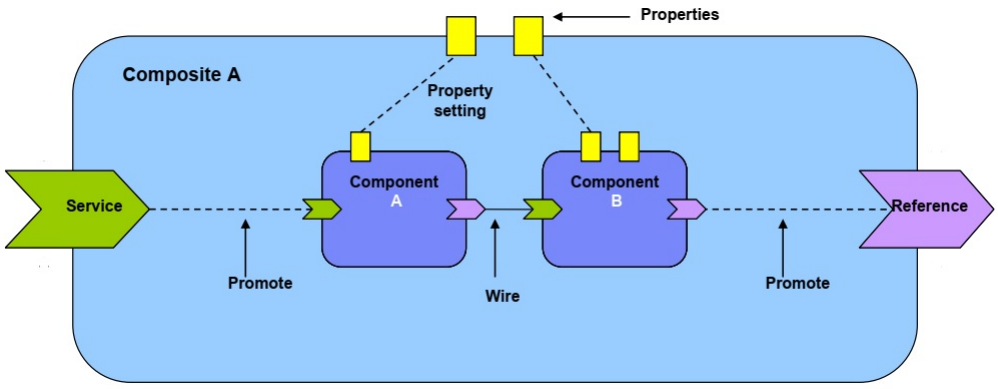



FIGURE 2.13 – *Diagramme de composite SCA [OASIS, 2009].*

pertinence particulière pour appréhender la complexité inhérente aux SoCs reconfigurables. Par ailleurs, grâce au principe de séparation des préoccupations offert par l'IDM, la DPR peut être gérée comme une préoccupation transverse, pouvant être abordée selon des mécanismes analogues à ceux de l'adaptation dynamique des logiciels. Dans ce contexte, et compte tenu de l'évolution constante des paradigmes de conception CBSE et SOSE pour les systèmes dynamiquement adaptatifs, l'exploration de leurs potentiels de combinaison dans le cadre de la DPR demeure un sujet de recherche intéressant.

Le chapitre suivant examine les principaux outils et flots associés à la conception des systèmes reconfigurables. Une attention particulière est portée aux solutions existantes en matière de modélisation à un haut niveau d'abstraction et de génération automatique de code, s'appuyant sur les principes de IDM.

# 3

# Etat de l'art de la conception des systèmes reconfigurables

Sommaire

## 3.1 Introduction

L'évolution rapide de la technologie des systèmes sur puces, conjuguée à la complexité croissante des applications qu'ils visent à prendre en charge, rend leur conception de plus en plus exigeante, tant sur le plan technique que temporel. Cette complexité résulte notamment du décalage persistant entre les avancées de la technologie matérielle et celles des outils de conception. Par ailleurs, l'introduction de la reconfiguration dans certains systèmes accentue davantage cette difficulté, soulignant la nécessité de recourir à une méthodologie de conception rigoureuse, capable de garantir abstraction, séparation des préoccupations, analyse et automatisation. Un tel cadre méthodologique vise à alléger la charge des concepteurs tout en améliorant leur productivité.

Dans cette perspective, le présent chapitre s'intéresse à un ensemble d'outils et de flots de conception, qu'ils soient commerciaux ou académiques, dédiés à la conception de systèmes reconfigurables. Il présente ensuite un certain nombre de travaux selon une approche bottom-up, visant la modélisation de la DPR à un niveau d'abstraction élevé. L'accent est mis en particulier sur l'utilisation d'UML en tant que langage de modélisation, et de SystemC en tant que langage de description matérielle. La complémentarité entre ces langages favorise une transition fluide entre les niveaux de modélisation abstraite et les phases de description détaillée, tout en assurant l'automatisation de la génération de code tout au long du processus de conception.

## 3.2 Outils de conception et d'implémentation des systèmes reconfigurables

En pratique, une adoption généralisée de la DPR dépendra de l'existence de chaînes d'outils efficaces qui offrent une vue de haut niveau d'abstraction aux concepteurs, en intégrant une compréhension détaillée de l'architecture sous-jacente. Dans ce qui suit, nous passons en revue les outils de conception, d'implémentation et de simulation pour les systèmes dynamiquement reconfigurables provenant à la fois de l'industrie et de la communauté de recherche.

### 3.2.1 Flots de conception des fournisseurs pour la DPR

Xilinx et Altera proposent des flots de conception très similaires avec quelques différences dues aux variations architecturales ; tous les deux requièrent des concepteurs experts de l'architecture de bas niveau du FPGA.

**Le flot Xilinx PlanAhead pour la DPR** [1] **:** Adopté par Xilinx après la suite logicielle Xilinx ISE, PlanAhead est un outil de conception hiérarchique basé modules offrant une interface complète pour créer et vérifier des conceptions RTL en utilisant Verilog ou VHDL. Les modules sont donc décrits soit en HDL soit sous forme de netlists pré-synthétisées. La conception matérielle se compose de deux parties : une région statique et une ou plusieurs régions reconfigurables (Partial Reconfigurable region - PRR). Les PRRs peuvent contenir divers éléments comme des LUT, des BRAM et des slices DSP, mais ils ne peuvent pas inclure de logique de modification d'horloge. La région statique correspond à la partie de la conception qui ne modifie pas sa fonctionnalité pendant le fonctionnement du système. Elle contient généralement un processeur exécutant le logiciel de gestion de reconfiguration, une interface de configuration interne et des modules d'interface mémoire. Les PRRs implémentent les modules reconfigurables et peuvent être reconfigurés à l'exécution. Une seule PRR peut implémenter plusieurs modules

1. https ://www.xilinx.com/products/design-tools/planahead.html

de manière multiplexée dans le temps ; tous les modules reconfigurables implémentés dans la même PRR constituent une partition reconfigurable. Après le partitionnement, chaque module individuel est synthétisé pour générer une netlist correspondante. Le floorplanning doit être effectué manuellement et les détails sont stockés dans le fichier UCF (User Constraints File) pour être incorporés à l'implémentation. Le concepteur doit ensuite déterminer les combinaisons valides de modules attribués aux PRRs pour constituer les modes globaux du système ; chaque combinaison valide est appelée configuration. Pendant l'implémentation, la région statique est implémentée une seule fois, avec la première configuration utilisée comme espace réservé, et le placement final et le routage de la région statique sont préservés pour toutes les autres configurations. La logique mise en œuvre dans la région statique peut utiliser les ressources de routage (mais pas les LUTs ou les bascules) disponibles dans les PRRs. Si un module reconfigurable devait utiliser des ressources de routage dans la région statique, cela provoquerait des anomalies pendant la reconfiguration. Les bus macros jouent un rôle important ici en assurant l'interface entre la région statique et les PRRs. L'outil les insère automatiquement et les concepteurs n'ont aucun contrôle sur leur emplacement. Enfin, l'outil génère un bitstream complet de reconfiguration ainsi que des bitstreams partiels pour chaque PRR et pour chaque configuration. Cela se traduit par des bitstreams complets pour chaque configuration et des bitstreams partiels correspondant au produit cartésien des modules attribués à chaque région. À l'exécution, le FPGA est initialement configuré en utilisant l'un des bitstreams complets, puis n'importe quel PRR individuel peut être reconfiguré en utilisant un bitstream partiel.

**Le flot Xilinx Vivado pour la DPR [2] :** À partir des FPGA de la série 7, Xilinx prend en charge la reconfiguration partielle via la suite de conception Vivado. Ce flot est très similaire à PlanAhead, les conceptions sont implémentées en utilisant un flot de commande basé sur TCL [3] (Tool Command Language) de Vivado ou une combinaison de commandes TCL et de l'interface utilisateur graphique. Initialement, les modules statiques et reconfigurables sont synthétisés séparément à l'aide des outils Xilinx ou de tiers. L'inclusion d'un contrôleur de reconfiguration (contrôleur ICAP) est nécessaire si le FPGA cible n'est pas un dispositif hybride tel que le SoC Zynq, bien que pour les dispositifs SoC, l'instanciation de l'ICAP soit facultative car la DPR est prise en charge via l'interface PCAP (Processing Configuration Access Port) dans le système de processeur. Les restrictions du floorplanning sont similaires à PlanAhead, avec une restriction supplémentaire pour les FPGA de la série 7 qui interdit aux limites des partitions de traverser les tuiles d'interconnexion, des ressources spéciales qui gèrent le routage entre différentes colonnes de ressources. Une innovation majeure de Vivado est l'implémentation de la logique d'ancrage, qui évite l'utilisation de bus macros en utilisant directement les tuiles d'interconnexion, améliorant ainsi l'efficacité du routage et les performances de synchronisation. Cependant, cela complique encore plus la relocalisation dynamique du bitstream en cours d'exécution. En raison de la possibilité pour les fils de la région statique de traverser les PRRs, toute modification mineure de la logique statique nécessite une réimplémentation complète de la région statique et de tous les PRRs. L'utilisation de ressources de routage dans les PRRs peut également entraîner une congestion de routage pour l'implémentation de la configuration ultérieure. Parallèlement, limiter la région statique à ne pas utiliser les ressources de routage des PRRs pourrait impacter négativement les performances de synchronisation globales du système.

2. https ://www.xilinx.com/products/design-tools/vivado.html

3. TCL est un langage de script utilisé par de nombreux outils de conception électronique, dont Vivado de Xilinx.

**Le flot Xilinx Dynamic Function eXchange (DFX)** [4] **:** Disponible en tant que fonctionnalité au sein de la suite de conception Vivado à partir de la version 2019.1, DFX permet aux concepteurs de modifier dynamiquement des parties de leurs conceptions FPGA à la volée. Les concepteurs peuvent charger des bitstreams partiels sur le FPGA pendant que la logique restante continue de fonctionner. Cela ouvre la possibilité pour des modifications de conception en temps réel et des améliorations de performances. Les utilisateurs peuvent mettre en œuvre des conceptions en utilisant le flot basé Tcl en mode non projet [5] ou bien les flots de projet basés sur RTL ou IP. Le support de conception avec IP Integrator (IPI) a été introduit dans la version 2021.1 grâce à l'utilisation de conteneurs de conception par blocs. Les points d'entrée du flot de conception incluent les langages de haut niveau traités via Vitis et HLS. Des fonctionnalités avancées telles que DFX imbriqué, qui permet aux utilisateurs de subdiviser une région dynamique en régions dynamiques de plus bas niveau, et Abstract Shell, qui rationalise le flot de l'outil d'implémentation en améliorant considérablement le temps d'exécution, sont disponibles en mode non projet.
Quatre éléments IPs sont disponibles pour aider les concepteurs à achever plus rapidement et plus facilement les conceptions DFX. Le contrôleur DFX est un contrôleur de configuration matériel qui peut aider à gérer tous les aspects des événements de reconfiguration, du déclenchement et de l'arbitrage jusqu'à la livraison du bitstream et la gestion des erreurs. Le découpleur DFX peut être utilisé avec le Contrôleur DFX ou avec tout autre contrôleur client pour isoler en toute sécurité la région dynamique pendant la reconfiguration. Le gestionnaire d'arrêt AXI DFX permet aux utilisateurs de cesser l'activité sur les interfaces AXI afin que les partitions reconfigurables puissent être reconfigurées en toute sécurité. Le moniteur de Bitstream DFX permet aux utilisateurs de déboguer et de surveiller les bitstreams partiels, garantissant la compatibilité avec la version de la cible.
La plupart des dispositifs de la série 7 et des dispositifs Zynq 7000 prennent en charge la DFX, à l'exception des plus petits dispositifs au sein de ces familles ; alors que certains Artix 7 et tous les Spartan 7 ne sont pas pris en charge. Le support UltraScale est complet, avec tous les dispositifs pris en charge jusqu'à la génération du bitstream dans la version actuelle de Vivado Design Suite. Le support des dispositifs UltraScale+ couvre tous les dispositifs en production. Le support de Versal a été ajouté avec le statut de production dans Vivado 2021.1.

**Le flot Xilinx Vitis HLS** [6] **:** Faisant partie du Vitis unified software platform et étroitement intégré à Vivado Design Suite pour la synthèse et le placement et routage, l'outil Vitis HLS [7] permet aux concepteurs de convertir des descriptions algorithmiques de haut niveau, écrites en langages C et C++, en code RTL spécifique à l'implémentation souhaitée sur régions de logique programmable d'un Versal Adaptive SoC, d'un Zynq MPSoC ou d'un dispositif FPGA AMD. Selon le flot de développement représenté sur la figure 3.1, Vitis HLS permet d'accéder à un niveau d'abstraction supérieur, en exploitant des bibliothèques optimisées et des bancs d'essai basés sur C pour une validation précoce, le débogage et la vérification fonctionnelle tout en réduisant le besoin d'itérations pendant la phase de conception matérielle. Il inclut une variété de directives et de pragmas d'optimisation pour aider à atteindre les performances de conception souhaitées.

4. https ://www.xilinx.com/products/design-tools/vivado/dynamic-function-exchange.html

5. Contrairement au mode projet, ce mode d'utilisation est destiné aux utilisateurs de scripts qui ne souhaitent pas que les outils Vivado gèrent leurs données de conception ou suivent l'état de leur conception.

6. https ://www.xilinx.com/products/design-tools/vitis/vitis-hls.html

7. À partir de la version 2020 de ses outils de développement, Xilinx a remplacé Vivado HLS par Vitis HLS. Cette nouvelle version applique davantage d'optimisations et repose sur le protocole d'interface AMBA AXI4 pour la communication avec la mémoire hors puce par défaut.

Il est possible d'utiliser la fonction d'analyseur de code de l'outil pour visualiser les estimations de performances, identifier le potentiel de parallélisme des tâches et repérer les changements architecturaux nécessaires pour optimiser les performances avant de lancer la synthèse en C. Une fois la synthèse en C terminée, un rapport récapitulatif fournit des informations sur le timing, les performances, l'utilisation des ressources, et plus encore. Vitis HLS permet de modifier des paramètres (comme la fréquence de base ou la résolution vidéo) dans le code C et laisser l'outil effectuer la conversion en RTL, réduisant ainsi la complexité et les risques d'erreur. De plus, il est possible de réutiliser des fonctions C/C++ existantes, ce qui augmente la productivité de conception. Le concepteur peut apporter des modifications rapidement en fonction des résultats de simulation, des métriques de performance ou d'autres commentaires dans l'outil, avant de régénérer le RTL. Ainsi il peut se concentrer sur les algorithmes sans être bloqué par les complexités matérielles. L'outil Vitis HLS est livré avec un ensemble robuste de fonctions de bibliothèque qu'il est possible d'instancier et qui sont optimisées pour les architectures SoC adaptatives et FPGA AMD. Il simplifie la création de IPs personnalisées et des fonctions puis les intégrer dans la suite de conception Vivado ou le compositeur de modèles Vitis pour construire une conception complète. Désormais, il devient possible de développer facilement et rapidement des conceptions hétérogènes pour les SOCs adaptatifs et les FPGA AMD en utilisant des solutions logicielles puissantes et complémentaires couvrant l'ensemble du flot de conception.
Bien que Vitis HLS offre une approche plus abstraite et conviviale pour la conception FPGA, une compréhension approfondie de la façon dont le compilateur HLS traduira une description de haut niveau en matériel reste nécessaire. Des mécanismes sous-jacents, comme la profondeur et la largeur des interfaces de bus, doivent être précisés par le concepteur ou seront inférés par le compilateur. Cela peut entraîner des conséquences indésirables pour l'optimisation et l'implémentation fournies par les outils.

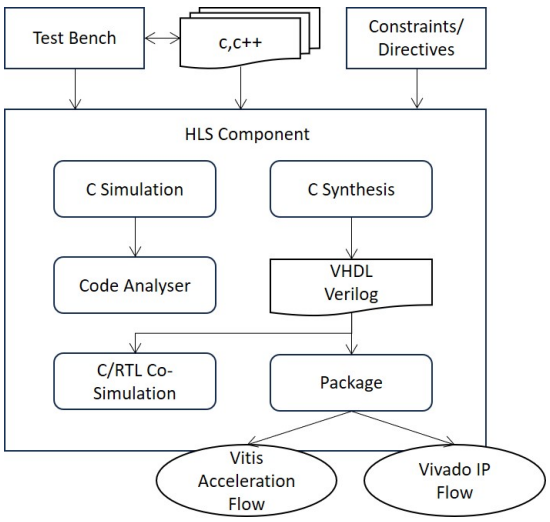



FIGURE 3.1 – *Flot de développement Xilinx Vitis HLS [AMD, 2023].*

**Le flot Altera pour la DPR :** Le flot pour la DPR chez Altera, supporté à travers les outils Quartus-II et Quartus Prime[8], présente des similitudes avec le flot utilisé dans PlanAhead, bien qu'il utilise une terminologie différente. Altera désigne les trames de configuration comme des trames de programmation et qualifie les configurations de révisions. Les variations de modules implémentées dans la même région reconfigurable sont appelées des personas. La révision initiale est la révision de base, où les limites de

8. https ://www.intel.com

la région statique et des PRRs sont définies. À partir de la révision de base, plusieurs révisions peuvent être créées. Les limites des PRRs sont fixées à l'aide des affectations LogicLock dans Quartus. La logique d'ancrage basée sur les LUTs est automatiquement insérée par l'outil pour corriger l'acheminement entre la région statique et les régions reconfigurables. Plus tard, la technique de compilation incrémentielle est utilisée pour préserver la région statique entre différentes révisions. Contrairement au flux d'outils Xilinx, Quartus crée un fichier de configuration complet uniquement pour la révision de base, avec seulement des fichiers de configuration partielle correspondant à chaque PRR généré pour les autres révisions. Les FPGA d'Altera permettent également le routage statique à travers les PRR, ce qui nécessite une réimplémentation complète en cas de modification de la région statique.

### 3.2.2 Outils de développement académiques pour la DPR

Les outils actuels des fournisseurs exigent généralement soit une expertise approfondie de la plateforme, étroitement intégrée dans le flot de développement du début à la fin, soit ils isolent chaque étape, nécessitant que les tâches de conception et d'implémentation soient gérées par différents experts spécialisés dans leurs domaines respectifs. La plupart des outils académiques ciblent les FPGA Xilinx et beaucoup utilisent les outils du fournisseur pour les opérations dépendantes du dispositif au niveau bas telles que le placement, le routage et la génération de fichiers de configuration.

**L'outil GoAhead** [Beckhoff et al., 2012] vise à surmonter les limites du flot de conception incrémentielle de Xilinx. Afin de faciliter la relocation des modules entre les PRRs, il empêche les ressources de routage dans les PRRs d'être utilisées par la région statique. Les modules statiques et reconfigurables sont implémentés à travers des flots de conception indépendants. Une fois le plan initial définissant les parties statiques et reconfigurables est élaboré, le floorplanning est effectué manuellemnt à l'aide d'un outil graphique et des boîtes de délimitation sont tracées autour des régions reconfigurables. GoAhead implémente les parties statiques et reconfigurables de la conception en utilisant des blocker macros pour contraindre le routage. Enfin, les outils du fournisseur sont utilisés pour générer des bitstreams partiels et complets à partir de la conception routée. GoAhead peut surmonter certaines des limitations du flot du fournisseur, mais il ne résout pas les problèmes de conception à un haut niveau abstraction et nécessite donc une expertise en conception du FPGA. GoAhead a été complété par une fonctionnalité de floorplanning automatique, cependant, sa dépendance à l'égard de Xilinx Design Language (XDL) pour gérer le placement des blocker macros pose problème, car ce langage a été abandonné dans le flot de conception Vivado [Vipin et Fahmy, 2018].

**CoPR** [Vipin et Fahmy, 2015] est un framework de conception entièrement automatisé ciblant l'architecture Zynq. L'objectif étant l'élévation du niveau d'abstraction pour décrire des applications partiellement reconfigurables. Plusieurs étapes manuelles nécessaires dans le flot du fournisseur sont automatisées, et les détails dépendants de l'architecture FPGA de bas niveau sont abstraits au concepteur. CoPR prend en entrée des spécification de configuration et d'adaptation ; la spécification de configuration détaille les différentes configurations système valides et les modules de bibliothèque correspondants présents dans chaque configuration au format XML. La spécification d'adaptation contient du code logiciel pour changer les configurations pendant l'exécution. Aucune de ces spécifications ne fait référence à des propriétés de bas niveau de la DPR, rendant CoPR accessible aux non-experts. L'outil Xilinx Synthesis Technology (XST) est utilisé pour la synthèse de tous les modules du FPGA cible. CoPR automatise le partitionnement des modules en régions, le floorplaning sur le FPGA et la génération de bitstreams partiels. La conception DPR est ensuite intégrée au système du

processeur ARM dans le Zynq en uilisant Xilinx Platform Studio (XPS). Les opérations d'implémentation de bas niveau et de génération de bitstream sont réalisées à l'aide des commandes du Xilinx Software Command-Line Tool (XSCT). Le logiciel pour gérer les opérations de reconfiguration de bas niveau est automatiquement généré par l'outil en langage C, puis intégré à la spécification d'adaptation de haut niveau à l'aide de la chaîne d'outils Xilinx Software Development Kit (SDK). Le processeur ARM exécute le système d'exploitation autonome de Xilinx et gère la reconfiguration via un contrôleur de reconfiguration personnalisé et un pilote associé. La chaine d'outils CoPR s'intègre avec les outils Xilinx ISE, XPS et SDK pour l'implémentation en backend, mais n'est pas prise en charge par Vivado.

**IMPRESS** [Zamacola et al., 2018] est un outil visant à automatiser la génération de systèmes dynamiquement reconfigurables en se basant sur un ensemble de scripts TCL fonctionnant sous Xilinx Vivado et permettant la génération de bitstreams partiels relocalisables. Il permet de produire des partitions reconfigurables avec une granularité plus fine, réduisant la quantité de blocs logiques gaspillés par les modules reconfigurables avec des exigences en ressources plus faibles. IMPRESS élimine bon nombre de contraintes imposées par les outils commerciaux en permettant la relocalisation des modules reconfigurables, l'empilement de plusieurs partitions reconfigurables dans une région d'horloge, la reconfiguration hiérarchique, en autorisant la communications entre modules reconfigurables et en découplant l'implémentation des parties statique et reconfigurable. Cependant, il impacte la taille des bitstreams partiels et augmente ainsi davantage le temps nécessaire pour les charger via l'interface PCAP.

**ARTICo$^3$** [Rodríguez et al., 2018] est un framework pour la conception des systèmes embarqués adaptatifs de haute performance pour le edge computing. Comme le montre la figure 3.2, il repose sur trois éléments : une architecture pour l'accélération matérielle flexible, une chaîne d'outils automatisée pour construire des systèmes reconfigurables basés sur un FPGA, et un environnement d'exécution pour gérer les applications actives. L'architecture $ARTICo^3$ offre un calcul embarqué à haute performance, dynamiquement adaptable et basé sur le matériel, exploitant à la fois le parallélisme au niveau des tâches et des données. L'architecture offre un support pour établir un compromis entre la performance de calcul, la consommation d'énergie et la tolérance aux pannes pendant l'exécution, permettant ainsi aux concepteurs d'explorer dynamiquement l'espace des solutions pour n'importe quelle application donnée. L'utilisation de la DPR, associée à un partitionnement basé sur des slots dans la matrice reconfigurable, permet la réplication de modules et, dans certains cas, même leur relocation. Ainsi, des accélérateurs matériels (noyaux) spécifiques à l'application peuvent être instanciés autant de fois que nécessaire, à condition qu'il y ait suffisamment d'espace disponible, afin d'exploiter au mieux le parallélisme aux niveaux des tâches et des données. Une infrastructure de communication optimisée, basée sur un bus et un DMA, est utilisée pour alimenter les accélérateurs et récupérer les données une fois le calcul terminé. L'architecture proposée est accompagnée d'une chaîne d'outils permettant de générer automatiquement des systèmes basés sur $ARTICo^3$ à partir de descriptions de haut niveau en C/C++ à la fois de l'application et des accélérateurs matériels, en utilisant la synthèse de haut niveau HLS. Les conceptions existantes écrites en langage de description matériel HDL au niveau RTL sont également prises en charge. Ainsi, la DPR peut être mise en œuvre sans connaissances préalables des détails de l'architecture cible du FPGA. L'environnement d'exécution $ARTICo^3$ s'exécute dans l'espace utilisateur Linux, écrit en langage C, son API utilise un module de plateforme personnalisé pour gérer la mémoire virtuelle/physique ainsi que le contrôle DMA. Bien que ce module de plateforme noyau soit relativement léger, cela implique que toutes les modifications apportées au noyau Linux doivent être intégrées dans le framework. En outre, la surcharge excessive et non

uniforme générée par le système d'exploitation lui-même affecte les fonctionnalités de base du framework, telles que l'exploration de l'espace des solutions.

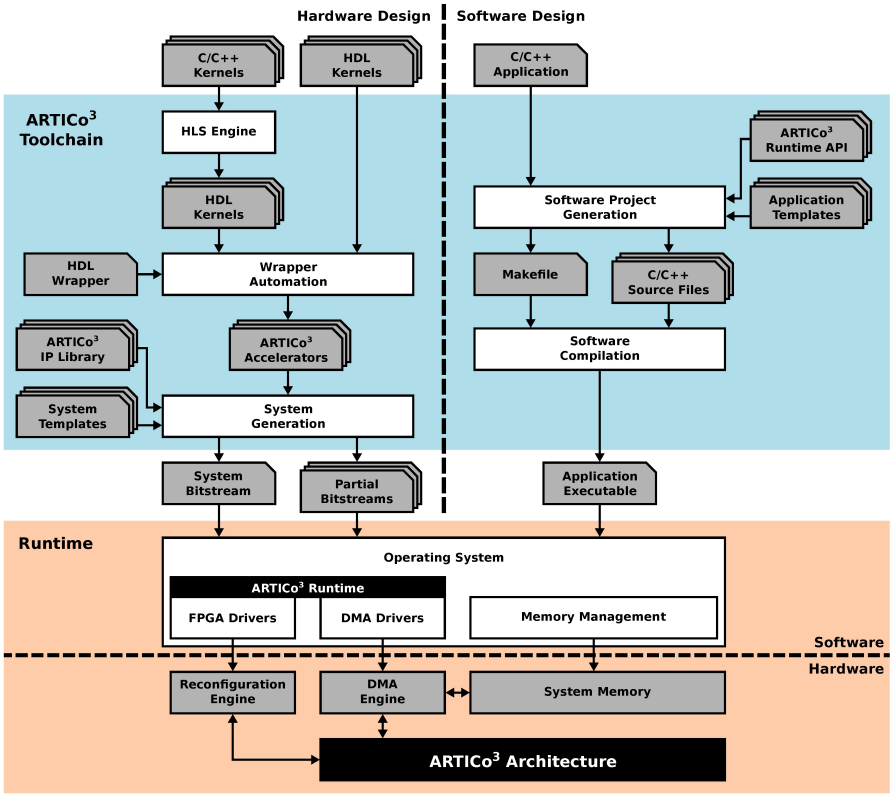



FIGURE 3.2 – *Le framework ARTICo³ [Rodríguez et al., 2018].*

**ZyPR** [Bucknall et Fahmy, 2023] est un framework complet de bout en bout qui permet une reconfiguration haute performance du matériel à partir d'une abstraction logicielle dans l'espace utilisateur Linux. Il automatise la création d'applications partiellement reconfigurables en prenant en charge les architectures Xilinx Zynq et Zynq UltraScale+. L'objectif est de permettre aux concepteurs d'applications non experts d'utiliser la DPR pour les applications de périphérie. ZyPR extrait la logique des applications utilisateur, en extrait les interfaces, et génère l'infrastructure au moment de la compilation pour prendre en charge le transfert de données entre le PS (Processing System) et la PL (Programmable Logic) ; ce qui simplifie la complexité du processus de construction DPR et la transition vers PetaLinux[9] pour la compilation du noyau Linux et la création du système de fichiers. ZyPR intègre également un gestionnaire de configuration haute performance qui utilise l'ICAP renforcé des architectures Zynq et Zynq Ultrascale+ pour charger les bitstreams de reconfiguration partielle dans le PL à des débits théoriques proches de 388 MiB/s et 757 MiB/s pour les Zynq et ZynqMP, respectivement. Le runtime ZyPR assure simultanément la reconfiguration partielle et le contrôle matériel, en utilisant des abstractions pour les configurations et les modes. Les modes définissent les accélérateurs matériels fonctionnels qui peuvent être chargés et déchargés du FPGA en utilisant la DPR comprenant les bitstreams partiels ; une configuration désigne un agencement fonctionnel de modes qui peut exécuter une fonction d'accélération abstraite. Bien que ZyPR offre des avantages en termes de performances et de comportements fonctionnels comparé aux outils du fournisseur, des améliorations res-

9. PetaLinux est un outil de développement fourni par AMD/Xilinx qui permet de créer, configurer et construire des systèmes d'exploitation Linux personnalisés pour les SoCs basés FPGA de Xilinx (Zynq-7000, Zynq UltraScale+ MPSoC, Versal, etc.).

tent possibles notamment en ce qui concerne la simplification d'écriture du code RTL ; la prise en charge davantage de HDLs autres que Verilog ; l'optimisation floorplanning et l'ordonnancement dynamique des ressources reconfigurables.

**HiPR (High-level Partial Reconfiguration)** [Xiao et al., 2024] est un framework open source pour la définition de fonction C/C++ partiellement reconfigurables au lieu de modules Verilog pour accélérer la compilation incrémentale du FPGA et automatiser le processus de génération de bitstreams sans expertise de bas niveau. Entièrement compatible avec le dernier flot Xilinx Vitis ; et comme le montre la figure 3.3, le flot HiPR prend en entrée des applications basées sur le modèle de calcul KPN (Kahn Processing Networks) et utilise des pragmas pour identifier une fonction en cours de développement et signaler qu'elle doit disposer de sa propre région de reconfiguration partielle afin de permettre une recompilation rapide. Lors de la première compilation de l'application, chaque fonction (opérateur dans KPN) en C est compilée en une netlist de niveau post-synthèse RTL. En utilisant les exigences en ressources de la synthèse RTL, HiPR génère automatiquement un overlay spécifique à la conception avec une région statique et des régions partiellement reconfigurables cibles personnalisées. Lorsque l'utilisateur modifie uniquement la ou les fonctions cibles, HiPR ne recompile que les fonctions modifiées. Si l'utilisateur désire modifier l'interconnexion entre différents opérateurs ou ajouter davantage de fonctions cibles pour la DPR, HiPR redéfinira automatiquement le floorplan pour les régions statiques et reconfigurables. L'algorithme traditionnel de floorplanning par recuit simulé est étendu avec de nouvelles fonctionnalités pour la DPR afin d'automatiser le floorplanner ; celui-ci peut générer des contraintes pour les fonctions reconfigurables en quelques secondes. HiPR peut réduire ainsi le temps de compilation incrémentale d'un facteur de 3 à 10 fois sans perte de performance et indépendamment de la taille des régions reconfigurables.

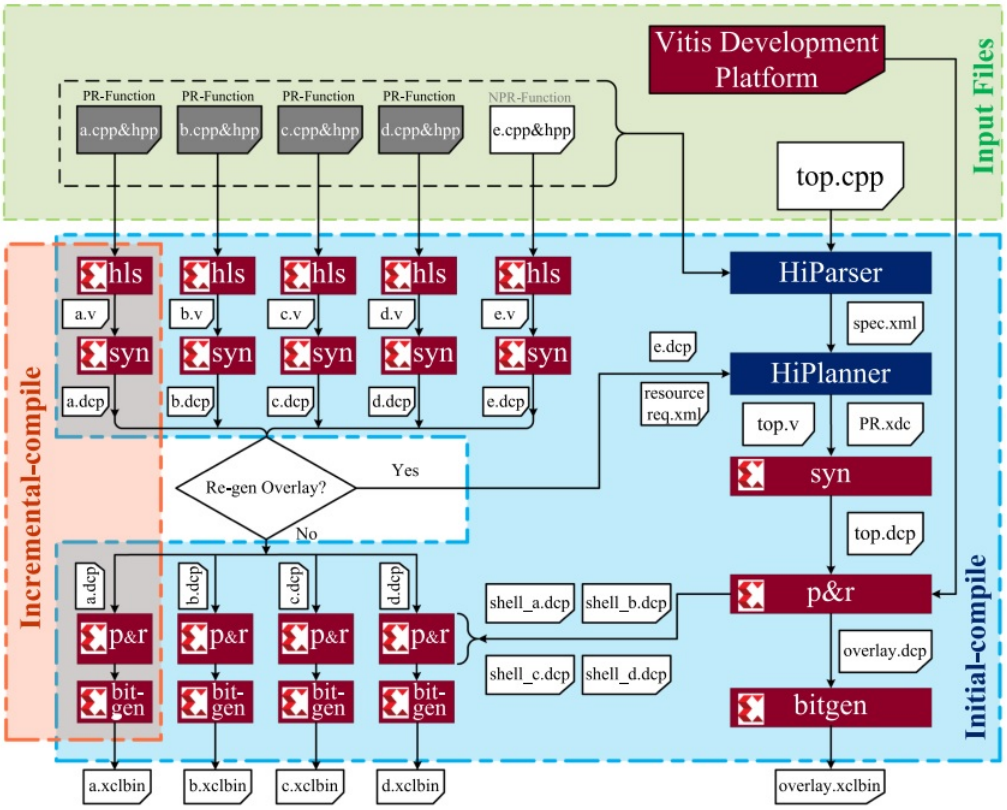



FIGURE 3.3 – *Flot de conception HiPR [Xiao et al., 2024].*

### 3.2.3 Discussion

Bien qu'il existe une large panoplie d'outils académiques open source visant à simplifier le processus DPR, il n'est pas possible de trouver un seul outil qui puisse prendre en charge tous les aspects du développement d'applications DPR et répondre à ses défis. Les noyaux matériels et les couches logicielles nécessaires pour communiquer avec la logique de l'accélérateur, sont courants dans les différents outils et poussent les concepteurs à apprendre à utiliser des workflows personnalisés pour concevoir et envelopper leurs fonctions d'accélération. Un certain nombre d'inconvénients liés à l'omniprésence de noyaux matériels dans les différents outils inclus :

— L'obligation d'apprendre et de mettre en œuvre la logique de l'accélérateur conformément aux spécificités de la chaîne d'outils cible.
— L'adaptation du logiciel d'application pour interagir avec des APIs spécifiques à l'outil.
— Une surcharge logicielle supplémentaire résultant des multiples couches d'abstraction pour accéder au matériel.

En plus, beaucoup de gestionnaires d'exécution inclus dans ces outils utilisent le pilote FPGA Manager de Xilinx pour la reconfiguration, ce qui limite le débit et la latence potentiels de la DPR à ce qui est disponible avec l'interface PCAP [Bucknall et Fahmy, 2023].

## 3.3 Modélisation à haut niveau de la DPR et génération automatique de code

Le langage SystemC est généralement reconnu comme un HDL de haut niveau, offrant des capacités avancées de modélisation du matériel et du comportement. À un niveau d'abstraction supérieur, le langage UML peut être judicieusement employé pour la spécification et la conception de systèmes complexes, en particulier lorsqu'une approche orientée modèle est privilégiée. Dans ce contexte, la modélisation de la DPR sera abordée selon une démarche bottom-up, en partant des niveaux inférieurs d'abstraction pour progressivement remonter vers des représentations plus abstraites.

### 3.3.1 SystemC pour la modélisation de la DPR

Deux courants majeurs ont longtemps dominé la recherche sur la modélisation de la DPR avec SystemC. Le premier reposait sur l'utilisation d'un noyau SystemC modifié [Brito et al., 2007; 2010], tandis que le second privilégiait une stricte conformité au standard en conservant le noyau de simulation intact. Cependant, la plupart des chercheurs préconisent l'adhésion au standard, considérant qu'un noyau de simulation modifié constitue un obstacle au processus de conception.

De plus, sous les contraintes de la conception statique de SystemC, une première approche pourrait consister à intégrer plusieurs modules statiques, chacun représentant une fonctionnalité spécifique. Lors de la simulation, un seul module serait activé de manière sélective afin de simuler, approximativement, le comportement de la DPR sur FPGA. Cependant, cette approche introduit un certain niveau de complexité, car les développeurs doivent s'assurer qu'aucun module n'est utilisé simultanément avec un autre, ce qui nécessite une gestion et une surveillance rigoureuses. Une autre approche consiste à tirer parti des constructions dynamiques introduites dans SystemC à partir de la version 2.1 ou à développer de nouvelles bibliothèques pour étendre le langage.

Dans [Sotiriou-Xanthopoulos et al., 2013], l'approche proposée repose sur un module SystemC reconfigurable, fondé sur les processus PRM (Process-based Reconfigu-

rable Module), afin d'accélérer l'exploration de l'espace de conception. Le PRM exploite des appels Unix pour gérer les processus et assurer la communication inter-processus, ce qui entraîne une dépendance au système d'exploitation. Il est structuré en trois éléments : une partie statique, une partie dynamique et un serveur de contrôle utilisateur, fonctionnant comme un serveur Unix à requête unique. La DPR est modélisée par l'affectation statique de plusieurs processus au PRM, avec une sélection dynamique de l'un d'entre eux à chaque exécution, ce qui restreint l'ajout de nouvelles configurations durant la simulation.

Les auteurs dans [Cervero et al., 2013] présentent un gestionnaire de ressources dynamiques structuré en couches, conçu pour optimiser l'ordonnancement et le contrôle des processus liés à la reconfiguration partielle des FPGA. Ils introduisent également un framework de simulation basé sur SystemC, reposant sur le concept de module dynamique. La reconfiguration y est modélisée comme un swapping entre plusieurs modèles de modules, exécutés dans des threads SystemC distincts, avec un basculement d'un modèle à l'autre au sein du module dynamique. Toutefois, une limite majeure de cette approche réside dans le fait que le nombre de threads nécessaires à l'implémentation des différents comportements d'un module dynamique doit être fixé statiquement lors de la compilation.

L'étude présentée dans [Suvorova et al., 2015] explore diverses approches de modélisation des SoCs et NoCs reconfigurables basés sur ASIC, en exploitant les fonctionnalités standard de SystemC et TLM, aussi bien au niveau système qu'au niveau RTL. Pour modéliser le concept de sélecteurs, des types spécifiques de sockets sont utilisés, permettant la connexion de différentes lignes à un même port à des instants distincts. Les blocs fonctionnels pouvant être reconfigurés dynamiquement sont représentés par des composants reconfigurables basés sur une table de correspondance (look-up table). Bien que les connexions actives soient déterminées dynamiquement au cours de la simulation, tous les composants sont générés de manière statique et les connexions possibles entre eux sont définies à l'avance. La reconfiguration dynamique d'un composant peut être mise en œuvre via des éléments reconfigurables issus d'une bibliothèque. Chaque élément reconfigurable est associé à une classe, et le comportement d'une instance particulière de cette classe peut être ajusté dynamiquement à l'aide de l'opérateur *case* en C++. Une autre solution consiste à exprimer les possibilités de reconfiguration sous forme de fonctions logiques, puis à utiliser *sc_spawn* pour générer un processus fils dynamique correspondant à la fonctionnalité requise. Toutefois, l'étude ne fournit pas d'informations sur la réutilisabilité et l'interopérabilité des composants, ce qui est particulièrement pertinent compte tenu des restrictions importantes de la reconfiguration dynamique dans les ASIC par rapport aux FPGA.

Pour assurer une simulation multiniveau efficace, [Herrera et al., 2012] proposent un flot de raffinement basé sur SystemC, A-HetSC (Adaptive Heterogeneous SystemC), permettant de transformer les processus HAPs (HetSC Adaptive Processes) en une description OSSS+R [Schallenberg et al., 2009]. Un HAP repose sur un processus SystemC exécutant une fonctionnalité adaptative transmise sous forme de pointeur de fonction par son environnement. Cette fonctionnalité est ensuite remplacée par une sélection parmi un ensemble fini de fonctionnalités adaptatives, définies statiquement dans la description OSSS+R. Dans un premier temps, le HAP, initialement modélisé sans contrainte temporelle, est affiné en une version synchronisée par horloge, tout en maintenant la fonctionnalité adaptative au sein d'un processus SC_THREAD. L'encapsulation de l'objet reconfigurable consiste à convertir les fonctions associées aux modes adaptatifs en classes d'implémentation spécifiques. Enfin, dans la dernière phase du raffinement, les threads asynchrones sont remplacés par des SC_CTHREADs, permettant d'utiliser des boucles bornées et de remplacer la gestion dynamique de la mémoire

par des structures de données statiques. Comme dans la bibliothèque OSSS+R, l'affectation des calculs adaptatifs aux modules HAP est réalisée de manière statique, ce qui empêche toute mise à jour des fonctionnalités adaptatives en cours d'exécution.

Les auteurs dans [Duhem et al., 2011] présentent une méthodologie fondée sur le diagramme en Y pour modéliser des architectures dynamiquement reconfigurables à un niveau d'abstraction élevé en utilisant SystemC/TLM. Leur approche vise, d'une part, à simplifier le développement de stratégies d'ordonnancement pour la gestion des tâches matérielles et, d'autre part, à intégrer l'exploration de l'espace de conception afin de proposer un flux de conception totalement compatible avec le processus de reconfiguration partielle de Xilinx. Dans le modèle proposé, un module reconfigurable repose sur deux threads dynamiques SystemC : *User Algorithm thread*, activé lors de l'exécution pour assurer la fonctionnalité du module reconfigurable et *Reconfig Control thread*, chargé de gérer la création et la suppression du *User Algorithm thread*. En outre, chaque module reconfigurable est doté de sockets initiateurs et cibles permettant la communication entre modules, conformément au protocole TLM2.0. L'interface de communication de chaque composant est associée à un ensemble de fonctions, appelé *socket control*, servant à connecter les sockets au module. Cette méthodologie a ensuite été utilisée pour concevoir RecoSim, un simulateur reconfigurable intégré au flot de conception FoRTReSS [Duhem et al., 2015].

La bibliothèque ReChannel [Raabe et Felke, 2008] enrichit SystemC en introduisant des constructions avancées pour la modélisation de la reconfiguration à haut niveau. Elle intègre le concept de *portails*, permettant de relier un canal de la partie statique du design aux ports des modules reconfigurables. Ces portails sont fournis pour toutes les interfaces de canaux de SystemC. La reconfiguration dynamique est assimilée à un commutateur de circuit, limitant l'activation à un seul module à la fois, sous réserve que tous ses portails puissent être commutés. Les modules reconfigurables sont dérivés de modules statiques grâce à une macro spécifique, tandis qu'un objet *rc_control* assure l'enregistrement et la gestion du processus de reconfiguration. ReChannel fournit également des extensions du langage permettant de spécifier explicitement la reconfiguration, autorisant ainsi la réinitialisation des modules sans modifier le noyau de SystemC. Cependant, les tâches sont attribuées de manière statique aux zones reconfigurables dès le début de la simulation, ce qui exclut la possibilité de mobilité des tâches. Une approche de modélisation top-down, s'appuyant sur ReChannel, a été développée pour la vérification fonctionnelle des systèmes dynamiquement reconfigurables par simulation [Gong et Diessel, 2011]. Concernant les défis liés à la reconfiguration dynamique, des bogues potentiels ont été identifiés et classés selon leur apparition avant, pendant ou après la reconfiguration, en couvrant les niveaux comportemental, TLM et RTL.

Dans [Peña et al., 2015], un framework basé sur SystemC permettant la modélisation et la simulation de la DPR tout en offrant la possibilité d'explorer précocement l'espace de conception des systèmes dynamiquement reconfigurables est présenté. Un adaptateur de communication spécifique est utilisé pour assurer la transparence entre les composants et éviter un couplage excessif entre la logique fonctionnelle et les détails d'intégration bas niveau. Pour ce faire, des services middleware de base sont mis en place, et TLM est employé comme couche de transport physique des messages. La DPR est modélisée à l'aide de bibliothèques dynamiques combinées à un adaptateur de composants, appelé unité reconfigurable, qui intègre plusieurs comportements et repose sur des plugins C++ ainsi que des threads dynamiques. Un contrôleur de reconfiguration s'occupe de charger dynamiquement les comportements associés à chaque unité reconfigurable et de configurer l'adaptateur afin de le rendre accessible aux autres composants du système. De plus, un service de localisation gère la correspondance entre les adresses physiques et logiques en traitant les requêtes de traduction de l'adaptateur.

Bien que cette séparation entre les espaces d'adressage logique et physique améliore la flexibilité des applications, le choix des composants reste tributaire de leur implémentation, ce qui constitue la principale limitation des approches basées sur les composants.

Dans [Giordano et al., 2019], un nouveau flot de conception basé sur les modèles pour la conception et la simulation des architectures reconfigurables est présenté. Le flot est fondé sur l'élaboration d'une machine à états finis étendue (Extended Finite State Machine - EFSM) offrant une description formelle du matériel, où les transitions entre les nœuds sont enrichies de fonctions d'activation et de fonctions de mise à jour. Le processus de conception s'appuie sur Pyngu, un outil qui automatise la génération de code et la simulation via SystemC en distinguant les parties statique et dynamique du système ; il prend en entrée un modèle de haut niveau de l'ensemble du système, basée sur le standard EFSM écrit en PynguLang au format YAML [10] et génère à la sortie du code SystemC conforme à la bibliothèque ReChannel [Raabe et Felke, 2008]. Afin de garantir la sureté du système, l'outil Pyngu génère une architecture maitre-esclave, divisant la partie statique en trois processus distincts en SystemC : *producteur*, *contrôleur* et *moniteur*. Le *producteur* génère et transmet des données au module reconfigurable actif en cours d'exécution, le *contrôleur* se charge de la gestion de la machine à états de l'ensemble du système et enfin le *moniteur* récupère les données de sortie du module reconfigurable actif.

Les auteurs dans [Haase et al., 2024] proposent une extension de bibliothèque pour SystemC appelée Nested-reConfiguration library, permettant la modélisation et la gestion dynamique du matériel reconfigurable sans modifier le noyau de simulation. Cette extension repose sur trois objectifs clés : la définition d'interfaces pour les régions reconfigurables, intégrant les sockets TLM et les ports SystemC standards, tout en prenant en charge des types personnalisés ; la création de modules reconfigurables par héritage des sc_modules, avec des macros pour la gestion des processus SC_THREAD, SC_METHOD et SC_CTHREAD ; et enfin une mise à disposition d'une API pour la gestion dynamique de la reconfiguration, incluant l'enregistrement des modules reconfigurables dans les régions reconfigurables à l'élaboration, le démarrage des régions reconfigurables, ainsi que la configuration dynamique des régions reconfigurables pendant la phase de simulation. Lors de la simulation, l'API assure également l'isolation des ports et des sockets durant la reconfiguration ; l'intégration des mécanismes de protection empêchant le déchargement des modules reconfigurables lors de transactions TLM bloquantes ; et en dernier lieu, la prise en charge de la reconfiguration imbriquée et divisée, améliorant ainsi la flexibilité et la granularité du processus de reconfiguration.

### 3.3.2 UML pour la modélisation des SoCs

De nombreux travaux de recherche ont porté sur la génération automatique de code, notamment SystemC, à partir de modèles UML. Certaines approches s'appuient sur le standard UML, tandis que d'autres suggèrent de l'adapter au domaine des SoCs en développant des profils spécifiques. UML 2.0 profile for SystemC [Riccobene et al., 2005], SOCP (UML Profile for SoC) [Zhu et al., 2004] [Obj, 2006] [11], le profile TUT (Tampere University of Technology) [Kukkala et al., 2005], MARTE et SysML en sont des exemples. Une comparaison des principaux profils UML 2.0 utilisés pour les systèmes embarqués et les SoCs est présentée dans [Boutekkouk et al., 2009].
La problématique de la combinaison de plusieurs profils, notamment SysML et MARTE, a été abordée dans [Espinoza et al., 2009] afin d'assurer une couverture complète des

10. YAML Ain't Markup Language : un format de représentation de données par sérialisation Unicode.
11. SOCP est devenu une norme OMG.

différents aspects nécessaires dans le domaine pluridisciplinaire des systèmes embarqués. SysML se révèle particulièrement adapté aux phases amont du processus de modélisation, notamment pour la définition des exigences, des comportements et des architectures systèmes. Toutefois, étant spécifiquement conçu pour l'ingénierie système, son usage devra être couplé à une notation orientée vers la conception détaillée et l'implémentation, afin d'assurer une continuité dans le cycle de développement.

Dans un domaine connexe, le profil R-UML (Reconfigurable UML) [Salem et al., 2016] a été conçu pour modéliser des systèmes de contrôle flexibles partageant des ressources adaptatives.

Une approche de génération automatique de code SystemC à partir de diagrammes de séquence et de diagrammes d'activité UML est présentée dans [Boutekkouk, 2010]. L'approche est appliquée aux premières phases de la conception des SoC et le code généré est destiné à l'exploration de l'espace algorithmique, la simulation et la synthèse.

Un DSML appelé AutoModel permettant de spécifier les exigences de haut niveau des systèmes embarqués à temps réel dans un langage textuel déclaratif a été proposé dans [Kahani, 2018]. Les exigences sont exprimées en termes de composants et de leurs actions, des objectifs et des contraintes de sécurité du système. En prenant les exigences spécifiées comme entrées, des techniques d'exploration de l'espace de conception sont appliquées pour générer les scénarios nécessaires à l'inférence des modèles en UML-RT (UML for Real-Time). Ces modèles, à la fois structurels et comportementaux, sont interopérables et immédiatement exploitables, notamment pour la génération automatique de code exécutable via des outils de développement dirigé par les modèles.

Dans [Carvalho et al., 2018], l'objectif est de générer automatiquement des modèles de simulation SystemC à partir de modèles UML au niveau service. Le modèle de simulation généré intègre l'ensemble de l'infrastructure de communication (ports, canaux et interconnexions) ainsi que les différents modes de communication et d'exécution définis par le concepteur.

L'approche proposée dans [Leite et Wehrmeister, 2016] permet la génération automatique de descriptions VHDL entièrement synthétisables à partir de modèles UML/-MARTE de haut niveau. En combinant les principes de l'IDM et de la programmation orientée aspects, cette approche permet de traiter les exigences fonctionnelles et non fonctionnelles de manière indépendante de la plateforme, facilitant ainsi la co-conception matériel-logiciel pour les systèmes embarqués.

La corrélation entre les concepts de MARTE et les préoccupations des systèmes temps réel et embarqués a été établie dans [Ribeiro et al., 2017]. Les profils SysML et MARTE sont utilisés pour élaborer des modèles structurels du système, mettant en évidence les caractéristiques embarquées et temps réel des exigences et des modèles de conception.

En vue de la vérification de la conception par simulation, les auteurs dans [Ebeid et al., 2015] proposent une méthodologie qui utilise les diagrammes de séquence UML/-MARTE avec des contraintes temporelles en VSL (Value Specification Language) pour générer automatiquement du code exécutable en SystemC/TLM et VHDL, intégrant des mécanismes de vérification. La simulation du modèle ainsi obtenu permet de contrôler la séquence des échanges d'informations entre les composants, tandis que les vérificateurs garantissent le respect des propriétés et des contraintes temporelles spécifiées. Pour automatiser la génération de code, l'outil front-end UML2HIF a été développé. Il analyse la description XML des diagrammes de séquence UML et produit une représentation intermédiaire sous forme de composants HIF (Heterogeneous Intermediate Format). Par la suite, les outils back-end de la suite HIFSuite sont utilisés pour convertir cette description HIF en code HDL cible. Plus précisément, l'outil HIF2SC permet la

génération de code en SystemC/TLM, tandis que HIF2VHDL est dédié à la génération de code VHDL.

Un processus automatisé pour l'analyse précoce de l'ordonnançabilité des systèmes embarqués à temps réel a été présenté dans [Magdich et al., 2020]. Le processus permet la prise en charge de la gestion des tâches dépendantes dans un contexte d'ordonnancement semi-partitionné et global, autorisant la migration des tâches. En s'appuyant sur les principes de l'IDM, ce processus vise à transformer les modèles UML/MARTE, dédiés à la modélisation et à l'analyse, en des modèles compatibles avec l'outil d'analyse SimSo. Si l'ordonnancement n'est pas possible, un retour d'information approprié doit être généré pour permettre à l'utilisateur d'ajuster les propriétés temporelles du système étudié.

La plupart des travaux cités précédemment se concentrent sur la génération automatique de code à partir de modèles UML, mais peu d'entre eux abordent la reconfiguration dynamique des systèmes à un niveau d'abstraction élevé. En effet, les approches existantes peinent à concilier l'élévation du niveau d'abstraction de la conception avec une intégration précise de la sémantique de la DPR dans les outils de modélisation. Si de nombreuses recherches privilégient l'usage de UML et/ou SystemC pour la modélisation de la DPR, d'autres s'orientent davantage vers l'optimisation au niveau RTL.

Dans [de la Fuente et al., 2016], les principes introduits dans [Peña et al., 2015] ont été repris pour concevoir un flot de conception intégré à Eclipse, spécifiquement dédié aux systèmes dynamiquement reconfigurables. Ce flot permet la génération automatique de modèles exécutables et de fichiers de programmation FPGA à partir de modèles UML/MARTE de haut niveau selon une approche par composants. Tel que montré à la figure 3.4, après la phase de modélisation, le flot propose aux développeurs deux options : la première consiste à simuler la conception via une spécification exécutable en SystemC, permettant ainsi une estimation précoce du comportement temporel du système. La seconde option s'intègre aux outils des fournisseurs existants afin de générer automatiquement les bitstreams destinés à la configuration de l'architecture reconfigurable.

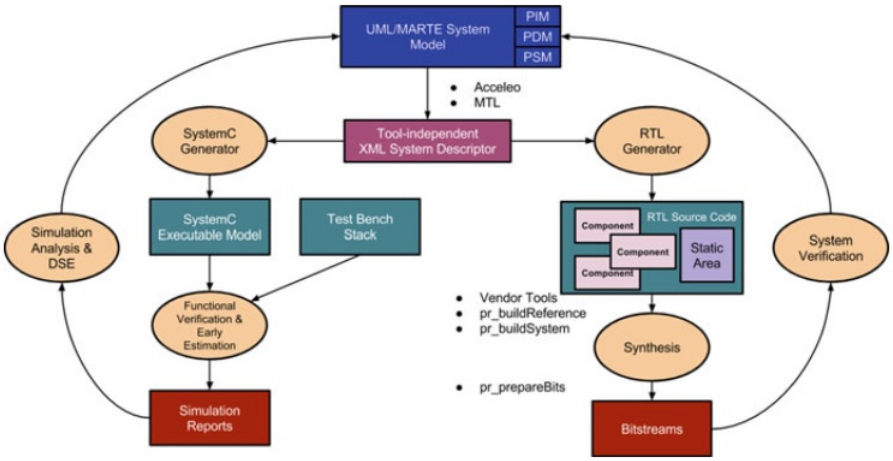



FIGURE 3.4 – *Flot de conception des systèmes dynamiquement reconfigurables [de la Fuente et al., 2016].*

Le framework de conception Gaspard (Graphical Array Specification for Parallel and Distributed Computing) pour les systèmes embarqués massivement parallèles est présenté dans [Gamatié et al., 2011]. À partir d'une spécification MARTE et en s'appuyant sur le package Repetitive Structure Modeling (RSM) dans une approche IDM, le concepteur peut générer automatiquement du code en Lustre, Fortran, C, SystemC et VHDL, facilitant ainsi la vérification formelle, la simulation et la synthèse matérielle.

Gaspard étend le profil MARTE avec le package *Deploy* afin de permettre le déploiement de composants élémentaires avec des IPs en introduisant de nouveaux concepts. Le concept de *VirtualIP* agit comme un wrapper générique permettant d'englober différentes implémentations d'un même composant élémentaire logiciel ou matériel, indépendamment de son contexte d'utilisation. Le concept *VirtualIP* est mis en œuvre par une ou plusieurs IPs, chacune définissant une implémentation spécifique à un certain niveau d'abstraction et dans un langage donné. Enfin, le concept de *CodeFile* est utilisé pour associer à une IP donnée le fichier correspondant au code source ainsi que les options de compilation requises. L'IP utilisée est sélectionnée par le concepteur du SoC en l'associant au composant élémentaire via la dépendance *Implements*. Par ailleurs, le concept *Characteristic* permet de spécifier les paramètres des IPs. Dans [Qadri et al., 2010], les auteurs proposent une sémantique de contrôle de la DPR basée sur les automates de modes et intégrée au framework Gaspard2. Les automates de modes ont été modélisés en s'appuyant sur des concepts du profil MARTE RSM, notamment la dépendance **interRepetition**, ainsi que les connecteurs **tiler** et **defaultLink**.

Dans [Vidal et al., 2011], une approche de co-conception basée sur MARTE est utilisée pour modéliser la DPR destinée aux MPSoPC (Multiprocessor System on Programmable Chip). Pour la modélisation de la plateforme, une zone reconfigurable est spécifiée à l'aide du stéréotype « **HwPLD** » du profil HRM de MARTE. Au niveau applicatif, les design patterns *strategy* et *state* sont employés pour modéliser respectivement les algorithmes de swapping dynamique et le comportement d'un objet en fonction de son état dans l'application. Le composant dynamique est modélisé à l'aide du stéréotype « *adaptive* », qui étend « **RtUnit** ». La méthode de reconfiguration adoptée peut être basée stratégie ou basée état, ce qui est respectivement représenté par les valeurs taguées *reconf_op* et *reconf_state*. Après l'affectation du composant client à un **HwProcessor** et du composant dynamique à un **HwPLD**, un code ciblant les FPGAs Xilinx est généré. Toutefois, l'extension de MARTE proposée dans ce travail reste assez limitée pour modéliser pleinement les concepts fondamentaux de la DPR.

Dans le cadre du projet FAMOUS (FAst Modeling and Design FlOw for Dynamically ReconfigUrable Systems), dont le flot de conception est illustré à la figure 3.5, une méthodologie de co-conception des systèmes dynamiquement reconfigurables basée sur l'IDM est présentée dans [Cherif, 2013]. Pour modéliser les concepts de la DPR, une extension de MARTE, appelée RecoMARTE, a été proposée. Au niveau applicatif, le stéréotype MARTE « **RtUnit** » est étendu en « *ReconfigurableRtUnit* » pour décrire les tâches reconfigurables. Pour l'interconnexion des composants, le stéréotype « *ExtendedFlowPort* » est utilisé afin de spécifier le type de port. Au niveau allocation, trois stéréotypes sont introduits : « *Deployed* » , « *IP* » et « *CodeFile* », qui étendent respectivement les métaclasses **NamedElement**, **Class** et **Artifact**. Pour la modélisation de l'architecture physique, le stéréotype « *HwRegion* », qui étend « **HwResource** » du profil HRM, est défini, de celui-ci dérivent deux sous-stéréotypes : « *HwStaticRegion* » et « *HwReconfigurableRegion* » . Les ports physiques sont modélisés à l'aide du stéréotype « *HwPort* », qui étend « **HwComponent** ». Le modèle final en MARTE est transformé en une description intermédiaire IP-XACT avant de générer les fichiers de spécification Xilinx XPS.

Dans un autre travail, une extension de RecoMARTE est proposée par [Guillet et al., 2014] afin de spécifier un contrôleur de reconfiguration destiné aux SoCs dynamiquement reconfigurables. Pour la vérification des contraintes à l'aide des valeurs NFPs (Non-Functional Properties), une extension des concepts NFP est introduite. Le stéréotype « **NfpType** » est étendu par « *NfpMeasure* », permettant la combinaison et la comparaison des valeurs associées aux types NfpMeasure. En outre, le stéréotype « *Controller* », qui étend « **RtUnit** », est introduit pour définir les mécanismes de contrôle. À partir des

modèles RecoMARTE, différentes transformations sont effectuées pour obtenir une spécification du contrôleur en langage *BZR*, prête pour la synthèse du contrôleur discret. En affet, *BZR* permet de spécifier un système dynamique reconfigurable en fonction d'événements contrôlables ou non, de définir les propriétés d'exécution souhaitées des modules (telles que des contraintes de compositions), et de générer un superviseur assurant ces propriétés par une gestion adaptée des événements contrôlables.

Une méthodologie basé sur l'IDM et utilisant RecoMARTE pour le prototypage rapide des SoCs reconfigurables est présenté dans [Ochoa-Ruiz et al., 2015]. Elle permet l'intégration de la gestion de la persistance et la relocalisation des bitstreams au sein du framework FAMOUS en se basant sur des métadonnées en IP-XACT. Ainsi, à partir de bibliothèques d'IPs et d'une modélisation de la plateforme matérielle en RecoMARTE, une description conforme à IP-XACT est générée automatiquement. Par la suite, un moteur de transformation de modèles traite ces modèles de haut niveau afin de générer automatiquement une description RTL synthétisable.

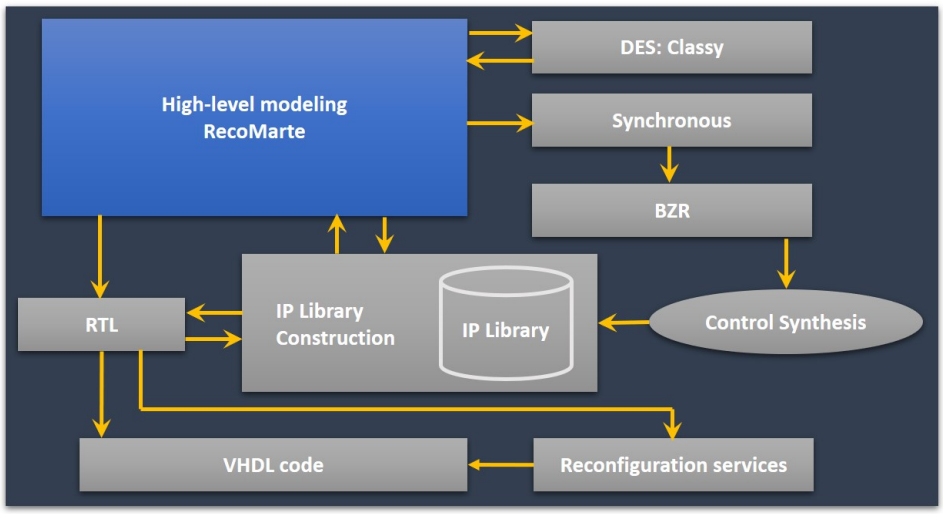



FIGURE 3.5 – *Flot de conception FAMOUS [Cherif, 2013].*

Dans [Fredj et al., 2019; 2020], les auteurs proposent un nouveau package nommé RuntimeReconfiguration étendant MARTE pour modéliser la structure et le comportement des ressources logicielles adaptatives (couche application) des systèmes embarqués temps réel adaptatifs. L'adaptation repose sur la modification des paramètres algorithmiques ou applicatifs pour lesquels chaque modification exige une quantité variable de ressources temporelles et implique différentes propriétés non fonctionnelles. Cette extension permet à la fois d'analyser dynamiquement l'historique des informations et d'évaluer les contraintes temps réel du système. L'extension traite uniquement la partie logicielle du système et touche principalement les quatre sous packages de MARTE : SRM pour décrire la structure du système fondée sur le multitâche logiciel, CommonBehavior et RuntimeContext pour fournir des concepts de modélisation comportementale et de sémantique d'exécution et enfin Timed pour définir la période d'exécution des ressources adaptatives. L'ensemble des extensions proposées est présenté dans la figure 3.6, où les métaclasses ajoutées sont représentées en vert. Conformément aux principes de l'IDM, des transformations de type M2T sont effectuées afin de transposer les propriétés du système, spécifiées dans MARTE, vers le métamodèle de Matlab/Simulink. Cette démarche vise à valider le comportement adaptatif du système ainsi que le respect des contraintes temps réel.

Un flot de conception générique pour les systèmes partiellement et dynamiquement reconfigurables (Generic Design Flow for Dynamic Partially Reconfigurable systems –

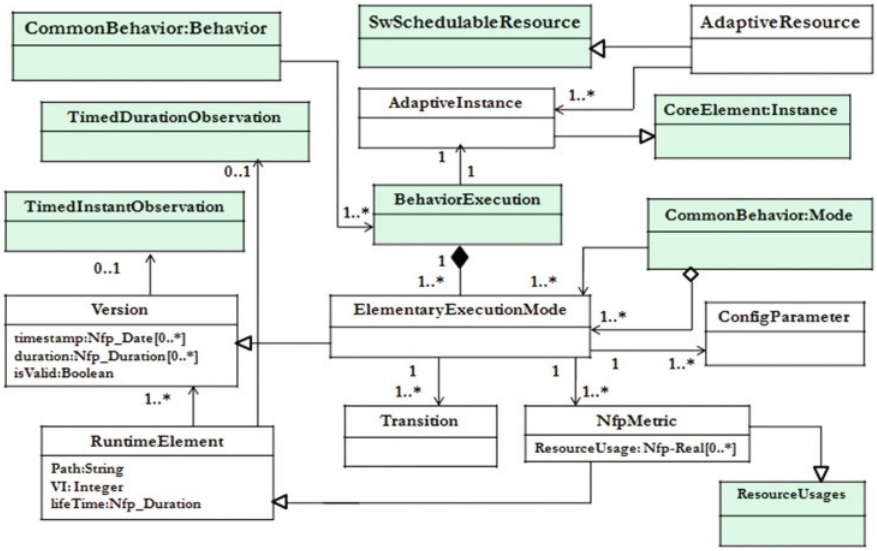



FIGURE 3.6 – *Le package RuntimeReconfiguration [Fredj et al., 2020].*

GDF4DPR) est présenté dans [Dalbouchi et al., 2023]. Une extension restreinte du profil MARTE a été réalisée afin de prendre en charge la DPR. Pour représenter la partie reconfigurable de l'architecture, le profil MARTE a été enrichi par l'ajout du stéréotype « *ReconfigurableHwComponent* », qui hérite du stéréotype « **HwComponent** » défini dans MARTE. Par ailleurs, la modélisation du contrôleur s'appuie sur le stéréotype « *ReconfigurationController* » , conçu comme une extension du stéréotype « **RtUnit** » du profil MARTE. Telle que montré dans la figure 3.7, au niveau du déploiement, les stéréotypes « *IP* » , « *CodeFile* » et « *Implements* » étendent respectivement les métaclasses **Class**, **Artifact** et **Abstraction**, dans le but de : modéliser des IPs, qu'elles soient logicielles (destinées à l'exécution sur processeur) ou matérielles (ciblant une implémentation sur la logique FPGA) ; associer des fichiers de code aux IPs correspondantes ; et établir un lien entre les modules matériels ou logiciels et les IPs d'implémentation. La communication inter-IPs est représentée en modélisant les ports de chaque bloc et en définissant les liens qui les connectent. Pour ce faire, le stéréotype « **FlowPort** » de MARTE est étendu par le stéréotype « *ExtendedPort* » offrant deux types de ports : le port de type *singlePort* utilisé pour spécifier une connexion point à point, et *busInterface* utilisée pour représenter une connexion par bus. Un fichier de conception IP-XACT décrivant l'ensemble de l'architecture est généré à partir du modèle d'architecture déployé, sur la base d'un ensemble de règles de transformation. Ce dernier est transformé par la suite en une conception spécifique, destinée à être importée dans un outil de conception FPGA. La génération du code VHDL/C consiste à traduire les machines à états « **modeBehaviors** » de MARTE en code destiné à la prise de décision en matière de reconfiguration, accompagné du code requis pour lancer le processus de reconfiguration.

Dans [Naija et Ahmed, 2016] , les auteurs proposent une extension du profil MARTE, et plus précisément du sous-profil SAM, afin d'intégrer des mécanismes d'adaptation dans l'ordonnancement des systèmes embarqués temps réel. L'extension touche plusieurs concepts clés du méta-modèle du package GQAM (voir figure 3.8), notamment **WorkloadEvent**, **Step**, **ExecutionHost** et **SchedulableResource**. Lors d'une migration entre processeurs, le temps d'exécution d'une tâche peut varier. Il devient alors pertinent d'attribuer à l'attribut *deadline* du stéréotype « **SaStep** » une multiplicité [0..*], afin de refléter cette variabilité. De la même manière, une tâche peut subir plusieurs interruptions au cours d'une même période. Ainsi, l'attribut *preemptT*, représentant la durée totale de préemption, doit également adopter une multiplicité [0..*] au lieu de

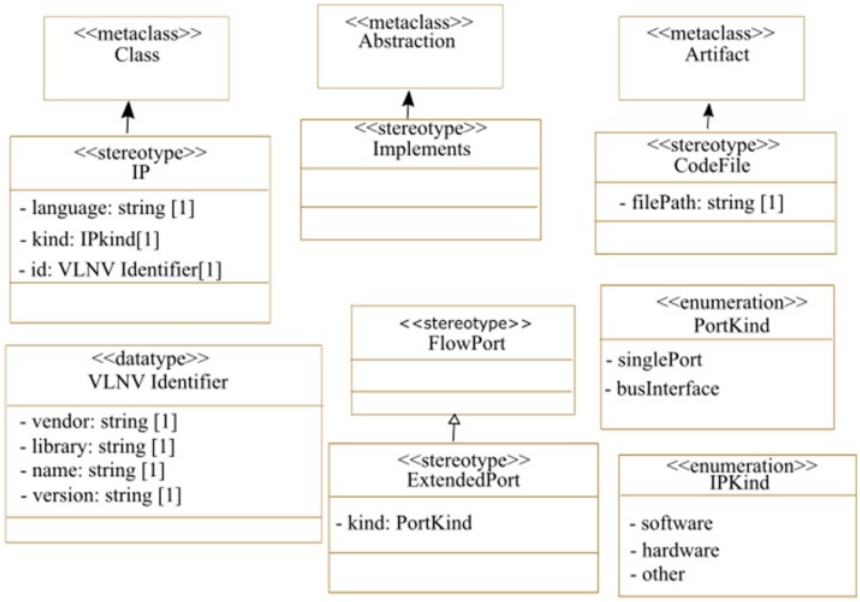



FIGURE 3.7 – *Extension de MARTE pour le déploiement des IPs [Dalbouchi et al., 2023].*

[0..1]. Il en est de même pour l'attribut *readyT*, qui indique le temps écoulé depuis le début de la période ; sa multiplicité doit elle aussi être étendue à [0..∗]. Le stéréotype « **ExecutionHost** » est modifié en y ajoutant l'attribut *unavailability*, afin de permettre la spécification de la durée d'indisponibilité de la ressource. Ces ajustements permettent de spécifier, dès les premières étapes de la conception, les contraintes et événements liés à l'adaptation, facilitant ainsi une analyse précoce de la flexibilité du système face aux changements contextuels ou aux variations de charge.

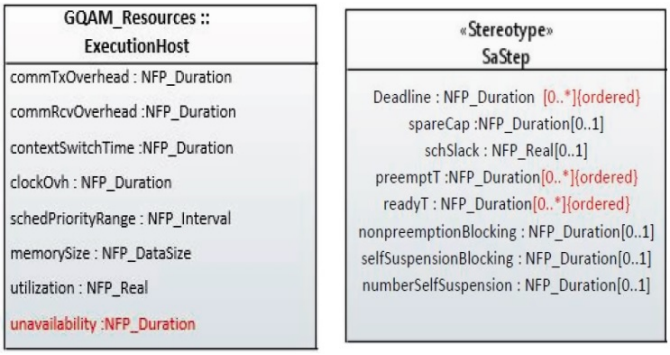



FIGURE 3.8 – *Quelques extensions de MARTE apportées au package GQAM [Naija et Ahmed, 2016].*

### 3.3.3 Discussion

La nécessité d'élever le niveau d'abstraction afin de maîtriser la complexité des SoCs reconfigurables est largement reconnue et partagée par la communauté scientifique. Comparé aux autres langages de description matérielle, SystemC est considéré comme un HDL de plus haut niveau, permettant la modélisation à différents niveaux d'abstraction. Toutefois, il reste un langage de bas niveau en regard d'UML et de ses profils, qui offrent une représentation plus abstraite et orientée système.

Ainsi, les approches exposées dans la section 3.3.1 présentent des limitations liées à un niveau d'abstraction relativement bas et à l'absence de mécanismes d'automatisation

appropriés, compromettant ainsi leur capacité à optimiser les étapes amont du cycle de développement, notamment en termes de modélisation, de vérification précoce et de génération de code.

Si les travaux présentés dans [de la Fuente et al., 2016],[Gamatié et al., 2011] et [Qadri et al., 2010] contribuent à une élévation du niveau d'abstraction, ils demeurent circonscrits à l'utilisation des concepts standards de MARTE et ne proposent aucun support spécifique pour la DPR. Cette absence se traduit par une expressivité restreinte des modèles, ce qui influe directement sur la qualité et la pertinence du code généré. D'autres travaux étendant MARTE aux concepts de la DPR, se focalisent sur des aspects spécifiques de la reconfiguration, tels que la modélisation du contrôle [Trabelsi, 2013] et [Guillet et al., 2014], l'adaptation logicielle [Fredj et al., 2019; 2020] ou la reconfiguration matérielle [Vidal et al., 2011] sans pour autant proposer un support méthodologique exhaustif couvrant l'ensemble du processus de reconfiguration dans MARTE. Bien que RecoMARTE [Cherif, 2013], [Guillet et al., 2014] et [Ochoa-Ruiz et al., 2015] prenne en charge plusieurs propriétés de la DPR, sa définition de la sémantique repose sur un style de codage propre à Xilinx et requiert des compétences de conception avancées. Cette exigence technique peut compromettre le niveau d'abstraction élevé généralement recherché dans les approches de modélisation.

Par ailleurs, malgré le fait que le profil MARTE intègre des capacités d'analyse avancées, notamment en termes d'analyses d'ordonnançabilité et de performance, les travaux existants n'exploitent que peu, voire pas du tout, ces capacités d'analyse, limitant ainsi la portée et la robustesse des solutions proposées pour la gestion de la DPR.

Ainsi, l'utilisation conjointe de UML/MARTE et de SystemC pour la modélisation de la DPR selon une approche IDM permet d'élaborer des modèles de haut niveau d'abstraction indépendants des détails techniques des plateformes cibles. Cette indépendance facilite la séparation des préoccupations et ouvre la voie à la génération automatique du code, via des chaînes de transformations de modèles conformes aux métamodèles définis, réduisant ainsi l'intervention manuelle et optimisant la productivité du processus de développement.

La modélisation des systèmes reconfigurables repose généralement sur une approche orientée composants, en raison de la nature hiérarchique et modulaire des systèmes électroniques. Ce paradigme facilite la structuration des applications et la réutilisation des éléments fonctionnels. Toutefois, bien qu'en accord avec la nature intrinsèquement structurée des systèmes électroniques, le développement par composants seul ne permet pas d'exprimer efficacement toute la sémantique de la DPR. À cet égard, le paradigme des composants orientés services constitue une alternative plus flexible et mieux adaptée. Il offre un cadre expressif pour la gestion de la disponibilité dynamique des services, ce qui en fait un choix pertinent pour modéliser les systèmes dynamiquement reconfigurables (voir la section 2.6.3).

## 3.4 Synthèse

La synthèse des travaux présentés précédemment permet d'établir une vue d'ensemble des différentes approches visant la modélisation haut niveau des SoCs reconfigurables. Comme présenté dans le tableau 3.1, les travaux connexes ont été comparés selon plusieurs critères pertinents, notamment la prise en charge explicite de la DPR au sein du langage de modélisation, le recours à une approche IDM pour la transformation des modèles, le paradigme de conception adopté, le langage cible et le niveau d'abstraction, ainsi que la génération automatique de code et les outils de transformation utilisés. Ces critères ont servi de fondement à la définition des axes de recherche abordés dans cette thèse, en orientant les choix méthodologiques vers une élévation à la fois

du niveau d'abstraction des modèles des systèmes cibles ainsi que du niveau d'expressivité de la DPR dans ses modèles. A la différence des extensions de MARTE proposées dans certains travaux, dont la portée reste limitée, notre adaptation vise une couverture plus étendue et une modélisation plus expressive. En effet, l'extension proposée intègre un plus grand nombre de packages et propose une palette élargie de stéréotypes, permettant de couvrir de manière plus exhaustive les concepts fondamentaux de la DPR. Par ailleurs, l'utilisation de composants orientés services permet de modéliser la dynamicité et la reconfigurabilité du système à un niveau d'abstraction plus élevé, en offrant une flexibilité supérieure à celle du paradigme basé sur les composants. Le recours à SystemC au niveau de modélisation TLM favorise le maintien d'un haut degré d'abstraction, en dissimulant les aspects bas niveau tels que la gestion des signaux ou des horloges, au profit d'une focalisation sur les interactions fonctionnelles entre composants, modélisées sous forme de transactions. Un aspect notable de notre proposition réside dans l'intégration d'une analyse précoce des modèles, visant à renforcer la robustesse du système dès les premières étapes de conception.

TABLE 3.1 – *Synthèse des principaux travaux liés à la modélisation haut niveau des SoCs selon les critères : C1 : Prise en charge de la DPR, C2 : Langage de modélisation, C3 : Packages étendus pour la couverture de la sémantique de la DPR, C4 : Utilisation de l'approche IDM, C5 : Paradigme de conception, C6 : Langage/environnement cible et niveau d'abstraction, C7 : Prise en charge de la génération automatique de code, C8 : Outils de transformation utilisés.*

| Référence | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 |
|---|---|---|---|---|---|---|---|---|
| [Carvalho et al., 2018] | non | UML | - | non | Service | SystemC | oui | ND |
| [Ebeid et al., 2015] | non | UML/MARTE | - | oui | Orienté objet<br>Contraintes temporelles | SystemC/TLM | oui | UML2HIF<br>HIF2SC<br>HIF2VHDL |
| [Leite et Wehrmeister, 2016] | non | UML/MARTE | - | oui | Orienté aspect | VHDL | oui | GenERTiCA |
| [Sotiriou-Xanthopoulos et al., 2013] | oui | - | - | non | Basé processus | SystemC | non | - |
| [Cervero et al., 2013] | oui | - | - | non | Basé composant | SystemC/TLM<br>RTL | non | - |
| [Suvorova et al., 2015] | oui | - | - | non | CDFG | SystemC/TLM | non | - |
| [Duhem et al., 2011]<br>[Duhem et al., 2015] | oui | - | - | non | Orienté objet<br>Basé processus | SystemC/TLM | non | - |
| [de la Fuente et al., 2016] | oui | UML/MARTE | - | oui | Basé composant | SystemC/TLM<br>RTL | oui | Acceleo<br>MTL |
| [Qadri et al., 2010] | oui | UML/MARTE | - | oui | Basé composant | RTL<br>VHDL/C | oui | QVTO<br>Acceleo |
| [Vidal et al., 2011] | oui | UML/MARTE | - | oui | Basé composant | VHDL | oui | ND |
| [Cherif, 2013]<br>[Guillet et al., 2014]<br>[Ochoa-Ruiz et al., 2015] | oui | RecoMARTE | HLAM,GCM<br>HRM, GRM | oui | Basé composant | Xilinx XPS | oui | QVTO<br>Acceleo |
| [Fredj et al., 2019; 2020] | oui | UML/MARTE | SRM<br>CommonBehavior<br>RuntimeContext<br>Timed | oui | Basé composant | Matlab/Simulink | oui | Acceleo |
| [Dalbouchi et al., 2023] | oui | UML/MARTE étendu | HRM, HLAM, GCM, {Class, Abstraction, Artifact} | oui | Basé composant | IP-XACT<br>C/VHDL | oui | ND |
| Le framework proposé [Allem et al., 2016; 2021] | oui | MARTE4DPR<br>MARTE4SCTLM | HLAM, GCM, SRM, HRM, GRM, GQAM | oui | Composant orienté service | SystemC/TLM | oui | ATL<br>Acceleo |

## 3.5 Conclusion

Ce chapitre a présenté un panorama des principaux outils et flots de conception, qu'ils soient d'origine commerciale ou issus du domaine académique, contribuant à la mise en œuvre de systèmes reconfigurables. L'objectif était de mettre en évidence les solutions existantes relatives à la prise en charge de la DPR, à la modélisation ainsi qu'à la génération automatique de code. Il convient également de souligner que dans le domaine de l'électronique, le terme haut niveau est le plus souvent associé à l'utilisation de SystemC. Toutefois, des langages de modélisation situés à un niveau d'abstraction supérieur, tels qu'UML, peuvent être positionnés en amont de SystemC, élevant davantage le niveau d'abstraction.

Dans un second temps, plusieurs approches de modélisation de la DPR à un niveau d'abstraction élevé ont été examinées, en particulier celles s'appuyant sur UML/-MARTE et SystemC. Il en ressort que ces langages ne disposent pas de mécanismes natifs pour la prise en charge de la DPR, ce qui justifie la nécessité de les adapter à cet objectif. Les approches présentées témoignent des efforts déployés visant à intégrer la reconfiguration au sein des processus de conception, tout en promouvant une meilleure abstraction, une séparation claire des préoccupations, une analyse précoce du système ainsi qu'une automatisation optimisée.

Le chapitre suivant présente en détail notre framework de modélisation de haut niveau et d'analyse précoce de la DPR, permettant la génération automatique de code SystemC au niveau transactionnel à partir de spécifications exprimées en UML/MARTE étendu.

# 4

# Le framework proposé

Sommaire

## 4.1 Introduction

Nous introduisons dans ce chapitre notre framework dédié à la modélisation et à l'analyse de haut niveau des systèmes sur puce reconfigurables. L'objectif principal de ce framework est de maîtriser la complexité inhérente à la conception de tels systèmes tout en améliorant la productivité des concepteurs. Nous débutons par une description de l'architecture en couches du framework, suivie de la présentation d'un flot de conception de haut niveau, inspiré du modèle en Y utilisé pour la co-conception matériel/logiciel. Cette introduction est ensuite complétée par la définition des métamodèles représentant les concepts clés du domaine applicatif, marquant ainsi le point de départ du processus de métamodélisation. L'approche adoptée pour la modélisation de l'analyse temporelle, telle que spécifiée par le profil MARTE, est présentée sous deux formes : l'analyse d'ordonnançabilité et l'analyse des performances. Par la suite, nous détaillons les extensions proposées au profil MARTE, à savoir MARTE4DPR et MARTE4SCTLM. Ces extensions constituent les fondations de notre framework en comblant les limitations expressives du profil standard, et permettent ainsi une modélisation plus fine de la DPR ainsi qu'une amélioration de la qualité du code généré. Enfin, nous exposons les mécanismes de transformation de modèles mis en œuvre pour générer automatiquement du code source SystemC/TLM à partir de modèles conformes au profil MARTE étendu, avant de conclure ce chapitre par une synthèse des contributions présentées.

## 4.2 Architecture du framework

En utilisant les langages UML/MARTE et SystemC pour élever le niveau d'abstraction de la modélisation DPR dans un cadre basé sur un modèle de composant orienté services SOCM, nous sommes inévitablement confrontés à divers défis. Le principal défi est de savoir comment modéliser le DPR dans MARTE, d'autant plus que le profil ne présente aucun support complet. Un autre dilemme est que SOCM n'est pas supporté nativement par MARTE, outre le fait qu'il reste difficile de combler le fossé entre les concepts spécifiques au domaine UML/MARTE et SystemC/TLM.

Pour relever ces défis, nous proposons un framework pour la modélisation et l'analyse des SoCs reconfigurables avec une architecture en couches dans laquelle les niveaux d'abstraction sont construits les uns sur les autres jusqu'à atteindre les niveaux techniques, comme le montre la figure 4.1. La mise en œuvre des principes SOCM permettant la prise en charge de la disponibilité dynamique des services est superposée au sommet de l'architecture. MARTE est étendu avec les concepts DPR et SOCM d'une part et avec les concepts SystemC/TLM d'autre part donnant naissance respectivement aux profils MARTE4DPR et MARTE4SCTLM (voir la section 4.5). La couche d'interopérabilité TLM est composée d'interfaces principales, de sockets, de charge utile générique et de protocole de base. La couche inférieure comprend la bibliothèque SystemC fournissant le langage de base, les types de données SystemC, les canaux prédéfinis et les utilitaires.

## 4.3 Flot de conception de haut niveau

L'association des techniques de transformation de modèles en ingénierie dirigée par les modèles et des fondements de MARTE pour le développement basé sur les modèles permet de prendre en charge un flot de conception commun pour les systèmes dynamiquement reconfigurables avec intégration d'outils de modélisation et d'analyse. Un flot de conception de haut niveau est un processus structuré qui permet de développer

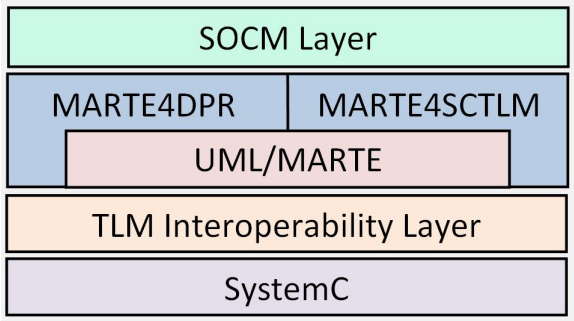



FIGURE 4.1 – *L'architecture en couches du framework.*

et de valider précocement des systèmes complexes tels que des SoCs reconfigurables, en utilisant des abstractions de haut niveau. Ainsi, des modèles abstraits indépendants des plateformes cibles devraient permettre de générer automatiquement du code grâce à des transformations/raffinements successifs de modèles. Un tel flot permet une vérification et une validation plus rigoureuses des modèles avant leur transformation en code. Comme le montre la figure 4.2, nous adoptons une approche de co-conception basée sur le modèle en Y (Y-chart design flow) pour guider le développement et structurer la conception des SoCs reconfigurables. Le modèle en Y repose sur le principe de séparation des préoccupations et se compose de trois branches principales qui convergent progressivement pour aboutir à un système complètement intégré. Les modèles d'application et d'architecture sont développés indépendamment puis combinés dans un modèle d'allocation à divers niveaux d'abstraction. Le modèle d'application se concentre sur les exigences fonctionnelles et les spécifications de haut niveau du système. Le modèle d'architecture décrit les détails techniques de la plateforme cible. A des niveaux d'abstraction plus bas, il spécifie les ressources matérielles du système, comme les processeurs, les mémoires, les bus de communication, et autres composants physiques. A ce niveau, le concepteur doit déterminer la meilleure configuration matérielle pour satisfaire aux exigences des applications, en optimisant les performances, la consommation d'énergie et les coûts. Dépendant de la plateforme technique, un modèle d'allocation inclut les détails d'implémentation à la fois pour le matériel et le logiciel, tels que le partitionnement entre les tâches logicielles et matérielles, le choix des langages de programmation ou de description matérielle, ainsi que la génération de code.

Dans notre framework, la modélisation de la préoccupation DPR de l'application avec MARTE4DPR est basée sur la composition de blocs de composants interactifs offrant une disponibilité dynamique des services. Alors qu'un modèle d'architecture est représenté comme une structure hiérarchique comprenant des ressources reconfigurables fournissant des services pour prendre en charge l'exécution de l'application. Les éléments fonctionnels sont mappés sur des ressources d'architecture au sein d'un modèle d'allocation, qui englobe à la fois des fonctionnalités de distribution spatiale et de planification temporelle. A partir de ce modèle d'allocation qui doit être suffisamment détaillé et précis, le flot propose de traiter deux préoccupations spécifiques, à savoir, l'analyse d'ordonnançabilité et la génération automatique de code. Pour la première préoccupation, le sous-profil SAM de MARTE est appliqué sur le modèle d'allocation donnant lieu à un modèle d'analyse utilisé comme modèle d'entrée pour les outils Cheddar [1] et Marte2Mast [2], ce dernier est employé pour générer un modèle textuel MAST (Modeling and Analysis Suite for Real-Time Applications) [3]. Les outils

1. https ://beru.univ-brest.fr/cheddar/
2. https ://mast.unican.es/umlmast/marte2mast/
3. https ://mast.unican.es/mast.html

d'analyse Cheddar et MAST sont invoqués pour fournir des analyses prédictives de la capacité du système à respecter ses contraintes de temps. Ils aident les concepteurs à prendre des décisions éclairées pour ajuster les paramètres d'ordonnancement, améliorer l'architecture du système, ou détecter les goulets d'étranglement.

Une fois le résultat de l'analyse d'ordonnançabilité jugé satisfaisant et la conception approuvée, le processus de transformation/raffinement des modèles pour la génération de code pourra commencer. Ainsi, selon le modèle en Y appliqué cette fois-ci à un niveau d'abstraction plus bas, un modèle de simulation SystemC/TLM peut être obtenu de deux manières différentes : soit automatiquement par une transformation de modèles soit manuellement par l'application du profil MARTE4SCTLM au modèle de conception obtenu précédemment. Le langage de transformations de modèles ATL est utilisé pour réaliser les transformations nécessaires. Pour le modèle en Y, le modèle de conception précédemment obtenu constitue le modèle d'application, le modèle d'architecture est représenté par un modèle de description du langage SystemC/TLM et le modèle d'allocation est le modèle de simulation SystemC/TLM résultant. À son tour, le modèle résultant est utilisé pour la génération automatique de code SystemC/TLM à l'aide du langage Acceleo pour la transformation des modèles en texte. Enfin, le code SystemC/TLM généré peut être utilisé comme entrée à un outil de synthèse de haut niveau (High Level Synthesis - HLS) transformant le code SystemC/TLM en un circuit logique décrit au niveau RTL généralement en Verilog ou VHDL.

De nombreux outils faisant partie de l'écosystème Eclipse ont été utilisés pour la mise en œuvre des différentes étapes du flot de conception. Cette chaine d'outils comprend un environnement de modélisation, un environnement de développement et des outils d'analyse. L'environnement de modélisation EMT (Eclipse Modeling Tools) permet de créer, manipuler et transformer des modèles grâce, notamment à Papyrus, outil de modélisation et support complet de UML/MARTE ; EMF (Eclipse Modeling Framework) utilisé pour la métamodélisation et les langages ATL et Acceleo pour réaliser les transformations de modèles. Eclipse CDT (C/C++ Development Tooling), un environnement de développement intégré C et C++ est utilisé pour la manipulation du code SystemC/TLM généré. Des analyses temporelles incluant l'analyse de l'ordonnançabilité au pire cas, le calcul des temps de blocage, le calcul des marges de temps et l'attribution optimisée des priorités sont réalisées par les outils Cheddar et MAST.

## 4.4 Définition du domaine métier

L'IDM distingue clairement deux niveaux de processus dans le développement de systèmes : le processus de métamodélisation, qui vise à formaliser un domaine de connaissance en créant un DSML (Langage de Modélisation Spécifique au Domaine) et son environnement, et le processus de modélisation, qui utilise les résultats du processus de métamodélisation pour concevoir des systèmes complexes, tout en tirant parti de l'expertise intégrée dans le DSML et les outils qui l'accompagnent. Avec notre framework, un concepteur doit pouvoir modéliser son système dynamiquement reconfigurable en termes de composants orientés services à l'aide du profil UML/MARTE, dans le but de produire automatiquement du code SystemC/TLM. En suivant une approche combinée, les concepts du domaine et leurs relations sont capturés par la syntaxe abstraite du langage sous forme d'un métamodèle. Grâce au métamodèle, il devient possible de créer, modifier, charger et sauvegarder des modèles qui lui sont conformes. Pour favoriser la réutilisabilité, il est préférable de composer ou d'étendre des métamodèles existants. Typiquement, les fonctionnalités orientées objets des métamodèles sont exploiter. Par exemple, des extensions peuvent être réalisées en ajoutant des métaclasses qui héritent du domaine de base. Des patrons de conception peuvent

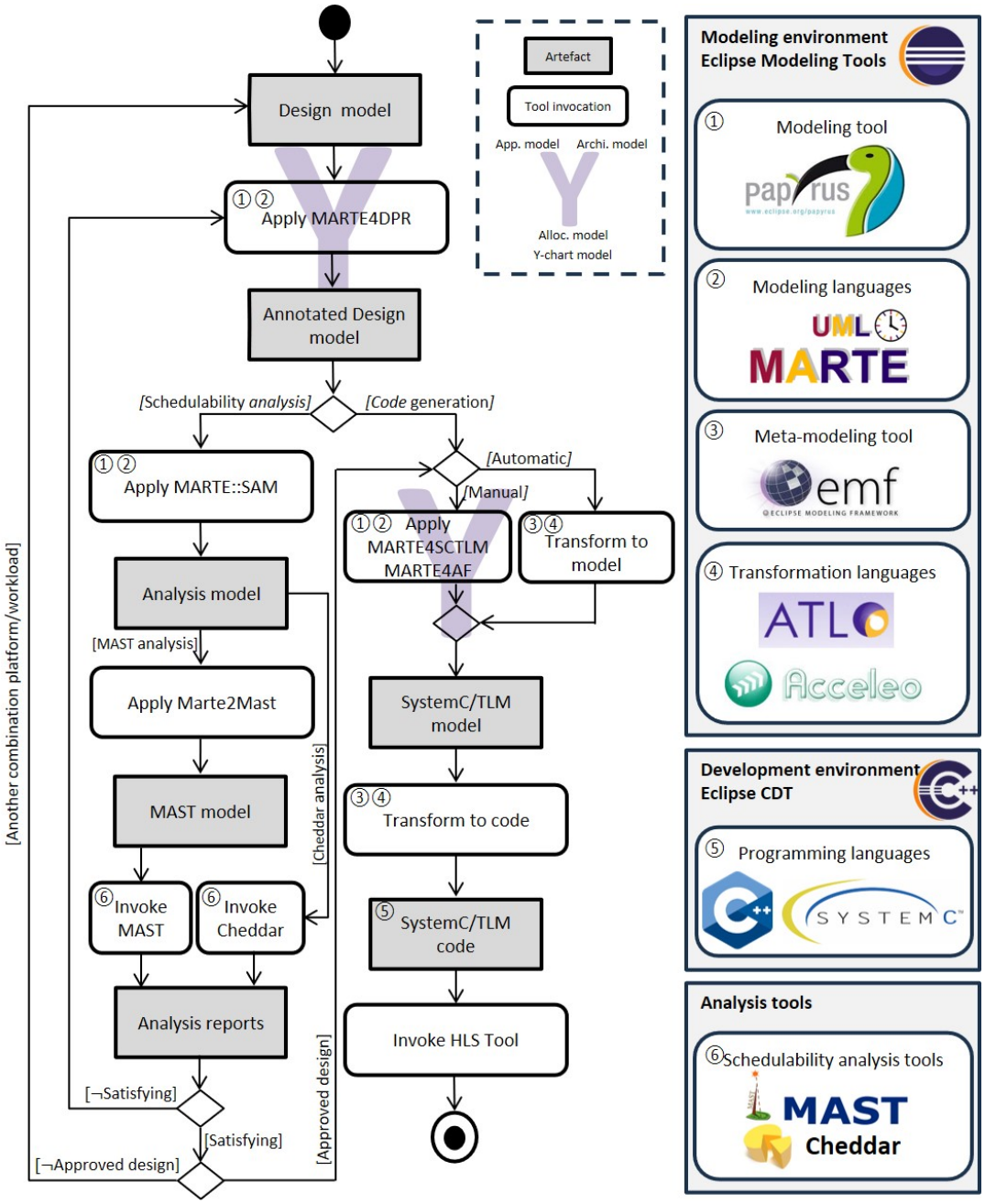



FIGURE 4.2 – *Flot de conception de haut niveau de la DPR et outillage utilisé.*

être également utilisés pour créer des points d'extension de manière élégante. La figure 4.3 représente le métamodèle (syntaxe abstraite) de la plateforme d'exécution virtuelle utilisée dans notre travail. Le métamodèle est créé dans l'environnement Eclipse Modeling Framework (EMF) [4] conformément au langage de métamodélsation Ecore.

### 4.4.1 Modèle du composant reconfigurable

Le profil MARTE adopte principalement une approche basée sur les composants pour la modélisation des systèmes. L'objectif n'est pas de créer un nouveau modèle de composants, mais de réutiliser autant que possible les modèles existants et de faciliter le déploiement optimal d'une architecture MARTE sur un large éventail de plateformes.

4. https ://eclipse.dev/modeling/emf/

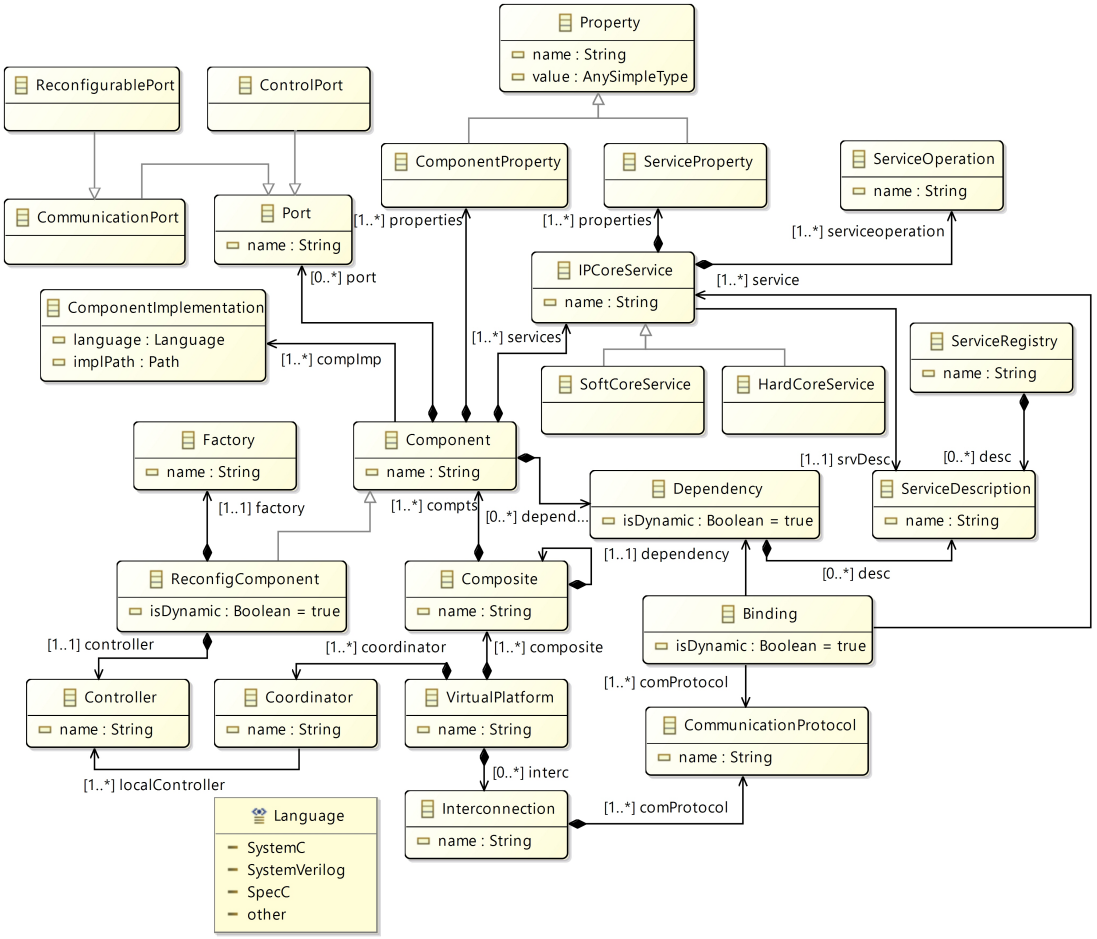



FIGURE 4.3 – *Métamodèle de la plateforme d'exécution virtuelle : Préoccupation de conception*

En tant que profil UML, MARTE s'appuie donc naturellement sur le modèle de composants d'UML2.

Pour assurer une prise en charge efficace de la disponibilité dynamique au sein du composant, nous avons opté pour un modèle de composant orienté service SOCM (Service Oriented Component Model) fusionnant la modularité et la séparation des préoccupations des modèles de composants traditionnels avec le couplage faible, la liaison tardive et la découverte de services du paradigme SOSE. Le couplage faible vise à minimiser les dépendances entre les composants et à atténuer l'impact des modifications apportées à la conception du système. La liaison tardive permet de prendre la bonne décision de déploiement le plus tard possible pour générer une infrastructure communicante au moment de l'exécution. La découverte de services permet à un système de localiser et d'intégrer dynamiquement des services au moment de l'exécution. Cela signifie que les composants peuvent s'adapter aux changements environnementaux ou aux besoins fonctionnels sans nécessiter de redéploiement ou de modification du code.

Conforme à un SOCM, notre composant reconfigurable est composé d'une fabrique et d'un contrôleur, séparant ainsi la logique métier des préoccupations transverses. La logique métier ou logique de domaine est la partie du code qui implémente les règles métier réelles en termes de manipulation des données. Tandis que la fabrique intègre la fonctionnalité du composant, le contrôleur gère entre autres le cycle de vie des instances, les interactions des services, l'adaptation et les liaisons. Un patron de conception (design pattern) particulier a été utilisé pour concevoir le sous-composant fabrique. Il

s'agit de la *fabrique abstraite* (Abstract Factory) qui permet de créer dynamiquement des familles d'objets apparentés et reporter certaines décisions jusqu'à la phase d'exécution.

#### 4.4.1.1 Fabrique abstraite

Les patrons de conception représentent un ensemble de solutions canoniques applicables pour des problèmes classiques. Ces solutions utilisent toutes le paradigme de programmation orientée objet [Gamma et al., 1994]. Les patrons de conception créationnels permettent d'abstraire le processus d'instanciation et de configuration des classes et des objets. Ces patrons mettent en avant deux idées principales : premièrement, ils encapsulent les informations sur les classes concrètes utilisées par le système et deuxièmement, ils dissimulent la manière dont ces classes sont instanciées et assemblées. Le système ne connaît que les interfaces des objets, telles que définies par des classes abstraites. Ainsi, les patterns de création offrent une grande flexibilité sur ce qui est créé, qui le crée, comment et quand. Ils permettent de configurer un système avec des objets produit aux structures et fonctionnalités très variées. Cette configuration peut être statique (définie à la compilation) ou dynamique (au moment de l'exécution). Parmi ces patrons, nous citons : Singleton, Factory Method, Abstract Factory, Builder et Prototype. Bien qu'ils soient complémentaires, certains peuvent être plus bénéfiques que d'autres selon les situations.

Le pattern de conception Abstract Factory aide à résoudre plusieurs problèmes qui mènent souvent au redéveloppement des systèmes, à savoir :

- Créer un objet en spécifiant explicitement une classe : afin d'éviter de spécifier explicitement une classe lors de la création d'un objet ce qui engage à une implémentation particulière plutôt qu'à une interface spécifique, il est préférable de créer les objets de manière indirecte.
- Dépendance aux plateformes matérielle et logicielle : un système qui dépend d'une plateforme logicielle ou matérielle spécifique sera plus difficile à adapter à d'autres plateformes. De plus, il peut être difficile de le maintenir à jour sur sa plateforme d'origine. Il est donc essentiel de concevoir un système de manière à réduire ses dépendances à la plateforme.
- Dépendance aux représentations ou aux implémentations des objets : si les clients ont connaissance de la façon dont un objet est représenté, stocké, localisé ou implémenté, ils pourraient être amenés à être modifiés lorsque l'objet évolue. En dissimulant ces informations aux clients, on prévient la propagation des changements.
- Couplage fort : les classes qui sont fortement couplées sont difficiles à réutiliser de manière isolée, car elles dépendent les unes des autres. Un couplage fort conduit à des systèmes monolithiques, où il est impossible de modifier ou de supprimer une classe sans comprendre et changer de nombreuses autres classes. Le système devient une masse dense qu'il est difficile d'apprendre, de porter et de maintenir. Le couplage faible augmente la probabilité qu'une classe puisse être réutilisée de manière autonome et qu'un système puisse être modifié et étendu plus facilement.

En se basant sur les concepts de fabrique et de produits, le patron Abstract Factory illustre comment créer des familles d'objets produits liés sans avoir à instancier directement des classes. Il est particulièrement approprié lorsque le nombre et les types généraux d'objets produits demeurent constants, tout en présentant des variations au sein de familles de produits spécifiques. Nous faisons le choix entre ces familles en instanciant une fabrique concrète spécifique et en l'utilisant de manière cohérente pour créer les produits par la suite. Il est également possible de remplacer des familles en-

tières de produits en substituant la fabrique concrète par une autre instance différente. L'accent mis par le pattern de la Fabrique Abstraite sur les familles de produits le différencie des autres patterns de création, qui se concentrent uniquement sur un seul type d'objet produit.

Comme le montre la figure 4.4, le métamodèle du patron Abstract Factory définit les concepts de base suivants :

— AbstractFactory : fournit une interface pour les opérations qui créent des objets produits abstraits (CreationMethod). AbstractFactory délègue la création des objets produits à la classe ConcreteFactory.
— ConcreteFactory : implémente les opérations pour créer des objets produits concrets (FactoryMethod). En général, une seule instance de la classe ConcreteFactory est créée à l'exécution. Cette fabrique concrète produit des objets avec une implémentation particulière. Pour générer différents objets produits, les clients doivent utiliser une fabrique concrète différente.
— AbstractProduct : définit une interface pour un type d'objet produit.
— ConcreteProduct : définit un objet produit à créer par la fabrique concrète correspondante et implémente l'interface AbstractProduct.
— Client : utilise uniquement les interfaces déclarées par les classes AbstractFactory et AbstractProduct et qui correspond au concept Factory dans notre domaine métier.

Dans notre approche de conception des systèmes dynamiquement reconfigurables, l'Abstract Factory fournit une structure élégante pour la création de services qui peuvent s'adapter à divers contextes durant l'exécution, tout en maintenant la cohérence et la flexibilité du système. C'est un outil puissant pour développer des systèmes réactifs et modulaires, surtout lorsque l'adaptabilité est une exigence clé.

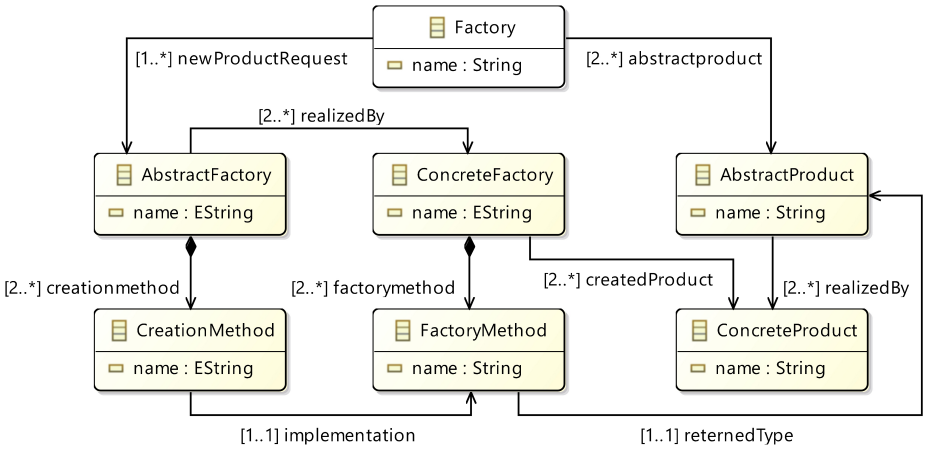



FIGURE 4.4 – *Métamodèle de l'Abstract Factory.*

#### 4.4.1.2 Modèle du contrôle

Pour assurer l'autonomie locale des composants, limitant ainsi la dépendance à un point de contrôle unique, tout en favorisant une coordination globale permettant la collaboration entre les contrôleurs locaux et allégeant la charge du coordinateur central, nous proposons un modèle de contrôle semi-distribué pour lequel la logique de reconfiguration peut être appliquée via une boucle de contrôle MAPE-K. Tel que montré dans la figure 4.5, pour un modèle semi-distribué, plusieurs instances d'une boucle

MAPE-K décentralisée sont réparties sur différents nœuds (contrôleurs locaux), chacun d'eux étant capable de gérer son propre processus de reconfiguration tout en collaborant avec une instance associée à un nœud central pour assurer la cohérence globale du contrôle. La connaissance agit de manière transversale sur chaque étape du contrôle, jouant le rôle de référentiel essentiel pour guider et optimiser les différentes étapes du modèle. Au monitoring, la connaissance permet de définir les métriques à observer, les seuils critiques et les fréquences de collecte. A l'analyse, les modèles, règles ou algorithmes stockés dans la base de connaissances permettent d'identifier des tendances et de préconiser des reconfigurations. Les règles de décision, stratégies ou modèles de planification présents dans la base de connaissances permettent de concevoir des actions de reconfiguration, en effet, elle peut inclure des scénarios ou des plans d'action déjà testés pour répondre à des situations similaires. La connaissance inclut également des protocoles d'exécution, des configurations de paramètres et des modèles pour ajuster les composants du système à l'exécution.

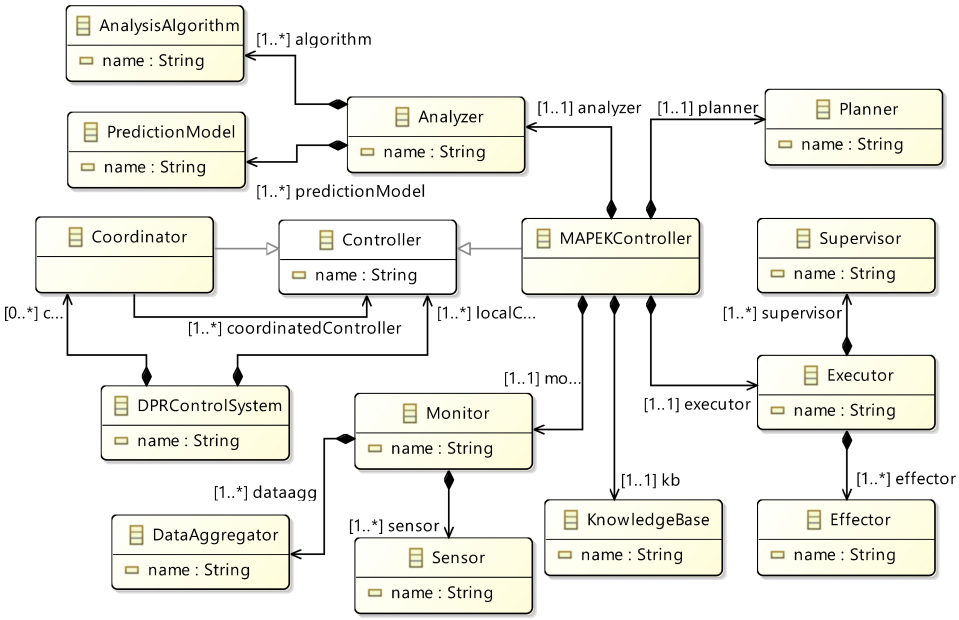



FIGURE 4.5 – *Métamodèle du contrôle de la DPR.*

Le protocole de contrôle local de la DPR selon la boucle MAPE-K est illustré dans la figure 4.6. L'étape du monitoring joue un rôle fondamental dans le contrôle de la DPR, elle permet de collecter les données (métriques systèmes : performances, utilisation des ressources, etc.) nécessaires pour suivre l'état du système et d'identifier les situations exigeant une reconfiguration. La collecte de données se fait par le biais de capteurs matériels, de moniteurs logiciels fournissant des outils pour la surveillance des activités logicielle et matérielles (par exemple des logs systèmes), des interfaces matérielles pour accéder directement à l'état d'un composant, etc. La mise à jour de la base de connaissances par les données collectées et filtrées permet d'enrichir le référentiel du système. Le passage du monitoring à l'analyse peut être déclenché par une détection d'un écart (par exemple un seuil dépassé : latence moyenne > 30 ms pour trois cycles d'horloge consécutifs), une notification d'un événement externe (par exemple une baisse de la qualité du service entrainant un besoin de reconfiguration) ou bien suite à une vérification des conditions formulées dans des règles/politiques fournies par l'utilisateur ou encore par un modèle prédictif pour identifier quand une analyse est nécessaire. L'étape d'analyse permet d'interpréter les données collectées pendant la surveillance

pour déterminer si une reconfiguration du système est nécessaire. Si la réponse est positive, un plan de reconfiguration est élaboré selon le rapport d'analyse. Le plan consiste en une séquence d'actions sélectionnées, évaluées puis ordonnancées pour atteindre un certain nombre d'objectifs. Les conditions de succès pour chaque action (par exemple : la vérification de la stabilité du système après redistribution des ressources ou d'activation/désactivation de modules) permettent de valider ou pas le plan d'exécution. En effet, un plan peut échoué pour plusieurs raisons, notamment : le non respect des contraintes temps réel du système, les préconditions nécessaires pour exécuter le plan ne sont pas vérifiées, une analyse prédictive montre que l'exécution du plan pourrait entrainer une dégradation des performances globale. Une fois le plan validé, son exécution permet d'apporter les modifications nécessaires à la reconfiguration du système et d'établir un état post-reconfiguration (informations sur les performances actuelles et utilisation des ressources). Enfin, le cycle itératif de la boucle MAPE-K traduit par le retour à l'étape du monitoring garantit que le contrôleur maintient une vision globale de l'état du système et peut réagir dynamiquement à toute potentielle reconfiguration.

Le coordinateur assure une fonction essentielle en tant qu'entité pivot, veillant à la gestion globale des processus et à la cohérence des décisions prises par les contrôleurs locaux. En effet, il gère la communication et la coordination entre les contrôleurs locaux et consolide les données et analyses locales pour avoir une vue d'ensemble du système. Il permet également de résoudre les conflits et garantir la cohérence entre les plans et actions des contrôleurs locaux tout en s'assurant que les actions exécutées localement contribuent aux objectifs stratégiques du système.

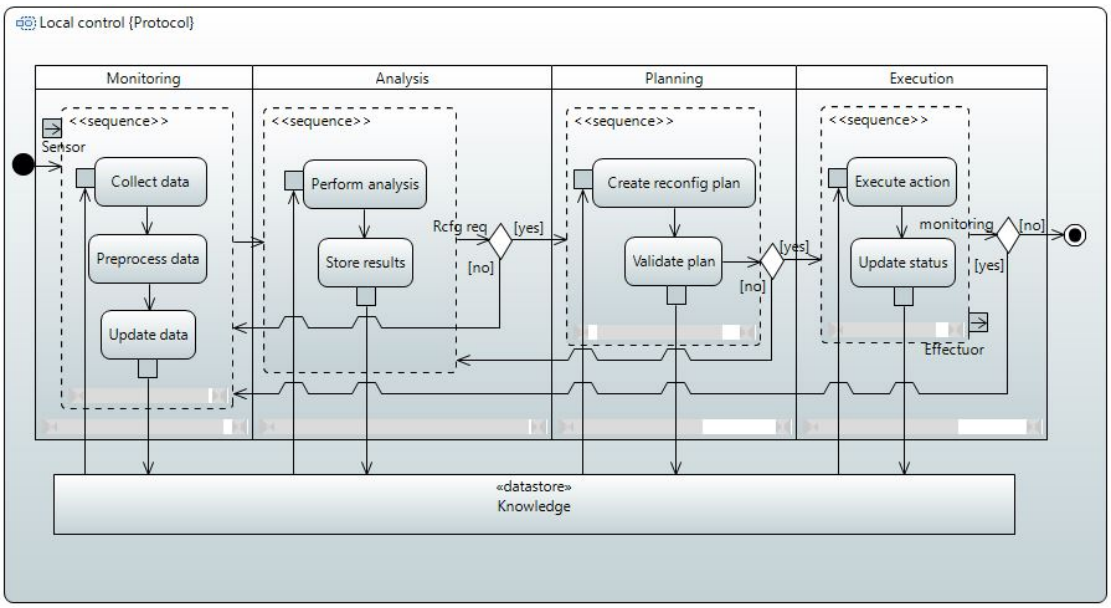



FIGURE 4.6 – *Protocole de contrôle local selon la boucle MAPE-K.*

#### 4.4.1.3 Modèle de communication

Comme UML est un langage orienté objet, les composants d'un système communiquent principalement par des appels d'opérations qui sont généralement synchrones ou asynchrones, ou par signaux, s'inscrivant dans une architecture couramment désignée sous le nom de modèle client-serveur. Dans ce cadre, le sous-profil GCM (Generic Component Model) de MARTE propose des extensions (réduites au strict minimum) pour permettre la communication par échange de données entre composants, tout en offrant un support pour la modélisation des interfaces fournies et requises dans une

architecture de communication client-serveur. De plus, GCM permet de spécifier avec précision le comportement au niveau des ports d'un composant lors de la réception d'un flux d'informations, qu'il s'agisse de flux de données, de signaux ou d'appels d'opérations. MARTE introduit le stéréotype « ClientServerPort » pour les ports UML, ainsi que les propriétés « provInterface » et « reqInterface », qui permettent de spécifier explicitement les interfaces fournies et requises. Pour intégrer une infrastructure de communication hautement flexible, capable de répondre aux exigences de la DPR, nous avons introduit le concept de port reconfigurable au travers une extension du stéréotype « ClientServerPort » (voir la sous-section 4.5.1.2). Nous avons naturellement choisi d'apporter les extensions nécessaires à ce modèle en raison de sa grande similarité avec le modèle de communication TLM visé. En effet, les concepts de client, serveur, appel synchrone , appel asynchrone et message trouvent leur pendant TLM dans les concepts d'initiateur, cible, transport bloquant, transport non bloquant et transaction.

En complément du modèle de communication client-serveur, MARTE permet de modéliser des architectures de composants échangeant des données. Il enrichit le concept de port d'UML en introduisant le stéréotype « FlowPort », représentant un port de données selon deux forme : active (event-triggered ou push-based) et passive (time-triggered ou pull-based).

### 4.4.2 Pattern d'interaction

Pour répondre à de nouvelles exigences, les fonctionnalités peuvent être découvertes, remplacées ou intégrées de manière dynamique dans le système, de la même manière que l'échange d'un module reconfigurable avec un autre. Le dynamisme inhérent à l'orientation service est intrinsèquement compatible avec la capacité de reconfigurer dynamiquement les modules matériels présentés par les SoCs dynamiquement reconfigurables.

Suivant ces principes, dans le framework proposé, une configuration sera composée dynamiquement de modules réutilisables fournissant des services via des liaisons dynamiques. Par conséquent, la reconfiguration dynamique sera perçue comme le résultat de l'arrivée ou du départ des services au moment de l'exécution selon le cycle publier-trouver-lier.

Pour développer des systèmes reconfigurables dynamiquement en minimisant les coûts, les ressources matérielles doivent être allouées uniquement lorsque cela est nécessaire et les services sont donc chargés paresseusement.

Comme le montre la figure 4.7 , après l'intégration IP, le fournisseur annonce l'arrivée du service en publiant sa description dans le registre de services via lequel il est rendu dynamiquement détectable. Un consommateur de services peut rechercher des services en interrogeant le registre de services ; il est également responsable de la sélection et du tri des services disponibles pour en extraire le meilleur. La liaison entre le consommateur de service et le fournisseur au niveau TLM peut être effectuée en utilisant les sockets initiateur et cible via un composant d'interconnexion. Un consommateur de services est informé respectivement de l'arrivée et de la suppression de services nouveaux et existants.

### 4.4.3 Modélisation de l'analyse temporelle

Les modèles d'analyse sont conçus pour être intégrés aux modèles de conception existants, ou au moins définis dans une vue séparée avec un lien clair vers les vues de conception. Dans MARTE, le sous-profil GQAM (Generic Quantitative Analysis Modeling) avec ses deux spécialisation SAM (Schedulability Analysis Modeling) et PAM (Performance Analysis Modeling) permettent d'annoter les modèles de conception avec

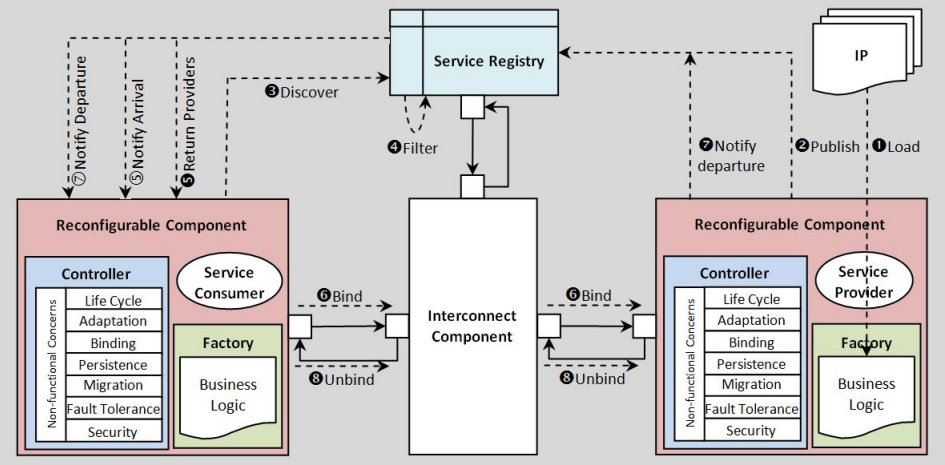



FIGURE 4.7 – *Le pattern d'interaction SOCM pour la plateforme d'exécution virtuelle.*

des informations quantitatives, facilitant ainsi des analyses formelles statiques réalisées directement sur les modèles, sans recourir à une simulation. Ces analyses permettent de détecter très tôt dans le cycle de développement des problèmes potentiels, contribuant ainsi à réduire les risques et les coûts associés au développement.

Les techniques d'analyse quantitative permettent de calculer les valeurs des propriétés non fonctionnelles NFPs (Non Functional Properties) de sortie, comme les temps de réponse, les dépassements d'échéances, les taux d'utilisation des ressources ou les tailles de files d'attente, etc., à partir de données fournies sous forme de NFPs d'entrée telles que les taux de requêtes ou de déclenchements, les exigences d'exécution, les échéances et les objectifs de qualité de service, etc. L'ensemble des NFPs utilisées pour l'analyse est présenté dans la clause 15 de la spécification de MARTE [Obj, 2023b].

Une NFP peut servir d'entrée ou de sortie selon le contexte. Par exemple, la latence dans le pire des cas peut être une sortie lors d'une analyse du temps d'exécution dans le pire des cas (Worst-Case Execution Time - WCET) ou une entrée dans une analyse d'ordonnançabilité.

Pour réaliser une analyse temporelle, un modèle UML doit préciser les services au niveau du système, la fréquence des requêtes, ainsi que les conditions d'exécution, également appelées l'environnement du système.

Le package GQAM est structuré autour du concept de contexte d'analyse **AnalysisContext**. Un exemple courant de contexte d'analyse est un diagramme de séquence représentant un scénario ou un ensemble de scénarios, nécessitant une analyse des caractéristiques temporelles ou de performance. Le contexte d'analyse comprend deux parties traitant des préoccupations distinctes :

— **WorkloadBehavior** : contient un ensemble d'opérations interconnectées de bout en bout, au niveau système, chacune avec un comportement défini, déclenché par un ensemble d'événements liés au **workload**. Le comportement en réponse à un événement déclencheur est décrit par un **BehaviorScenario**, composé de sous-opérations appelées **Steps**, dont chacune peut être raffinée comme un autre **BehaviorScenario**. Un **BehaviorScenario** capture toute description de comportement au niveau système ou toute opération en UML, en y associant l'utilisation des ressources (voir figure A.3). Les **BehaviorScenarios** sont largement employés dans l'analyse temporelle, souvent sous des formes similaires. Dans l'analyse d'ordonnançabilité, ils sont désignés sous le terme séquences de tâches, et ils sont également utilisés pour élaborer des modèles de performance.

— **ResourcesPlatform** : constitue un conteneur logique regroupant les ressources

utilisées par le comportement au niveau système décrit dans le modèle précédent. Dans une perspective d'analyse, quatre types de ressources sont particulièrement importants (voir figure A.4). **ExecutionHost** : un processeur ou un dispositif capable d'exécuter les opérations définies dans le modèle, jouant le rôle d'hôte pour les processus et les Steps qui s'exécutent dessus. **CommunicationsHost** : des connexions matérielles entre dispositifs, servant d'hôte pour le transfert des messages. **SchedulableResource** : un service ordonnançable, tel qu'un processus ou un pool de threads, représentant une ressource logicielle gérée par le système d'exploitation. Enfin **CommunicationChannel** : une couche logicielle ou un protocole permettant le transport des messages. Toutes les ressources disposent d'une politique d'ordonnancement, d'une multiplicité (quantité d'unités disponibles) et offrent des services. L'utilisation des ressources par le logiciel peut s'étendre à l'ensemble d'un **BehaviorScenario** ou se limiter à certaines **Steps**, une **Step** s'exécute sur un processeur qui lui sert d'hôte.

La séparation d'un modèle de contexte d'analyse en ces deux aspects permet aux modélisateurs IDM (et MDA en particulier) de distinguer clairement les modèles indépendants de la plateforme (annotés avec des éléments **WorkloadBehavior**) des modèles décrivant la plateforme (annotations **ResourcesPlatform**).

#### 4.4.3.1 Analyse d'ordonnançabilité basée sur les modèles

Tel que spécifier dans MARTE, les outils couramment utilisés pour ce type d'analyse de modèles proposent deux fonctionnalités principales. La première vise à évaluer l'ordonnançabilité d'un système ou d'une partie spécifique d'un logiciel, c'est-à-dire sa capacité à respecter des contraintes temporelles définies, telles que des échéances ou des taux d'échec. Ces outils identifient généralement les entités qui sont ordonnançables et celles qui ne le sont pas. La seconde, l'analyse de sensibilité, permet de déterminer comment améliorer le système. Cela peut inclure des recommandations pour rendre une entité ordonnançable ou pour équilibrer davantage l'utilisation des ressources du système. Un concepteur de système explore souvent plusieurs configurations en modifiant les paramètres de chaque scénario ou en étudiant la variabilité liée aux allocations de ressources et leur déploiement sur différentes plateformes matérielles et logicielles.

L'analyse d'ordonnançabilité peut être utilisée à différentes étapes. Une analyse précoce d'un modèle de conception aide les développeurs à détecter des architectures temps réel potentiellement irréalisables et à éviter des erreurs de conception coûteuses, en particulier celles liées au comportement temporel. En revanche, une analyse ultérieure d'un système déjà implémenté permet aux analystes d'identifier, avec des informations quantitatives plus précises sur le système, des défauts liés au temps ou d'évaluer l'impact de possibles migrations ou modifications de la plateforme sur les stratégies d'ordonnancement. Les politiques d'ordonnancement dans MARTE se divisent en deux grandes catégories, chacune associée à une méthode d'analyse spécifique. La première catégorie est statique : les décisions paramétriques relatives à l'importance de l'ordonnancement sont déterminées à l'avance, sur la base d'une connaissance préalable des contextes et des scénarios d'exécution possibles. Rate Monotonic Analysis (RMA) et Deadline Monotonic Analysis (DMA) sont des exemples de méthodes d'analyse d'ordonnançabilité statique. La seconde catégorie est dynamique et s'appuie sur des décisions prises en temps réel, en fonction des informations disponibles dans le contexte d'exécution en cours. Earliest Deadline First (EDF) est un exemple concret de méthode d'analyse d'ordonnançabilité dynamique [Obj, 2023b].

Le sous-profil SAM spécialise GQAM en une collection de concepts de modélisation destinés à l'analyse d'ordonnançabilité, ainsi qu'en un ensemble de NFPs asso-

ciées à ces concepts fondamentaux. Le stéréotype « **SaAnalysisContext** » spécialisant « **GaAnalysisContext** » est un type de contexte d'analyse doté d'attributs supplémentaires. L'attribut **isSchedulable** indique si toutes les contraintes temporelles définies pour le contexte d'analyse sont respectées. L'attribut **optimalityCriterion** représente un critère global utilisé pour déterminer un ordonnancement pour le contexte analysé (par exemple, respecter toutes les échéances strictes, minimiser le nombre d'échéances manquées, réduire le retard moyen ou maximiser le débit). Pour une analyse d'ordonnançabilité, il est nécessaire de définir au minimum une plateforme de calcul et un comportement de workload.

a. **Spécification des workloads :** Pour effectuer une analyse d'ordonnançabilité sur un modèle d'application, il est nécessaire de modéliser explicitement l'ensemble des comportements de cette application qui illustrent son mode d'utilisation. Ces comportements sont collectivement désignés sous le terme de workload. Un modèle spécifique de **WorkloadBehavior** est redéfini dans SAM sous forme d'un flux de traitement de bout en bout **EndToEndFlow** représentant le workload analysé. Ainsi, le stéréotype « **SaEndToEndFlow** » spécialise le stéréotype « **GaWorkloadBehavior** » de GQAM en y ajoutant différentes propriétés spécifiques à l'analyse d'ordonnançabilité telles que :
   — isSched : une propriété booléenne qui indique le résultat de l'analyse visant à déterminer si le flux satisfait les exigences d'ordonnançabilité en temps réel imposées. Un système de tâches est qualifié d'ordonnançable lorsque le respect de toutes ses échéances (deadlines) est assuré.
   — schSlak : la variation dans le temps d'exécution réel d'une étape dans le flux qui peut être tolérée sans compromettre l'ordonnaçabilité.
   — end2EndT : le temps de réponse total du flux mesuré à partir de l'instant où il est déclenché.
   — end2EndD : l'échéance requise pour le flux mesurée à partir de l'instant où il est déclenché et des contraintes temps réel (durées estimées ou requises).
   — timing : ensemble d'exigences ou de prédictions temporelles qui contraignent des fragments locaux ou le flux d'exécution global de bout en bout.

   Les deux stéréotypes généraux « **GaStep** » et « **GaCommStep** » sont spécialisés en « **SaStep** » et « **SaCommStep** » respectivement afin d'annoter les étapes d'un flux avec les propriétés suivantes :
   — deadline : le temps de calcul maximum de l'étape.
   — spareCap : un temps de calcul supplémentaire possible (capacité de réserve) par rapport au temps spécifié dans l'attribut deadline.
   — schSlack : une marge positive ou négative selon laquelle le temps d'exécution de l'étape peut varier sans compromettre l'ordonnançabilité.
   — preemptT : la durée pendant laquelle l'étape est préemptée en faveur d'une étape plus prioritaire.
   — readyT : le délai entre le moment où une entité est éligible à l'exécution et le début réel de l'exécution.
   — nonpreemptionBlocking : la durée maximale qu'une étape peut tolérer en attendant que l'exécution d'une étape de priorité inférieure se termine.
   — selfSuspensionBlocking : la durée maximale pendant laquelle une étape peut rester suspendue après s'être elle-même suspendue.
   — numberSelfSuspensions : le nombre maximum de fois qu'une étape peut s'autosuspendre.

b. **Spécification des plateformes :** La conception de la plupart des systèmes temps réel (reconfigurables notamment) dépend fortement des caractéristiques de la plateforme sous-jacente. Naturellement, l'analyse d'ordonnançabilité doit prendre en

compte de manière appropriée l'influence des plateformes sur les caractéristiques temporelles du système. Par conséquent, SAM propose de spécialiser les stéréotypes suivants dédiés à la modélisation des plateformes, tels que « **GRM : : MutualExclusionResource** », « **GQAM : : GaExecHost** » et « **GQAM : : GaCommHost** » en « **SaSharedResource** » , « **SaExecHost** » et « **SaCommHost** » respectivement. Un hôte d'exécution est incarné par un processeur ou un autre dispositif qui exécute des étapes fonctionnelles, le stéréotype « **SaExecHost** » ajoute des métriques d'ordonnançabilité, des surcharges liées aux interruptions et l'utilisation de la capacité de traitement pour l'ordonnancement. Dans un hôte de communication (ex. réseaux ou bus), l'élément de ressource ordonnançable associé est un canal de communication, qui peut être caractérisé par des paramètres d'ordonnancement concrets (comme la taille des paquets). Les hôtes d'exécution possèdent des ressources partagées, telles que des périphériques d'entrée/sortie, des canaux DMA, des sections critiques ou des adaptateurs réseau. Les ressources partagées sont allouées dynamiquement aux ressources ordonnançables au moyen d'une politique d'accès.

#### 4.4.3.2 Analyse des performances basée sur les modèles

La performance constitue une préoccupation essentielle pour de nombreux systèmes logiciels/matériels, notamment les systèmes en temps réel, qui imposent des exigences strictes en matière de réponses rapides aux entrées provenant de l'environnement. La performance des systèmes dépend de divers facteurs, tels que les spécifications de la plateforme sous-jacente, la conception de l'application, ainsi que des éléments externes comme le workload imposé. Un aspect qui rend la prédiction des performances particulièrement complexe est le fait qu'une plateforme peut simultanément prendre en charge plusieurs tâches concurrentes ou même plusieurs applications, partageant toutes les mêmes ressources, souvent en compétition pour ces dernières. Cette complexité augmente si le workload du système est dynamique et varie au fil du temps, comme c'est le cas pour les systèmes dynamiquement reconfigurables.

Les mesures de performance, issues de l'analyse, reposent sur des statistiques, telles que le débit moyen, le délai, ou encore la probabilité de dépasser un délai de réponse prédéfini. De même, les paramètres d'entrée de l'analyse peuvent présenter un caractère probabiliste, comme un processus d'arrivée aléatoire ou une durée d'exécution variable pour une trame multimédia. Les principales mesures de performance à évaluer sont les suivantes : la valeur moyenne du temps de réponse, le débit moyen, la capacité maximale de débit pour une valeur moyenne donnée du temps de réponse et l'utilisation moyenne d'une ressource [Selić et Gérard, 2014]. Ces mesures statistiques et stochastiques permettent des analyses basées sur la théorie des files d'attentes et ses extensions.

Un contexte de performance définit un ou plusieurs **BehaviorScenarios** permettant d'analyser différentes situations dynamiques associées à un ensemble spécifique de ressources. le stéréotype « **GaAnalysisContext** » est appliqué à un diagramme UML et définit les paramètres globaux de l'analyse via l'attribut **contextParams**.

a. **Spécification des workloads :** Dans le cadre d'une analyse des performances, un contexte peut contenir plusieurs workloads, représentant diverses sources de requêtes ou d'initiations d'opérations. Chaque workload dispose de son propre mécanisme d'initiation des requêtes, d'une intensité de charge spécifique et de ses propres exigences en termes de qualité de service. Le **BehaviorScenario** constitue l'unité de description du comportement et correspond à un diagramme de comportement (interaction, activité ou machine à états). Il représente une séquence d'actions, appelées **Steps**, reliées par des relations prédécesseur-successeur qui peuvent inclure

des bifurcations, des jonctions et des boucles. Chaque **Step** précise les besoins en ressources du système, tant pour son exécution sur le processeur hôte que pour d'autres ressources. « **PaStep** » est une spécialisation du stéréotype « **GaStep** », il s'applique généralement à une action UML ou à un message qui invoque un comportement. Il définit une unité de comportement (une étape dans un scénario) nécessitant généralement certaines ressources. Sans un scénario de raffinement, **PaStep** représente une étape d'exécution séquentielle de base sur un processeur hôte ; avec un scénario de raffinement, elle constitue une unité de comportement plus large. Les propriétés suivantes caractérisent une **PaStep** primitive :

— noSync : booléen identifiant une branche asynchrone d'une bifurcation.
— extOpDemands : un ensemble d'identifiants représentant les opérations des services externes sollicitées par cette étape, dans un format compatible avec l'environnement de performance.
— extOpCount : le nombre de requêtes effectuées pour chaque opération externe lors d'une exécution de l'étape, dans le même ordre que les demandes.
— behavDemands : un ensemble de scénarios définissant les opérations invoquées par cette étape.
— behavCount : le nombre de requêtes effectuées pour exécuter chaque opération de scénario lors d'une exécution de l'étape, dans le même ordre que les demandes.

Une **PaStep** de niveau supérieur comporte un comportement imbriqué décrit par un **GaScenario** et défini comme l'opérande d'un fragment combiné UML ou d'une activité structurée UML. « **PaCommStep** » est une spécialisation des stéréotypes « **GaCommStep** » et « **PaStep** » et représente une activité de communication.

b. **Spécification des plateformes :** L'analyse des performances repose sur la manière dont le comportement du système exploite ses ressources. Les ressources clés comprennent des moteurs d'exécution matériels **GaExecHost** et **GaCommHost**, des threads de processus concurrents **SwScheduledResources** et des ressources logique définies par le logiciel **PaLogicalResources**. Une ressource logique peut être toute entité à laquelle le logiciel doit accéder et pouvant entraîner une attente du programme à un moment donné telle qu'un sémaphore, un verrou, un tampon, un bloc de mémoire, etc. Dans les spécifications comportementales, les processus peuvent être représentés comme des lignes de vie (diagramme de séquence) ou des couloirs (diagramme d'activités). Pour cela, le concept spécifique de **RunTimeObjectInstance** a été introduit afin de jouer le rôle d'alias pour une ressource de processus.

#### 4.4.3.3 Modèle de coût pour la DPR

Les dispositifs à grain fin, tels que les FPGA, offrent une personnalisation au niveau du bit, garantissant une grande flexibilité. Cependant, ils nécessitent des bitstreams volumineux pour leur configuration, contrairement aux dispositifs à gros grain. Cette caractéristique engendre un temps de reconfiguration significatif, qui reste non négligeable, même pour les dispositifs prenant en charge la reconfiguration partielle. Le transfert des bitstreams de la mémoire externe vers la mémoire interne constitue la principale source de surcharge liée à la DPR.

Avec l'utilisation croissante des plateformes FPGA en tant que produits finaux plutôt que de simples outils de prototypage rapide, la DPR prend une importance majeure. Il devient ainsi pertinent d'anticiper et de calculer efficacement le surcoût qu'elle engendre.

Le temps de reconfiguration a souvent été associé au port de configuration du FPGA, bien que seul un contrôleur de reconfiguration performant puisse permettre d'atteindre un débit de reconfiguration proche de la bande passante de l'ICAP. Cependant, cette

performance reste rarement atteinte en pratique, en raison des limitations imposées par les composants du système ou des choix de conception et des exigences spécifiques de l'application.

Pour les systèmes réels, une approche plus holistique est nécessaire, prenant en compte l'ensemble des composants du système qui contribuent au surcoût total de reconfiguration, tels que : la mémoire interne et externe, le port de configuration interne, le contrôleur de reconfiguration, la mémoire de configuration du FPGA, et les moyens de connexion [Papadimitriou et al., 2011].

Un modèle de coût permettant d'estimer à l'avance le temps de reconfiguration attendu peut faciliter l'évaluation des performances. Idéalement, ce modèle devrait être exploitable dès les premières étapes du processus de conception, afin de dissuader le concepteur de s'engager dans un processus complexe de conception de la DPR si cela s'avère inutile, ou d'ajuster rapidement les paramètres de configuration du système.

Une nouvelle méthode pour calculer le temps de la DPR, le temps de la DPR au pire cas (Worst-Case DPR Time) ainsi que la rentabilité du processus de la DPR appliqué à une tâche générique, adaptée aux systèmes temps réel stricts a été proposée dans [Valente et al., 2021]. Cette méthode s'appuie sur une analyse temporelle statique, déterministe et indépendante de la cible, et peut être appliquée à toutes les plateformes actuelles offrant la fonctionnalité DPR. La formalisation du concept de rentabilité constitue une base solide pour développer des outils aidant à déterminer le moment optimal pour lancer le processus de la DPR. Cette méthode a été retenue comme modèle de coût à intégrer au sein de notre framework, en raison de son approche holistique prenant en compte l'ensemble des composants pertinents liés à la reconfiguration.

Comme le montre la figure 4.8, le processus de la DPR consiste à transférer un bitstream (BS) depuis une mémoire externe vers une zone de reconfiguration de l'FPGA, en empruntant un chemin de reconfiguration (Reconfiguration Path - RP) constitué de bus, de mémoires, de contrôleurs et d'interfaces dédiés à la reconfiguration. Le RP est une chaine de N chemins élémentaire (Elementary Paths - EPs), où le BS peut être transmis par segments de tailles variées $size_{chunk}$.

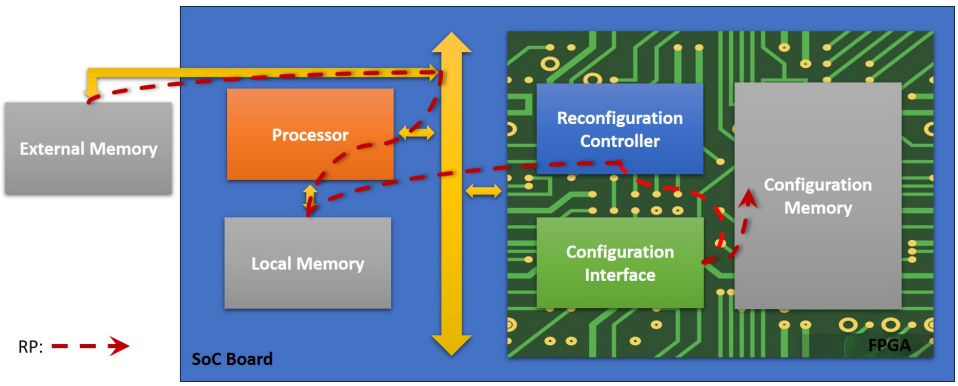



FIGURE 4.8 – *Plateforme de référence pour la DPR [Valente et al., 2021].*

Un i-ième chemin élémentaire $EP_i$ est généralement défini comme une entité maître (par exemple, un processeur ou un contrôleur DMA) qui assure le transfert de données entre deux mémoires (telles qu'une RAM et une file d'attente) à travers une interconnexion (un bus par exemple). Ainsi, le $t_{DPR}$ dépend des variables suivantes :

$$t_{DPR} = f_M(BS_{size}, TP_{i,j}, RP_{M_{i,j}}) \tag{4.1}$$

$$i = 1, ..., N \; et \; j = 1, ..., F_i$$

Tel que :

— $N$ est le nombre de chemins élémentaires EPs qui composent le chemin de reconfiguration RP.
— $F_i$ est le nombre de segments à transférer dans le i-ième chemin élémentaire $EP_i$.
— $BS_{size}$ est la taille du bitstream BS
— $TP_{i,j}$ est le débit associé au transfert de chaque segment $chunk_{ij}$ de taille $size_{chunk_{ij}}$ au sein de l'$EP_i$.
— $RP_{M_{i,j}}$ désigne la stratégie de gestion du chemin de reconfiguration RP utilisée pour orchestrer le transfert des segments entre les chemins élémentaires adjacents EPs.

En posant les hypothèses suivantes :
— hp1 : Les transferts concurrents (ou en pipeline) entre les EPs sont interdits, ce qui constitue le scénario du pire cas.
— hp2 : La taille des segments $size_{chunk_{ij}}$ est constante au sein de chaque $EP_i$ ; le pire cas est obtenu en supposant que le segment est aussi petit que possible.

Ainsi, sous l'hypothèse hp1, $f_M$ s'exprime sous forme d'une somme, et sous l'hypothèse hp2, nous obtenons : $F_i = BS_{size}/size_{chunk_i}, \forall\ EP_i$. L'équation 4.1 devient :

$$\begin{aligned} t_{DPR} &= \sum_{i=1}^{N} \left[ \sum_{j=1}^{F_i} \left( \frac{size_{chunk_{i,j}}}{TP_{i,j}} \right) \right] \\ &= \sum_{i=1}^{N} \left[ \left( \frac{BS_{size}}{size_{chunk_i}} \times \frac{size_{chunk_i}}{TP_i} \right) \right] \\ &= \sum_{i=1}^{N} \frac{BS_{size}}{size_{chunk_i}} \times t_{M_i} \end{aligned} \tag{4.2}$$

$t_{M_i}$ représente le temps nécessaire pour transférer le segment $chunk_i$ le long de l'$EP_i$. En considérant le métamodèle de la figure 4.9, le calcul de $t_{M_i}$ dépend de plusieurs paramètres et peut être formulé comme suit :

$$t_{M_i} = f_t \left( t_{MEM_i}, t_{MEM_{i+1}}, t_{INT_i}, t_{MST_i} \right) \tag{4.3}$$

$$i = 1, ..., N$$

Tel que, $\forall\ EP_i$ :

$$t_{MEM_i} = f_{MEM_i}(t^a_{MEM_i}, t^s_{MEM_i}) \tag{4.4}$$
$$t_{INT_i} = f_{INT_i}(t^t_{INT_i}, t^s_{INT_i}) \tag{4.5}$$
$$t_{MST_i} = f_{MST_i}(t^e_{MST_i}, t^s_{MST_i}) \tag{4.6}$$

Dans l'équation 4.4, $t^a_{MEM_i}$ représente le temps nécessaire pour effectuer un accès exclusif à $MEM_i$ et $t^s_{MEM_i}$ correspond au temps d'attente causé par le partage de $MEM_i$. Ces valeurs peuvent être déterminées à partir des paramètres disponibles dans la fiche technique (datasheet) de la plateforme, comme suit :

$$t^a_{MEM_i} = f^a_{MEM_i} \left( freq_{in}, freq_{out}, size_{in}, size_{out}, size_{tot}, latency \right) \tag{4.7}$$
$$t^s_{MEM_i} = f^s_{MEM_i} \left( sh_{policy} \right) \tag{4.8}$$

$freq_{in}$ et $freq_{out}$ désignent respectivement les fréquences d'entrée et de sortie de $MEM_i$. $size_{in}$ et $size_{out}$ correspondent aux tailles des ports d'entrée et de sortie, tandis que $size_{tot}$ représente la capacité totale de la mémoire. $sh_{Policy}$ est un facteur qui prend en compte le partage de la mémoire. Enfin, $latency$ est le temps nécessaire pour accéder à une donnée stockée en mémoire. Cette latence, exprimée en cycles d'horloge, est supposée

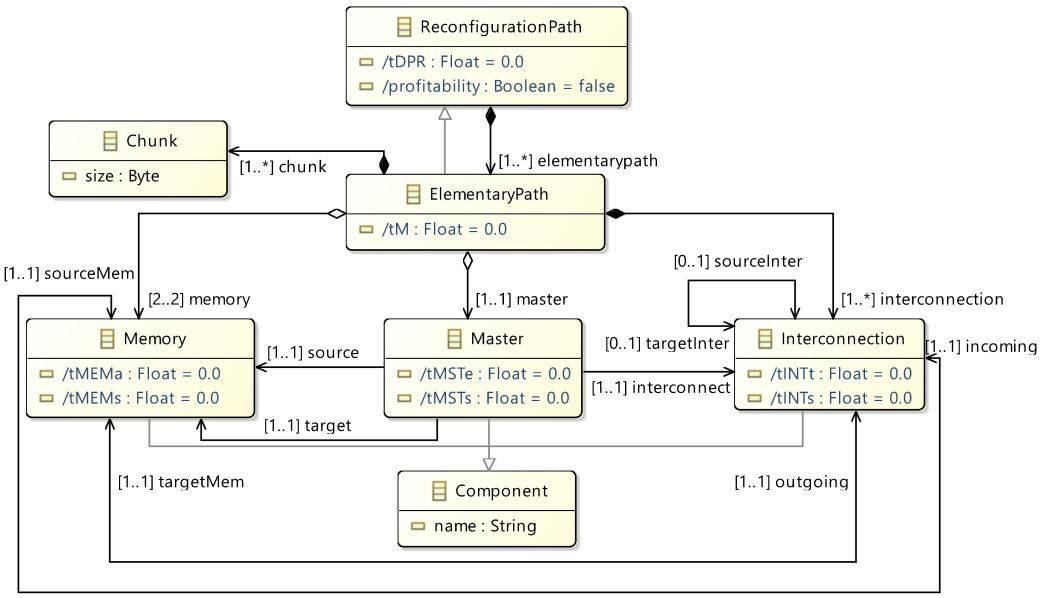



FIGURE 4.9 – *Modèle du domaine pour l'analyse de la DPR.*

constante pour tous les accès aux adresses mémoire, à l'exception du temps d'attente causé par le partage de la mémoire.

Dans l'équation 4.5, $t^t_{INT_i}$ représente le temps nécessaire pour un accès exclusif à $INT_i$, tandis que $t^s_{INT_i}$ correspond au délai provoqué par le partage de $INT_i$. Puisque $INT_i$ peut modéliser plusieurs interconnexions matérielles (comme une cascade de bus avec des tailles de données différentes), l'interconnexion à la bande passante la plus faible est alors considérée dans chaque $EP_i$. Par conséquent, l'équation 4.5 est raffiné par :

$$t^t_{INT_i} = f^a_{INT_i}\left(freq, data_{size}, beat_{size,lat,num,WT}\right) \tag{4.9}$$

$$t^s_{INT_i} = f^s_{INT_i}\left(sh_{policy}\right) \tag{4.10}$$

$freq$ désigne la fréquence de fonctionnement, $data_{size}$ correspond à la taille des données du bus, et $sh_{policy}$ représente un facteur prenant en compte le partage de l'interconnexion. Dans l'équation 4.9, $beat_{size,lat,num,WT}$ sont quatre paramètres associés à la politique de transfert du bus. En supposant une politique basée sur des rafales, chaque rafale se compose d'un certain nombre de battements $beat_{num}$, chaque battement ayant une taille définie par $beat_{size}$. La transmission de chaque battement requiert un temps spécifique, appelé $beat_{lat}$, exprimé en cycles d'horloge, auquel peut s'ajouter un temps d'attente entre battements, désigné par $beat_WT$. $t^s_{INT_i}$ est déterminé à partir des fiches techniques, tandis que $t^t_{INT_i}$ peut être détaillé de la manière suivante :

$$\begin{aligned} t^t_{INT_i} &= \sum_{k=0}^{G} burst_{ctb_k} \\ &= \sum_{k=0}^{G} \left[\left(beat_{lat_k} + beat_{WT_k}\right) \times beat_{num_k}\right] \times \frac{1}{freq} \end{aligned} \tag{4.11}$$

Où $G$ représente le nombre total de rafales nécessaires pour transférer un segment $chunk_i$, et $burst_{ctb_k}$ est la contribution de la $k^{ime}$ rafale à $t^t_{INT_i}$.

Dans l'équation 4.6, $t^e_{MST_i}$ désigne le temps d'exécution des commandes par $MST_i$, tandis que $t^s_{MST_i}$ représente le temps d'attente engendré par le partage de $MST_i$. Le

calcul de $t_{MST_i}$ est formulé par :

$$t^e_{MST_i} = f^e_{MST_i}\left(num_{cmd}, master_{ime}\right) \tag{4.12}$$

$$t^s_{MST_i} = f^s_{MST_i}\left(sh_{policy}\right) \tag{4.13}$$

Où, $\forall MST_i$, $num_{cmd}$ représente le nombre de commandes à exécuter pour gérer le transfert, $master_{ime}$ est le temps nécessaire pour exécuter une commande, et $sh_{policy}$ est un facteur à prendre en compte pour le partage de $MST_i$. $t^e_{MST_i}$ et $t^s_{MST_i}$ sont calculés à partir des fiches techniques.

Dans l'équation 4.3, la fonction $f_t$ peut être formulée comme suit :

$$t_{M_i} = G \times \left(t_{MEM_i} + t_{MEM_{i+1}} + t_{MST_i}\right) + t_{INT_i} \tag{4.14}$$

En supposant qu'un temps d'accès à la mémoire et un temps d'exécution sur le master sont nécessaires pour chaque rafale, l'équation 4.14 devient :

$$t_{M_i} = \sum_{k=0}^{G} \left(t^a_{MEM_{i,k}} + t^s_{MEM_{i,k}} + t^a_{MEM_{i+1,k}} + t^s_{MEM_{i+1,k}} + burst_{ctb_k} + t^s_{INT_{i,k}} + t^e_{MST_{i,k}} + t^s_{MST_{i,k}}\right) \tag{4.15}$$

Enfin, en utilisant l'équation 4.15 dans l'équation 4.2, il est possible de formuler l'expression de $t_{DPR}$, laquelle peut également servir au calcul du $t_{WCDPR}$. La rentabilité de la DPR $P$(Profitability) est exprimée par une variable booléenne utilisée pour déterminer l'utilité de la DPR dans l'accélération d'une tâche. $P$ est définie comme suit :

$$P = \begin{cases} 1, & \text{si } t^e_{task} + t_{DPR} \leq D_{task}, \\ 0, & \text{sinon.} \end{cases}$$

Où $t^e_{task}$ indique le temps d'exécution de la tâche accélérée, $t_{DPR}$ correspond au temps requis pour réaliser la DPR et $D_{task}$ représente l'échéance de la tâche. $t^e_{task}$ et $D_{task}$ sont déterminés lors de la phase de conception, tandis que $t_{DPR}$ est évalué à l'aide de l'équation 4.2.

## 4.5 Extension du profil MARTE

Les utilisateurs de MARTE opérant dans un domaine d'application particulier peuvent constater que MARTE n'est pas toujours assez précis ou complet pour répondre pleinement à leurs besoins en matière d'expression des concepts propres à leur projet ou domaine [Selić et Gérard, 2014]. Par exemple, bien que MARTE propose des outils pour modéliser les threads des systèmes d'exploitation, ces outils restent assez génériques et couvrent les caractéristiques communes des threads dans divers systèmes d'exploitation. Cependant, au niveau des détails d'implémentation, les différentes implémentations de threads varient d'un système à l'autre. Si les modèles doivent intégrer des détails spécifiques à une plate-forme, par exemple pour la génération automatique de code dans le cas de notre travail, il sera nécessaire d'étendre les concepts généraux de MARTE pour y inclure les informations manquantes. Dans certains cas, il se peut même que certains concepts propres à un domaine d'application n'aient pas d'équivalent dans MARTE, auquel cas c'est UML lui-même qui devra être étendu.

En général, lorsqu'un concept spécifique à un domaine manque dans MARTE, il existe trois méthodes pour l'ajouter :

1. Construction au niveau de l'application : simple et facile à réaliser puisqu'elle ne nécessite aucune connaissance du métamodèle UML. Cela consiste à ajouter des constructions spécifiques à l'application qui représentent les concepts manquants.

Si le concept doit être réutilisé dans plusieurs applications, il peut alors être stocké dans une bibliothèque de modèles personnalisée. En revanche, elle présente l'inconvénient que les sémantiques spécifiques de ces entités construites ne seront pas reconnues et, par conséquent, pourraient ne pas être correctement traitées par des outils tels que les générateurs de code ou les analyseurs de modèles.

2. Nouvelle construction du langage : Dans cette approche, une nouvelle construction du langage de première classe est ajoutée à l'aide de stéréotypes. Cela permet d'éviter le problème précédemment évoqué, mais, sans précautions adéquates, elle peut entraîner une surabondance de concepts du langage hautement spécialisés, aboutissant à un langage spécifique au domaine complexe, difficile à maîtriser et lourd à utiliser.
3. Approche hybride. Il s'agit d'une combinaison des approches 1 et 2. En pratique, l'approche hybride s'avère souvent la plus efficace. Dans ce cas, les caractéristiques principales du concept sont intégrées à la définition du stéréotype, tandis que les autres caractéristiques moins importantes sont laissées à la responsabilité du concepteur de l'application. Cette approche permet de réduire le nombre de concepts spécifiques au domaine, conduisant ainsi à un langage plus simple.

Il est toutefois difficile de définir des règles strictes pour déterminer quelle approche adopter dans une situation donnée. En général, la première approche est suffisante lorsque les sémantiques particulières des concepts ne revêtent pas d'importance majeure, comme lorsqu'il n'est pas prévu de générer automatiquement du code ou d'analyser le modèle avec des outils informatiques ce qui n'est pas le cas pour notre travail. Dans tous les autres cas, il est préférable d'opter pour la deuxième ou la troisième approche.

La norme MARTE peut être affinée grâce à sa structure hautement modulaire, qui permet de choisir et de spécialiser certains sous-ensembles, rendant possibles des extensions plus compactes. La méthode employée pour y parvenir est assez simple et parfaitement en accord avec le principe UML suivant : les définitions de stéréotypes peuvent être sous-classées. Ce processus de raffinement peut être répété de manière récursive jusqu'à obtenir le niveau de raffinement du langage désiré.

Pour couvrir les concepts manquants des domaines de la reconfiguration dynamique, du paradigme de conception des composants orientée services, du langage SystemC au niveau transactionnel, ainsi que du pattern de conception Abstract Factory, nous proposons d'étendre MARTE avec des constructions spécifiques, des stéréotypes et des valeurs étiquetées appropriées pour la modélisation et l'analyse des SoCs reconfigurables. Les extensions sont référencées dans trois sous-profils d'annotation : *MARTE for DPR (MARTE4DPR)*, *MARTE for SystemC/TLM (MARTE4SCTLM)* et *MARTE for Abstract Factory (MARTE4AF)* utilisés séparément ou conjointement, comme le montre la figure 4.10. En effet, MARTE propose de décomposer un profil en une hiérarchie de sous-profils ; cette structuration présente l'avantage de séparer les préoccupations du profil, ce qui en facilite la compréhension. De plus, chaque sous-profil peut être utilisé indépendamment, éventuellement avec les sous-profils dont il dépend. Les nouveaux stéréotypes sont définis dans différents packages de profils en dehors de MARTE laissant le profil standard inchangé. Naturellement, pour pouvoir accéder aux stéréotypes de base requis, MARTE4DPR, MARTE4SCTLM et MARTE4AF doivent importer des sous-profils MARTE.
Comme illustré par la figure 4.11, le package MARTE4DPR dépend des packages HLAM, GCM, SRM, HRM, GRM et SAM. Cela permet d'utiliser une terminologie adaptée, de donner une syntaxe et une sémantique à de nouvelles constructions, et d'ajouter des informations utilisées dans la transformation des modèles. D'autre part, le package MARTE4SCTLM dépend des packages HLAM, GCM et SRM ; l'objectif est de fournir

des extensions spécifiques pour le mapping des concepts MARTE vers leurs correspondants dans SystemC/TLM. Les packages HLAM, SRM et Alloc sont utilisés pour la création du profil MARTE4AF afin d'enrichir les modèle de conception avec les concepts du pattern Abstract Factory. Les extensions proposées permettent d'utiliser la sémantique spécifique au domaine DPR et la terminologie du langage SystemC/TLM tout en partageant les objectifs fondamentaux de MARTE. Ainsi, les Stéréotypes de base de MARTE peuvent être réutilisés en étant référencés ou spécialisés dans les sous-profils MARTE4DPR et MARTE4SCTLM. Les utilisateurs pourraient appliquer l'incrément, combiné ou non à un ou plusieurs sous-profils MARTE pour répondre à différentes préoccupations sans contraintes conflictuelles. Le sous-profil SAM est particulièrement utilisé pour réaliser une analyse d'ordonnançabilité à partir de modèles décrits avec MARTE4DPR.

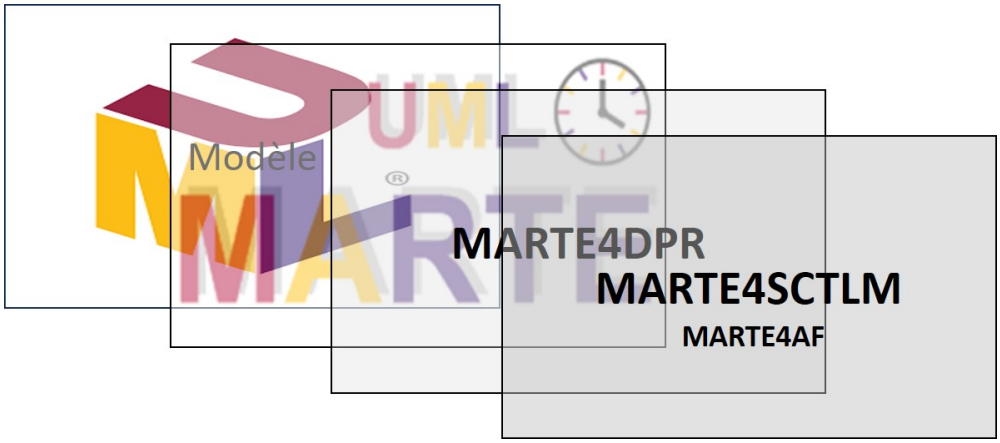



FIGURE 4.10 – *Superposition des profils de conception proposés.*

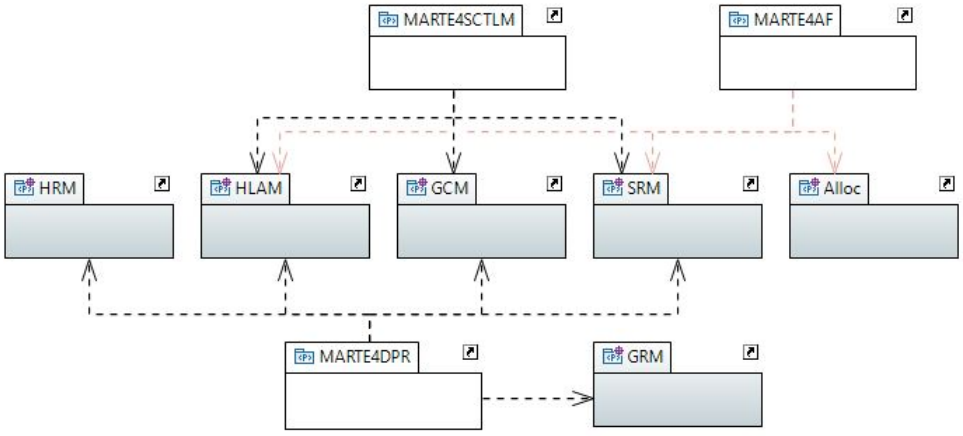



FIGURE 4.11 – *Dépendances des packages MARTE4DPR et MARTE4SCTLM.*

### 4.5.1 Le profil MARTE4DPR

Étant donné que le standard MARTE n'offre aucun support natif pour la DPR et la SOCM, des point de départ pertinents [5] se référant à des constructions clés ont été d'abord sélectionnés. Ces constructions doivent être sémantiquement alignées avec les

5. Les relevant starting points sont des éléments ou concepts préexistants qui correspondent aux objectifs de conception actuels et fournissent une base pour la personnalisation ou l'extension.

concepts du domaine. En effet, le concept de domaine et le stéréotype de base doivent représenter le même type d'entité ; de plus, le stéréotype de base ne doit pas contenir de contraintes ni de définitions de syntaxe abstraite en contradiction avec la sémantique du concept de domaine qu'il représente ; de même, aucune propriété du stéréotype de base (attributs, opérations, etc.) ne doit être en conflit avec la sémantique du concept de domaine. A partir de ces points de départ pertinents, des stéréotypes de base sont raffinés pour incorporer les concepts manquants du domaine.

Dans ce qui suit, nous présentons les différentes extensions impliquant les sous-profils de MARTE.

#### 4.5.1.1 Extension de MARTE : : HLAM

Le package HLAM a pour objectif de fournir des concepts de modélisation de haut niveau permettant d'annoter les éléments d'un modèle de façon à définir les propriétés temps réel et embarqué d'un système. Les extensions apportées à HLAM sont décrites dans la figure 4.12.

Dans MARTE4DPR, Le stéréotype « ReconfigurableUnit » spécialise le stéréotype **« RtUnit »** (Real-time Unit) en y ajoutant plusieurs propriétés liées au domaine de la DPR. Effectivement, l'unité temps réel **RtUnit** est une construction de haut niveau qui dispose d'une ou plusieurs ressources ordonnançables pouvant être créées dynamiquement. Elle présente également un comportement spécifique basé sur des états, où les états symbolisent des configurations et les transitions décrivent les reconfigurations de l'unité. Une unité temps réel peut offrir des services en temps réel et appeler les services d'autres unités temps réel. Une « ReconfigurableUnit » correspond à un élément conteneur ; elle est constituée d'une unité « FactoryUnit » intégrant la logique métier et d'une unité « ReconfigControllerUnit » responsable de la gestion du processus de reconfiguration des composants et du protocole de liaison des ports. Si une unité reconfigurable est à la fois dynamiquement et partiellement reconfigurable, alors ses attributs booléens *isDynamicallyReconfigurable* et *isPartiallyReconfigurable* doivent être évalués à *true*. L'attribut *mode* définit le rôle d'un composant conformément au principe d'interopérabilité TLM. Ses valeurs littérales peuvent être *initiator*, *target*, ou *interconnect*. L'unité *FactoryUnit* modélise le concept de fabrique abstraite, facilitant la conception d'objets apparentés sans avoir à définir leurs implémentations spécifiques. Dans une unité *FactoryUnit*, il est possible de substituer plusieurs fabriques afin d'obtenir différents comportements en créant des produits spécifiques. L'unité « ReconfigControllerUnit » gère le cycle de vie de l'instance *FactoryUnit*, ainsi que les dépendances, les transactions et la liaison des ports selon une logique de reconfiguration contrôlée par exemple par la boucle MAPE-K et représentée par l'attribut *componentReconfigProtocol*.

Le stéréotype « ServiceRegistry » mappe l'élément du domaine registre de services qui constitue une spécialisation du concept **PpUnit** (Protected passive Unit). Il assure le stockage et le partage des métadonnées de services, telles que la description du service, les configurations IP et les paramètres liés à l'utilisation des ressources. Aux types prédéfinis de MARTE s'ajoutent des types de données personnalisés et des types énumérés, conçus pour prendre en charge les diverses extensions introduites et permettre une meilleure expressivité des modèles. Par exemple, les nouveaux types de données *ServiceDescription*, *ServiceFeautures* permettent d'ajouter la sémantique de l'orienté service au profil étendu. Le stéréotype « IPCoreService » mappe les deux concepts du domaine IP et service qui spécialisent le concept **RtService**. C'est un concept abstrait qui désigne les propriétés comportementales temps réel d'un composant orienté services. Selon le type du composant, il peut être décliné en deux stéréotypes : « SoftCoreService » caractérisé

par les attributs *type* et *language* et « HardCoreService » caractérisé par les attributs *type* et *layoutModel*.

Comme le montre la figure 4.13, au début de son cycle de vie, une instance de composant orienté services entre dans l'état *Created* lorsqu'elle est initialisée pour la première fois, sans être encore prête à l'utilisation. Pour être configurée, une instance accède à l'état composite *Configured*, qui regroupe plusieurs sous-états du cycle de vie. L'instance passe de l'état *Unresolved* à *Resolved* dès que les dépendances deviennent disponibles. Après la publication de son service et l'établissement des liaisons, l'instance passe de l'état *registered* à l'état *Connected*. Une fois que la fonction attachée au service est lancée, l'instance entre dans l'état *Active*. Lorsque une demande de reconfiguration ou une notification de départ d'une dépendance est déclenchée, les canaux de communication sont verrouillés et l'instance devient *Passive*. A la réception d'un signal d'activation, le contexte est restauré et l'instance redevient *Active*. L'état *Stopped* est atteint après une demande de déconnexion et la suppression du service associé entraine le passage de l'état *Disconnected* à l'état *Unregistered*.

#### 4.5.1.2 Extension de MARTE : : GCM

Le package GCM fournit des concepts de modélisation supplémentaires pour les stratégies basées sur les composants des systèmes temps réel et embarqués. Notre attention porte principalement sur des raffinements spécifiques aux aspects de communication, afin de mieux saisir les concepts relatifs à la reconfiguration dynamique des liaisons. Une sémantique additionnelle concernant les aspects de communication nécessite une spécialisation du modèle client-serveur à travers le concept **ClientServerPort**. Ce dernier spécifie les services qu'il offre ou demande ainsi que le type du signal qu'il envoie ou reçoit. La figure 4.14 illustre l'adaptation apportée au sous-profil GCM. Les stéréotypes « ReconfigurablePort » et « ReconfigurableExport » mappent les concepts de port et export de SystemC et spécialisent le stéréotype **« ClientServerPort »** afin de prendre en charge la propriété d'évolution des interconnexions dans les systèmes reconfigurables. Si l'attribut booléen *isDynamic* est initialisé à *true*, l'interconnectivité peut changer au moment de l'exécution. L'attribut *state* représente l'état du port/export et prend deux valeur : *active* ou *passive*. Afin de gérer efficacement les dépendances du composant, le contrôleur de reconfiguration s'appuie sur une table de dépendances locale *DependencyTable*. Chaque dépendance est spécifiée par l'identifiant du composant de service distant (fournisseur), l'identifiant de son port *remotePortId* et l'état de la liaison *connectionState* qui peut prendre les valeurs : *connected*, *disconnected*, *reconfigured* ou *reconnected*.

Comme le montre la figure 4.15, lorsqu'une reconfiguration de composant (évolution structurelle) est initiée, tous les ports/exports reconfigurables doivent être mis à l'état *passive*, et les canaux de communication bloqués. Une fois les signaux de connexion/-reconnexion requis interceptés, les ports/exports reconfigurables seront réactivés (état *active*). Quand il s'agit d'une reconnexion, le contexte est restitué.

#### 4.5.1.3 Extension de MARTE : : GRM

Le package GRM fournit les concepts nécessaires à la modélisation des plateformes au niveau système, destinées à l'exécution d'applications embarquées en temps réel. Pour modéliser le logiciel et le matériel, GRM est respectivement spécialisé en SRM et HRM. Il peut également être affiné davantage pour intégrer des fonctionnalités de reconfiguration. Dans la norme, le concept de plateforme a été abstrait sous la forme de ressources, distinguant ainsi les ressources de communication et les ressources de calcul. Une ressource **Resource** désigne une entité physique ou logique persistante qui

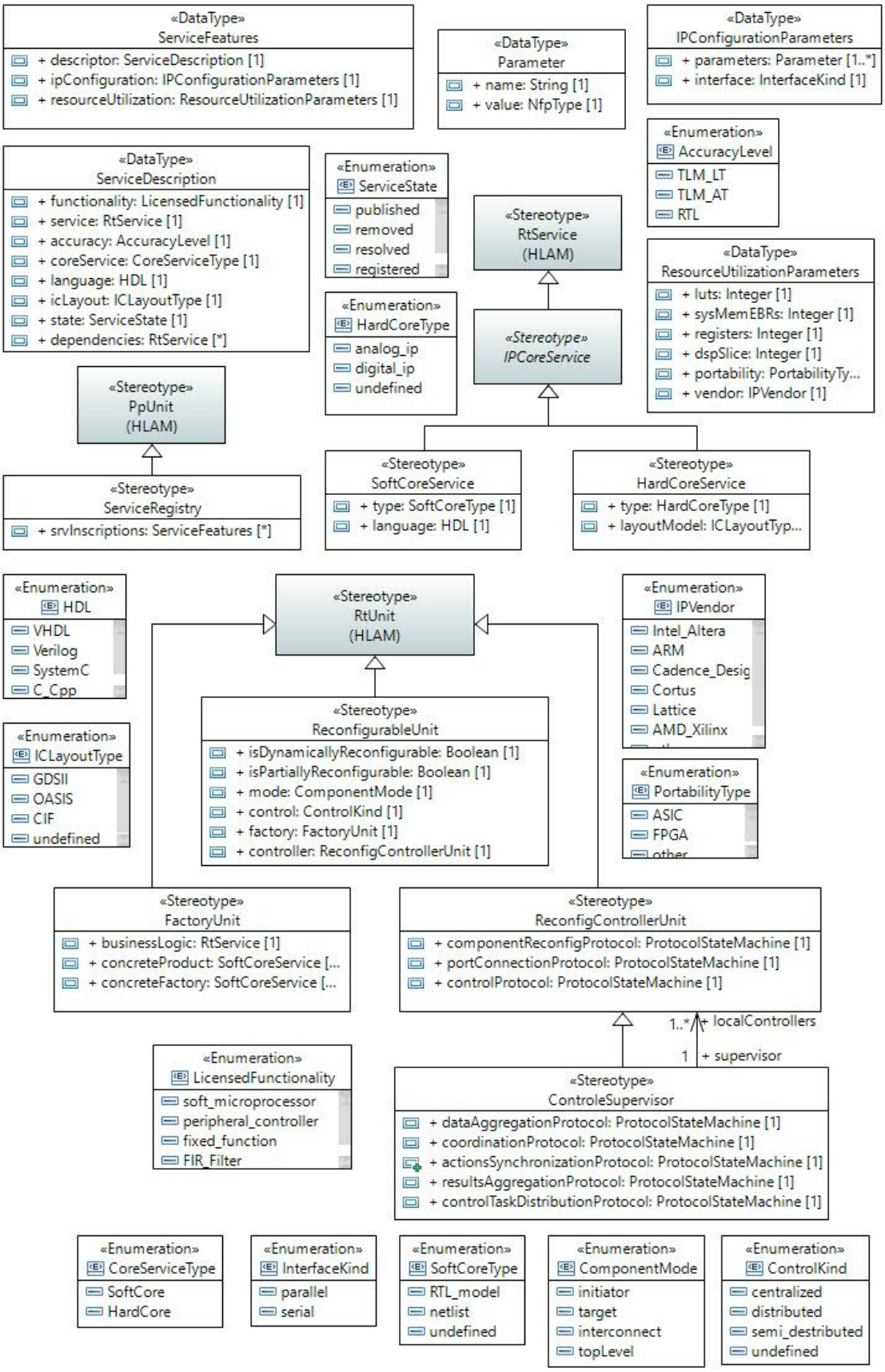



FIGURE 4.12 – *L'extension du sous profil HLAM.*

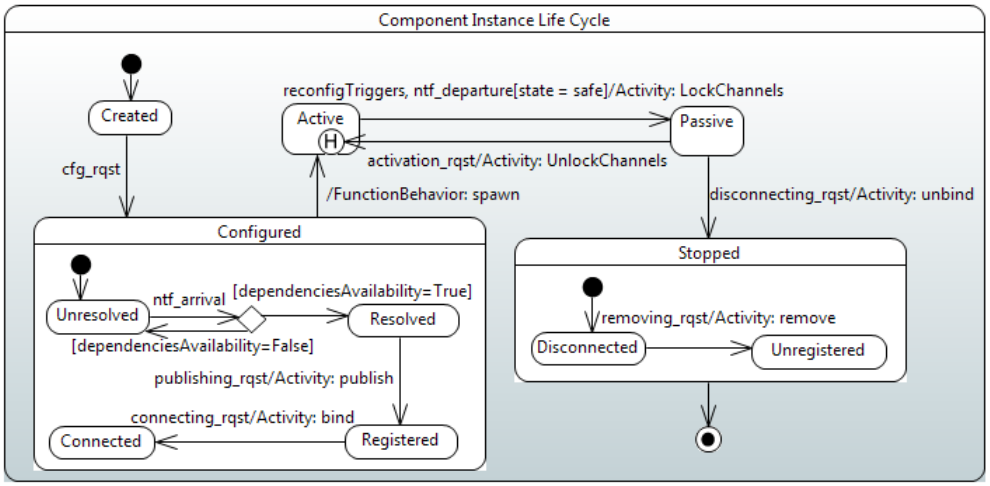



FIGURE 4.13 – *Le cycle de vie d'une instance d'un composant orienté services.*

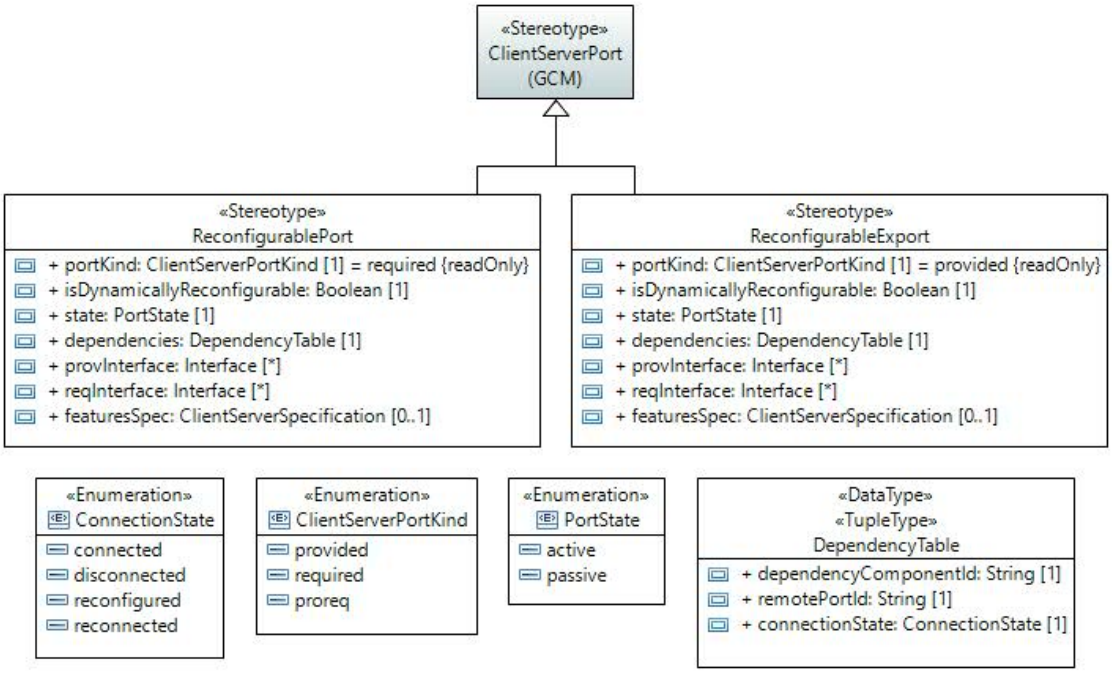



FIGURE 4.14 – *L'extension du sous profil GCM.*

fournit un ou plusieurs services **ResourceService**. Elle permet de modéliser la plateforme d'exécution sur le plan structurel, tandis que les services associés en décrivent le comportement. Etant donné que la notion de **Resource** est un concept central du GRM, elle est étendue dans MARTE4DPR par l'ajout du stéréotype « ReconfigResource » mappant le concept de ressource reconfigurable. Les attributs booléens *isReconfigurable*, *isDynamicallyReconfigurable* et *isPartiallyReconfigurable* permettent de spécifier si le ressource est dynamiquement et partiellement reconfigurable. L'attribut *IsReconfigurable* est initialisé à *true* et il est immuable, d'où la présence de la contrainte OCL *ReadOnly*. La sémantique de l'élément du domaine **ressource** est ainsi enrichie et peut être propagée à travers ses sous-types : **ComputingResource**, **CommunicationMedia**, **ConcurrencyResource**, **DeviceResource**, **SchedulableResource** et **ProcessingResource** comme illustré à la Figure 4.16.

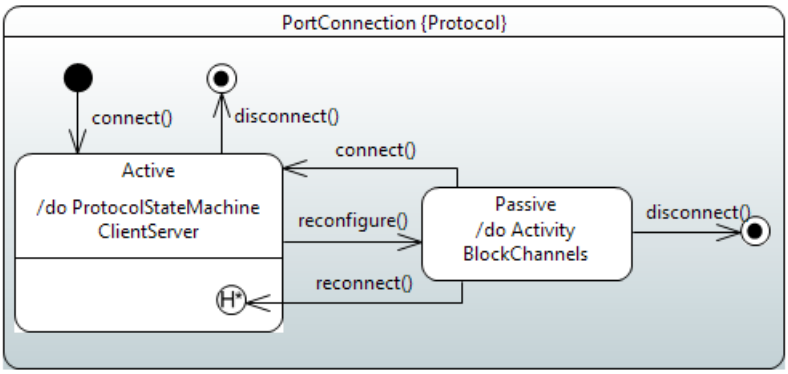



FIGURE 4.15 – *Le protocole de connexion du port reconfigurable.*

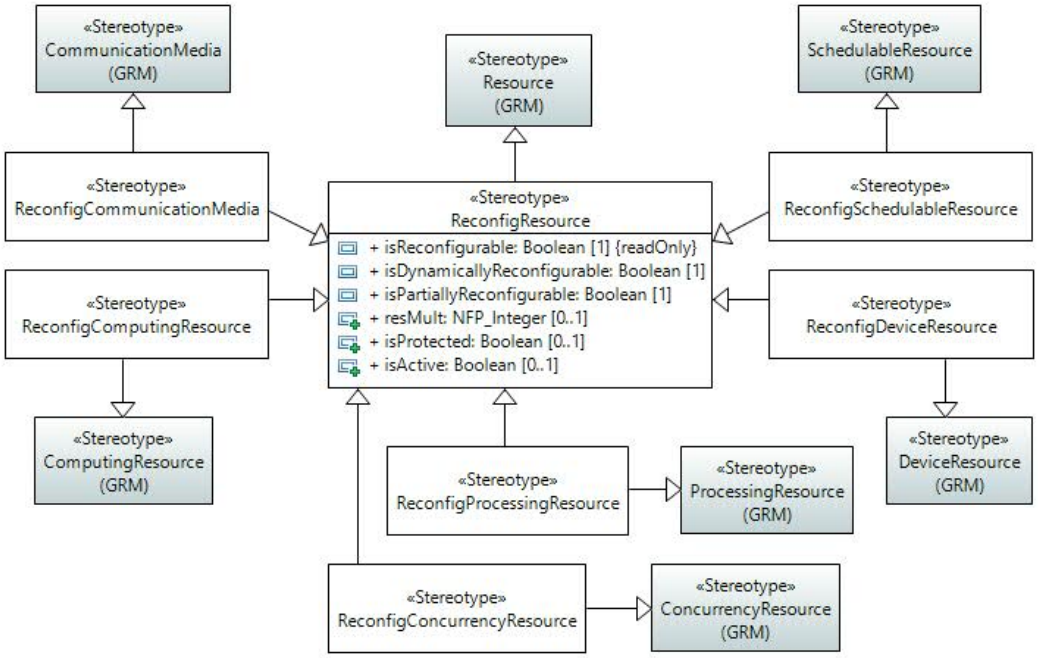



FIGURE 4.16 – *L'extension du sous-profil GRM.*

#### 4.5.1.4 Extension de MARTE : : HRM

Le package HRM regroupe la plupart des concepts matériels sous une taxonomie hiérarchique avec plusieurs catégories en fonction de leur nature, fonctionnalité, technologie et forme. La séparation des préoccupations et l'abstraction sont les principales caractéristiques de ce profil. HRM est composé de deux vues complémentaires et convergentes : une vue logique qui fournit une classification fonctionnelle des entités matérielles basée sur les services offerts par chaque ressource et une vue physique qui se concentre sur leurs propriétés physiques telles que la forme, la taille et la position au sein de la plateforme, la consommation d'énergie et la dissipation thermique. Pour prendre en charge les fonctionnalités de la DPR, il serait intéressant d'étendre la sémantique de certains stéréotypes à partir de modèles logiques ayant leurs correspondants physiques tels que **HwLogical : : HwComputing : : HwPLD** et **HwPhysical : : HwLayout : : HwComponent** comme décrit dans la figure 4.17.

Le package HW_Layout fournit une classification des composants matériels en fonction de leurs formes et propose des constructions d'arrangement utilisant des grilles rectilignes. Le stéréotype « **HwComponent** » mappe l'élément de domaine

**HW_Component** qui est l'entité physique principale du package HW_layout. Il est spécialisé par le stéréotype « ReconfigHwComponent » possédant des attributs supplémentaires liés aux fonctionnalités de reconfiguration. Si un **HwComponent** est dynamiquement et partiellement reconfigurable, les attributs booléens *isDynamicallyReconfig* et *isPartiallyreconfig* doivent être définis à *true*. Les attributs *isActive* et *isReconfigurable* sont hérités de **Resource** et déclarés en lecture seule (contrainte OCL *readOnly*). Quand il est à *true*, l'attribut *isActive* signifie que le composant reconfigurable dispose de son propre plan d'action qui lui permet d'exécuter ses services de manière autonome.

Le stéréotype **« HwPLD »** mappe une ressource matérielle programmable, qui a une organisation particulière et peut posséder plusieurs IPs, câblées ou non, telles que des processeurs, des mémoires et des dispositifs analogiques. Le stéréotype « FPGA » est un **« HwPLD »** spécialisé qui contient d'autres attributs spécifiques. L'attribut *granularité* prend ses valeurs dans l'énumération *GranularityType* ; la capacité de relocalisation dynamique est indiquée par l'attribut booléen *isRuntimeReloc*. L'attribut *regions* du FPGA désigne les caractéristiques des régions en termes de *dimensions* et de type (*kind*). Alors que l'attribut *designFlow* spécifie le style du flot de conception de la DPR, l'attribut *configInterface* indique les modes d'accès à l'interface de configuration du FPGA.

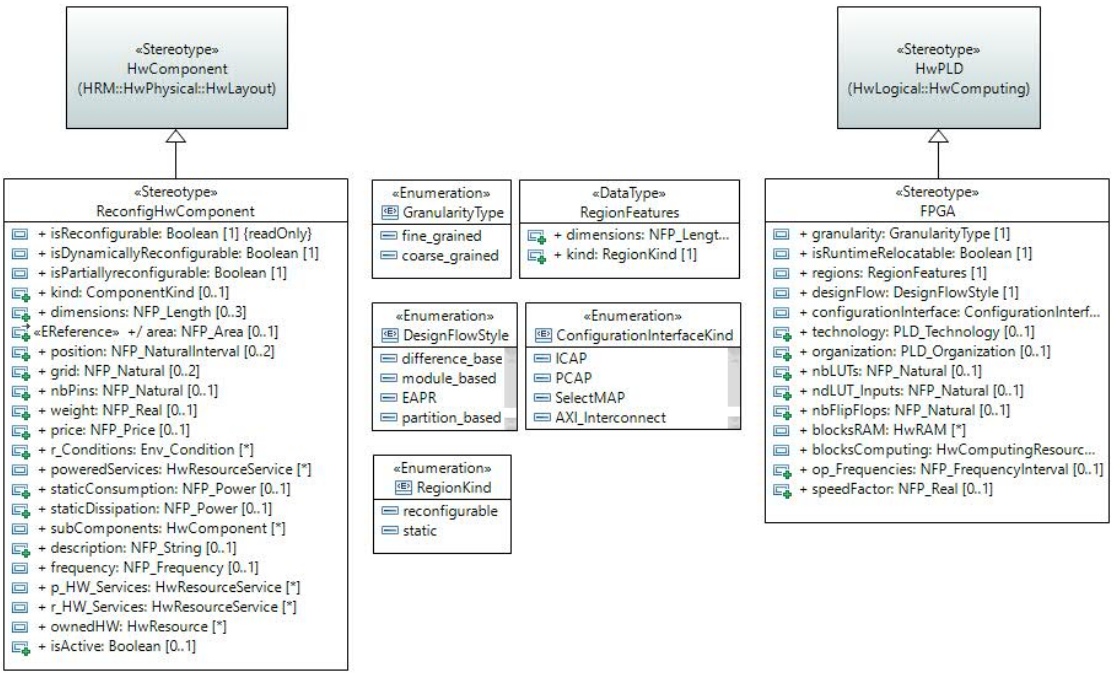



FIGURE 4.17 – *L'extension du sous-profil GRM.*

#### 4.5.1.5 Extension de MARTE : : SRM

Le package SRM est une spécialisation des ressources et services définis dans le package GRM pour fournir un support à la modélisation des plateformes logicielles multitâches. Nous nous intéressons principalement ici à la modélisation des contextes d'exécution concurrents décrits dans le package SW_Concurrency.

Comme le montre la figure 4.18, le stéréotype « FactoryProcess » représente un contexte de traitement métier propre à une unité de fabrique. Il étend le stéréotype **« SwSchedulableResource »** pour permettre la modélisation de processus générés statiquement ou dynamiquement. L'attribut *spawnedFunction* désigne la fonction associée à l'instance du processus engendré. Le stéréotype « ReconfigControllerProcess » offre des

services dédiés au contrôle local du processus de reconfiguration, il spécialise **« SwSchedulableResource »** pour la prise en charge des concepts de la boucle de contrôle MAPE-K, de la résolution des dépendances dynamiques (*dependenciesResolution*) entre services, le déclenchement des reconfigurations (*reconfigurationTriggers*), ainsi que la publication et la découverte de services. D'autres propriétés liées aux services, politiques d'ordonnancement et à la gestion des ressources sont hérités de « SwSchedulableResource ». Le stéréotype « ReconfigSupervisorProcess » mappe le concept de superviseur de reconfiguration dans un système de contrôle semi-distribué en spécialisant « ReconfigControllerProcess ». Il modélise des services de coordination, d'agrégation et de mise à jour des règles, politiques et stratégies globales d'adaptation utilisées par le superviseur pour coordonner les contrôleurs locaux et optimiser les performances du système. L'opération de coordination est centrale pour le superviseur, elle assure que toutes les actions des contrôleurs locaux s'alignent avec les objectifs globaux et que les ressources partagées sont utilisées efficacement. La politique globale su système de contrôle englobe des règles prédéfinies pour l'allocation des ressources et la gestion des conflits ainsi que des objectifs globaux à atteindre tels que minimiser le temps de reconfigurations et optimiser l'utilisation des ressources. Les données collectées des contrôleurs locaux sont consolidées en utilisant des techniques d'agrégation appropriées telles que le calcul des moyennes, des médianes ou des max/min, modélisées par l'attribut *aggregationServices*. L'attribut *updateStrategies* mappe l'opération de mise à jour des stratégies qui vise à adapter et améliorer les stratégies utilisées pour gérer les ressources, synchroniser les tâches et résoudre les conflits en fonction des retours d'expérience, des performances actuelles, et des changements dans l'environnement. L'ensemble des stratégies, politiques et objectifs d'adaptation sont accessibles depuis la base de connaissances, modélisée par la propriété *knowledgeResources*.

### 4.5.2 Le profil MARTE4SCTLM

Le profil MARTE se limite à une approche de modélisation conceptuelle, sans inclure de mécanismes ou de dépendances natifs à un langage de programmation donné. Son objectif est de fournir une abstraction haut niveau pour la conception et l'analyse, facilitant ainsi son intégration avec des environnements multi-langages. Ainsi, l'absence de support natif pour le langage SystemC/TLM impose l'extension du profil afin de modéliser et intégrer les différentes constructions spécifiques à SystemC/TLM. Ces éléments servent de points de départ pertinents pour l'extension de MARTE en vue de la génération automatique de code en SystemC/TLM ciblant les SoCs reconfigurables, malgré l'absence de support natif de SystemC pour la DPR. Néanmoins, la fonction *sc_spawn* permet de générer des processus dynamiques pendant l'exécution de la simulation (voir annexe B). Cela le rend particulièrement utile pour modéliser des comportements où le nombre de processus n'est pas connu à l'avance ou varie avec le temps, une propriété très intéressante pour l'instanciation dynamique de comportements.

Un module SystemC est le plus petit conteneur de fonctionnalité intégrant un état, un comportement et une structure pour une connectivité hiérarchique. Le stéréotype « Sc_Module » mappe l'élément de domaine *SC_Module*. Il spécialise le stéréotype **« HLAM : :RtUnit »** en y ajoutant des attributs supplémentaires. Le rôle d'un composant au sein de la couche d'interopérabilité TLM est défini par l'attribut *role*, dont les valeurs proviennent de l'énumération *initiator*, *target*, *interconnect*, *top*. Les attributs *variables* et *helpers* définissent les variables membres locales et les fonctions helpers déclarées dans « Sc_Module ». L'ensemble du profil est présenté dans la figure 4.19.

*SC_PrimChannel* est une construction SystemC utilisée pour implémenter la communication entre processus et l'interconnexion des modules. Le stéréotype


FIGURE 4.18 – *L'extension du sous profil SRM.*

« Sc_Prim_Channel » spécialise le stéréotype « **MessageComResource** » du package SRM : :SW_Interaction avec une sémantique spécifique appropriée à la modélisation au niveau de la transaction. En effet, « **MessageComResource** » définit une ressource de communication pour échanger des messages (structure de données) qui est conforme à la sémantique de la file d'attente d'événements de charge utile TLM. Les attributs *sensitivityList* et *triggerList* permettent de spécifier les arguments passés aux fonctions membres *wait* et *next_trigger*, respectivement.

Dans SystemC, une instance de processus peut être créée en appelant les macros *sc_method*, *sc_thread* ou *sc_cthread* ou en appelant la fonction *sc_spawn*. Il est possible de reconnaître les processus engendrés, non engendrés, statiques et dynamiques en fonction du rappel (callback) de la phase d'exécution et de la fonction/macro à partir de laquelle l'instance de processus est créée. Les processus engendrés dynamiquement sont les plus appropriés pour modéliser et simuler des systèmes reconfigurables dyna-

miques. En effet, dans de tels systèmes, des éléments sont générés ou éliminés pendant l'exécution du système, ce qui peut être naturellement spécifié par des processus engendrés dynamiquement créés à partir du rappel *end_of_elaboration* ou pendant la simulation. Le stéréotype **« SwSchedulableResource »** du sous-profil SW_Concurrency est spécialisé par les stéréotypes « Sc_Process » et « Sc_Spawn ». Le concept *Sc_Process* est abstrait, il est étendu par les stéréotypes « Sc_Method », « Sc_Thread » et « Sc_Cthread » ; lorsqu'il est généré dynamiquement, ses attributs *isDynamic* et *isSpawned* doivent être définis à *true*. La sensibilité statique de l'instance de processus est spécifiée par l'attribut *sensitiveList* ; tandis que la sensibilité dynamique est indiquée dans les stéréotypes étendus par les attributs *waitArgs* et *nextTriggerArgs*. L'attribut *function* définit la fonction membre associée à l'instance de processus. Le stéréotype « Sc_Spawn » est appliqué aux fonctions à générer en tant que processus. Si son attribut *isDynamic* est initialisé à *true*, l'instance du processus est générée dynamiquement. Le processus ou le module à partir duquel la fonction *sc_spawn* est appelée est spécifié par l'attribut *parent*, tandis que l'attribut *options* désigne les propriétés possibles de l'instance du processus.

TLM implique la communication entre processus via des appels de méthodes d'interface à travers des ports et des exports. Un port (respectivement export) définit un ensemble de services requis (respectivement fournis) par le module contenant le port (respectivement export). Les sockets sont utilisés pour transmettre des transactions entre les initiateurs et les cibles. Techniquement, un *socket initiateur* est dérivé de la classe *sc_port* et possède un *sc_export*, et vice versa pour un *socket cible*. D'autre part, le port client-serveur **ClientServerPort** dans MARTE prend en charge un modèle de communication basé demande/réponse et spécifie un ensemble de services fournis/requis, ainsi que le type de signaux produits/consommés représentés par des messages. Pour ces raisons, dans MARTE4SCTLM, le stéréotype **« GCM : : ClientServerPort »** est spécialisé par les stéréotypes « Sc_Port » et « Sc_Export » qui à leur tour sont respectivement étendus par les stéréotypes « TLM_Initiator_Socket » et « TLM_Target_Socket ». Ces derniers sont également spécialisés par les stéréotypes « Simple_Initiator_Socket » et « Simple_Target_Socket » pour mapper des sockets particuliers facilitant la génération dynamique des processus. L'attribut *module* spécifie le composant propriétaire du *socket*, tandis que l'attribut *types* permet de paramétrer le *socket initiateur/cible* avec le type de protocole par défaut *TLM_Base_Protocol_Types*, comprenant la charge utile générique *genericPayload* et la phase du protocole de base.

Les interfaces principales de TLM-2.0 comprennent les interfaces de transport bloquantes et non bloquantes, l'interface de mémoire directe (DMI) et l'interface de transport de débogage. Tandis que l'interface de transport bloquant n'utilise que le chemin direct de l'initiateur vers la cible et prend en charge le style de codage LT, l'interface de transport non bloquant utilise à la fois le chemin direct de l'initiateur vers la cible et le chemin inverse de la cible vers l'initiateur, et elle prend en charge le style de codage AT. Le concept *Sc_Interface* dans SystemC correspond au concept **ClientServerSpecification** de MARTE. Le stéréotype abstrait « Sc_Interface » est raffiné en plusieurs stéréotypes qui correspondent aux interfaces de base. Les stéréotypes « TLM_Fw_NonBlocking_Transport_If » et « TLM_Bw_NonBlocking_Transport_If » spécialisent deux interfaces non bloquantes distinctes à utiliser sur des chemins opposés. Ils disposent de quatre attributs supplémentaires : *trans*, *phase*, *delay* et *sync* pour spécifier respectivement le type de transaction *TLM_Generic_Payload*, le type de phase *TLM_phase*, l'annotation temporelle de type *Sc_Time* et la valeur de retour de la synchronisation *TLM_Sync*.

La charge utile générique peut être utilisée comme type de transaction à usage général pour la modélisation abstraite de bus à mémoire mappée ou comme base pour modéliser une large gamme de protocoles spécifiques à un niveau plus dé-

taillé. Pour maximiser l'interopérabilité entre les modèles, le type de transaction par défaut *TLM_Generic_Payload* doit être utilisé avec le protocole de base. Dans le profil MARTE4SCTLM, le stéréotype « TLM_Generic_Payload » spécialise le stéréotype **« GQAM : : GaStep »** avec un ensemble standard d'attributs *command*, *adress*, *dataPointer*, *dataLength*, etc.

Le stéréotype « Sc_Event » spécialise le stéréotype **« GCM : : DataEvent »**. Si l'événement est déclaré dans structure hiérarchique alors l'attribut booléen *isInHierarchy* doit être défini à *true*. Chaque instance de processus peut avoir sa propre sensibilité statique qui est établie immédiatement après son enregistrement. Le stéréotype « Sc_Sensitivity » mappe la sémantique de la sensibilité statique en spécialisant le stéréotype **« GCM : : GCMTigger »**; l'attribut *event* désigne l'ensemble des événements pouvant entraîner la reprise ou le déclenchement d'un processus. Le stéréotype « Container » mappe un conteneur C++ et spécialise le stéréotype **« HLAM : : PpUnit »** pour la transformation de modèles.

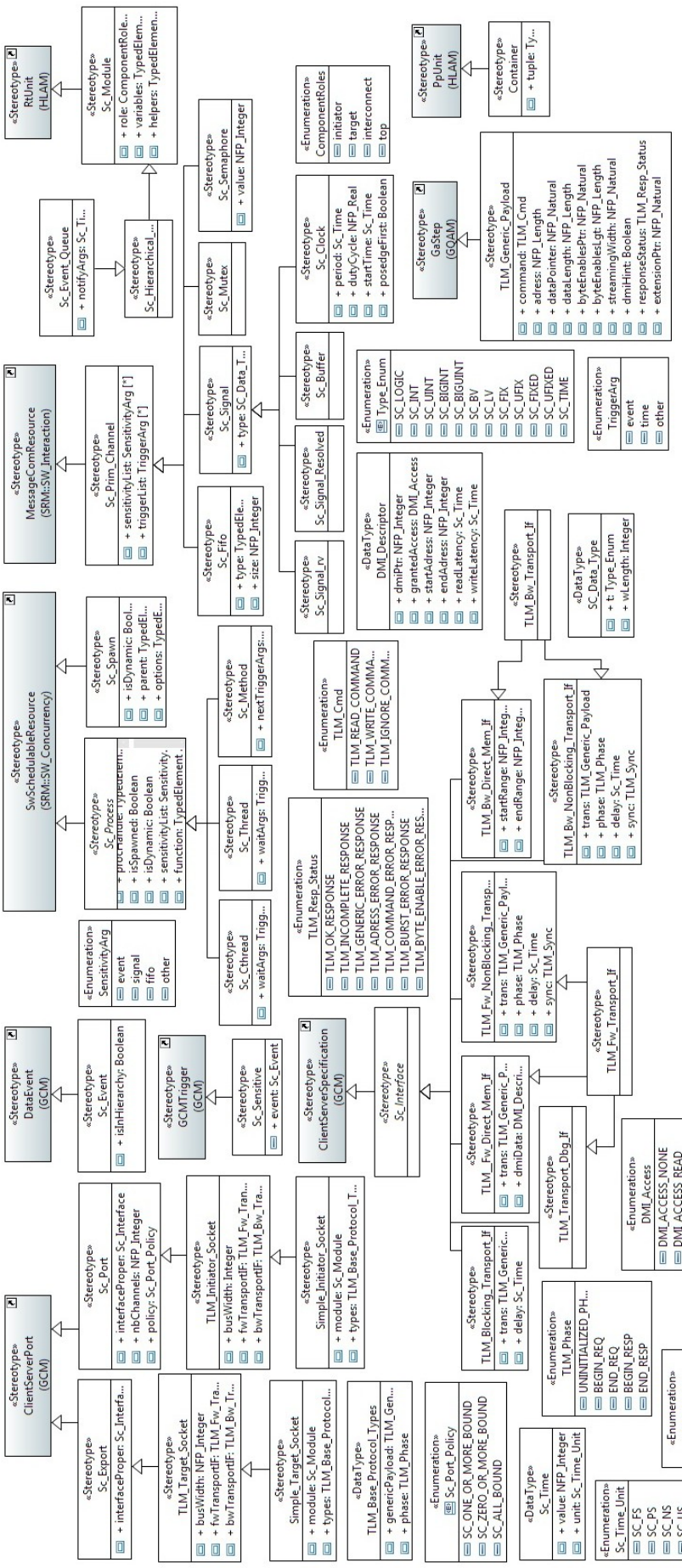


FIGURE 4.19 – *Le profil MARTE4SCTLM.*

### 4.5.3 Le profil MARTE4AF

En s'appuyant sur les concepts définis dans le métamodèle du patron de conception Abstract Factory présenté dans la section 4.4.1.1, le profil MARTE a été enrichi par l'introduction d'un ensemble de nouveaux stéréotypes. Ces derniers permettent de mapper les éléments du domaine associés au pattern Abstract Factory, aboutissant ainsi à la définition du profil MARTE4AF. L'objectif principal de MARTE4AF est de compléter le profil MARTE4SCTLM, en vue de réduire la distance sémantique entre les concepts abstraits des modèles MARTE et le code SystemC/TLM généré. La figure 4.20 illustre la structure du profil MARTE4AF, et met en évidence les différents packages ainsi que les stéréotypes standards utilisés dans cette extension. À la différence d'un mécanisme d'allocation, qui intervient entre modèles indépendants, le raffinement consiste à modifier la perspective sur une structure comparable existante au sein d'un même modèle. Un raffinement s'applique soit exclusivement aux éléments du modèle applicatif, soit exclusivement aux éléments du modèle de la plateforme d'exécution. Dans cette logique, le stéréotype **« NfpRefine »** du package Alloc est spécialisé par le stéréotype « Creates », afin de permettre son application au modèle de simulation SystemC/TLM, facilitant ainsi la transition vers des artefacts de bas niveau tout en conservant la traçabilité entre les concepts abstraits et leur implémentation concrète. Les stéréotypes « CreationMethod » et « FactoryMethod » constituent des spécialisations du stéréotype **« SwSchedulableResource »**, en contraignant le type de retour à AbstractProduct et ConcreteProduct respectivement. Les stéréotypes « ConcreteFactory » et « ConcreteProduct » spécialisent le stéréotype **« RtUnit »** en mettant l'attribut *isDynamic* à *true*. Cette configuration permet de modéliser des classes actives dont le comportement interne est spécifié de manière dynamique

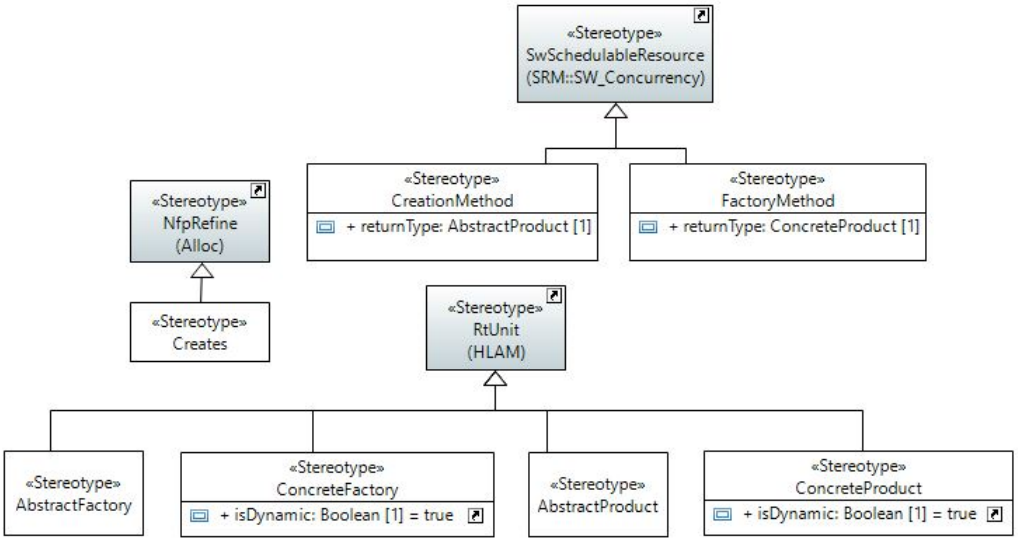



FIGURE 4.20 – *Le profil MARTE4AF.*

## 4.6 TRANSFORMATION DE MODÈLES

Concept clé de l'IDM, la transformation de modèles repose sur un processus de raffinement qui réduit le niveau d'abstraction des modèles en ajoutant suffisamment de détails aux modèles mappés de manière à rendre possible la génération automatique de code. A partir de modèles UML annotés avec le profil MARTE4DPR et/ou

MARTE4SCTLM, des transformations exogènes et verticales sont réalisées, aboutissant à la génération de code SystemC/TLM. Pour développer notre chaîne d'outils, nous avons combiné des transformations M2M afin de générer un modèle cible SystemC/TLM à partir d'un modèle source conforme à MARTE4DPR, ainsi que des transformations M2T pour convertir le modèle obtenu en code source SystemC/TLM.

### 4.6.1 Transformation M2M

Dans notre framework, la transformation M2M s'exprime au moyen de ATL, un langage de transformation de modèles et une boîte à outils développés sur la plateforme Eclipse. Un modèle source conforme à MARTE4DPR est transformé en un modèle cible conforme à SystemC/TLM en suivant un modèle de transformation conforme au métamodèle ATL. Etant donné qu'ATL est un langage textuel basé sur des règles, la génération d'éléments de modèle SystemC/TLM est effectuée en appliquant des règles déclaratives écrites au niveau du métamodèle. Afin de spécifier ces règles de correspondance (matched rules), il est nécessaire de trouver des similitudes entre les concepts des modèles source et cible pour relier leurs métamodèles respectifs.

Bien qu'il existe une distance sémantique entre les concepts de UML/MARTE et ceux de SystemC, l'écart peut être réduit en se basant sur des équivalents conceptuels. Ainsi, la proximité entre les concepts est principalement évaluée en termes de similarités sémantique et syntaxique. Cela simplifie l'établissement de correspondances (mappings) entre des éléments similaires des modèles UML/MARTE, MARTE4DPR et SystemC/TLM.

Dans le cadre de la DPR, la convergence entre UML/MARTE et SystemC/TLM est facilitée par le profil MARTE4DPR. Ce dernier assure une compatibilité avec UML/-MARTE tout en partageant avec SystemC un intérêt commun pour le domaine des systèmes temps réel et embarqués. Le tableau 4.1 illustre le mapping entre des concepts sémantiquement et syntaxiquement alignés d'UML, MARTE, MARTE4DPR et SystemC/TLM, en prenant en compte les aspects structurels et comportementaux liés à la DPR.

La génération des éléments du modèle cible repose sur la définition de règles de transformation. Un aperçu de l'ensemble des règles de correspondance utilisées dans notre framework est montré dans la figure 4.21. Une fois les règles déclaratives déclenchées, il est nécessaire de parcourir et de mettre en correspondance les éléments du modèle MARTE4DPR pour créer et initialiser les éléments du modèle cible. Comme le montre le listing 4.1, le module de transformation ATL est composé de helpers, d'éléments de modèle et de règles de correspondance standard écrites dans un style déclaratif. Les helpers **isSpawned** et *isDynamic* sont définis dans le contexte de l'élément *FactoryProcess* et utilisés pour calculer des valeurs booléennes indiquant si un processus de fabrique est généré dynamiquement. La règle *ReconfigurableUnit2SCModule* a pour objectif de générer un module SystemC Sc_Module à partir d'une unité reconfigurable et dont les attributs sont initialisés à l'aide de liaisons. A partir de l'élément *System*, un module top-level est créé et l'arborescence modulaire de SystemC est générée. Les sous-modules, les canaux de communication ainsi que le registre de services sont générés par la règle *System2SCModule*. En utilisant la règle *FactoryUnit2SCModule*, un module SystemC est généré à partir d'une unité de fabrique, avec l'initialisation du processus, des ports, de la configuration et des variables associés. La règle *FactoryProcess2SCThread* génère un thread SystemC à partir d'un processus de fabrique en initialisant la fonction adjacente et la liste des sensibilités.

TABLE 4.1 – *Le mapping des concepts.*

| UML Concepts | MARTE Concepts | MARTE4DPR Concepts | SystemC/TLM Concepts |
|---|---|---|---|
| Active Class | HLAM : :RtUnit {isDynamic=true} | ReconfigurableUnit | SC_Module |
| Active Class | HLAM : :RtUnit {isDynamic=true} | FactoryUnit | SC_Module |
| Active Class | HLAM : :RtUnit {isDynamic=true} | ReconfigControllerUnit | SC_Module |
| Operation, StateMachine | SRM : :... : :SwSchedulableResource | FactoryProcess | SC_Thread, SC_Method |
| Operation, StateMachine | SRM : :... : :SwSchedulableResource | RecconfigControllerProcess | SC_Thread, SC_Method |
| Port | GCM : :ClientServerPort | ReconfigurablePort | TLM_Initiator_Socket TLM_Target_Socket |
| Interface | GCM : :ClientServerSpecification | ClientServerSpecification | SC_Interface |
| Class | HLAM : :PpUnit | ServiceRegistry | C++ Container |
| BehavioralFeature | HLAM : :RtService | IPCoreService | C++ Function |
| DataType | VSL : :DataTypes : :TupleType | ServiceDescriptor | C++ Tuple |
| Message | GQAM : :... : :GaStep | Payload | TLM_Generic_Payload |
| Class, Connector | SRM : :... : :MessageComResource | MessageComResource | SC_Prim_Channel |
| Event | GCM : :DataEvent | DataEvent | SC_Event |
| Trigger | GCM : :GCMTrigger | GCMTrigger | SC_Sensitive |
| Class | HRM : :HwLogical : :... : :HwPLD | FPGA | SC_Module |
| Class | GRM : :Resource { isActive=true } | Resource | SC_Module |
| Class | HRM : :... : : HwComponent | ReconfigHwComponent | SC_Module |
| BehavioralFeature | GRM : :GrService | GrService | C++ Function |

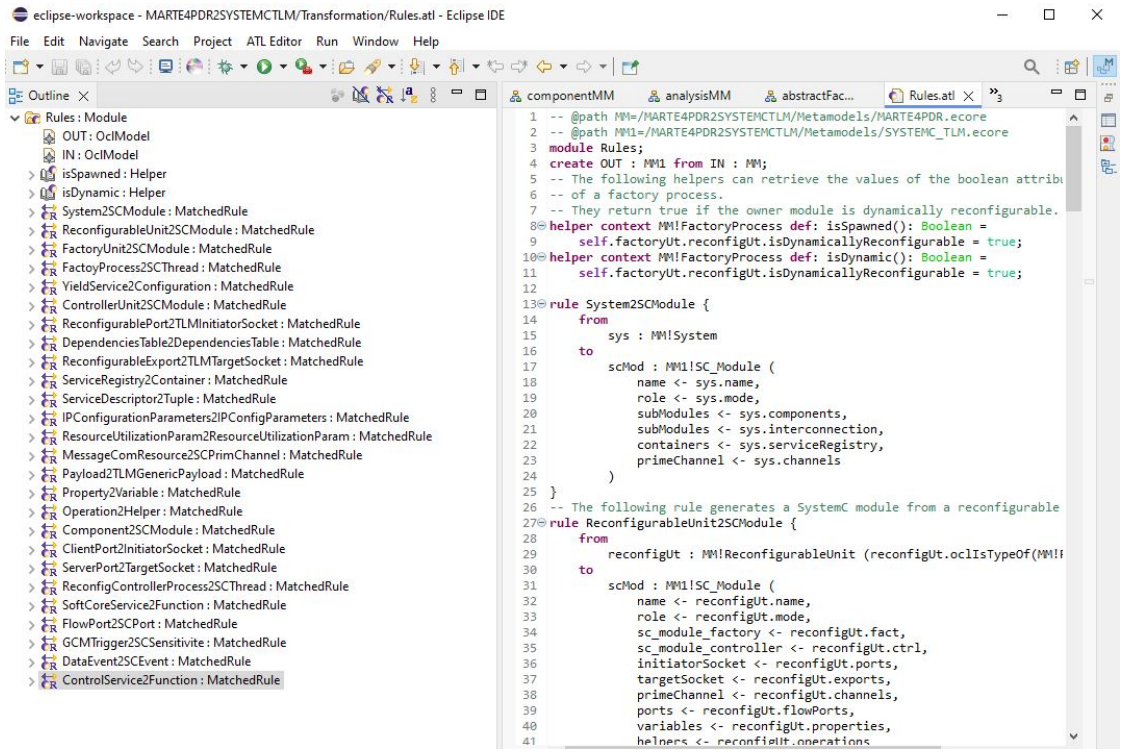



FIGURE 4.21 – *Aperçu des règles de transformation ATL pour la génération de modèles SystemC/TLM.*

```
1 -- @path MM=/MARTE4PDR2SYSTEMCTLM/Metamodels/MARTE4PDR.ecore
2 -- @path MM1=/MARTE4PDR2SYSTEMCTLM/Metamodels/SYSTEMC_TLM.ecore
3 module Rules;
4 create OUT : MM1 from IN : MM;
5 -- The following helpers can retrieve the values of the boolean
```

```
    attributes of a factory process. They return true if the owner
    module is dynamically reconfigurable.
helper context MM!FactoryProcess def: isSpawned(): Boolean = self.
    factoryUt.reconfigUt.isDynamicallyReconfigurable = true;
helper context MM!FactoryProcess def: isDynamic(): Boolean =  self.
    factoryUt.reconfigUt.isDynamicallyReconfigurable = true;
-- The following rule generates a sc_module from a reconfigurable unit
    .
rule ReconfigurableUnit2SCModule {
  from
    reconfigUt :MM!ReconfigurableUnit (reconfigUt.oclIsTypeOf(MM!
       ReconfigurableUnit))
  to
      scMod : MM1!SC_Module (
      name <- reconfigUt.name,
      role <- reconfigUt.mode,
      sc_module_factory <- reconfigUt.fact,
      sc_module_controller <- reconfigUt.ctrl,
      initiatorSocket <- reconfigUt.ports,
      targetSocket <- reconfigUt.exports,
      primeChannel <- reconfigUt.channels,
      ports <- reconfigUt.flowPorts,
      variables <- reconfigUt.properties,
      helpers <- reconfigUt.operations  )
}
-- Hierarchical sc_module creation.
-- The following rule generates the top level sc_module.
rule System2SCModule {
  from
    sys : MM!System
  to
    scMod : MM1!SC_Module (
      name <- sys.name,
      role <- sys.mode,
      subModules <- sys.components,
      subModules <- sys.interconnection,
      containers <- sys.serviceRegistry,
      primeChannel <- sys.channels  )
}
-- The following rule generates a SystemC module from a factory unit.
rule FactoryUnit2SCModule {
  from
    factUt : MM!FactoryUnit
  to
    scMod : MM1!SC_Module (
      name <- factUt.name,
      process <- factUt.factProc,
      ports <- factUt.ports,
      configuration <- factUt.yieldServices,
      variables <- factUt.properties  )
}
rule ControllerUnit2SCModule {
  from
    ctrlUt : MM!ReconfigControllerUnit
  to
    scMod : MM1!SC_Module (
      name <- ctrlUt.name,
      process <- ctrlUt.ctrlProc,
```

```
58        ports <- ctrlUt.ports
59      )
60  }
61  -- The following rule generates a Sc_Thread from a factory process.
62  rule FactoyProcess2SCThread {
63    from
64      factProc : MM!FactoryProcess
65    to
66      scThread : MM1!SC_Thread (
67        name <- factProc.name,
68        isSpawned <- factProc.isSpawned(),
69        isDynamic <- factProc.isDynamic(),
70        function <- factProc.services,
71        sensitivity <- factProc.associatedEvents  )
72  }
73  rule ReconfigurablePort2TLMInitiatorSocket {
74    from
75      rcfgPort : MM!ReconfigurablePort
76    to
77      tlmInitSocket : MM1!TLM_Initiator_Socket (
78        name <- rcfgPort.name,
79        Dependencies <- rcfgPort.dependenciesPorts
80      )
81  }
```

Listing 4.1 – *Extrait du module de transformation ATL.*

### 4.6.2 Transformation M2T

Désormais, avec une sémantique suffisamment précise du modèle SystemC/TLM précédemment produit, il est possible de générer le code applicatif en transformant les modèles en artefacts textuels. Acceleo, un langage de transformation de modèles basé template est utilisé pour élaborer une transformation M2T, structurée en deux moteurs : l'un génère le code source SystemC/TLM de la plateforme de simulation, et l'autre génère le code C++ de la fabrique selon le pattern Abstract Factory.

Le premier moteur de génération utilise comme entrée le modèle SystemC/TLM précédemment produit, qui est conforme au métamodèle de la figure 4.22. Ce modèle d'entrée est utilisé pour extraire des données et les insérer dans des espaces réservés (placeholders) au niveau du template. Le listing 4.2 illustre la partie du template utilisée pour générer la hiérarchie des objets au sein d'un module SystemC. Le code généré par le deuxième moteur (voir listing 4.3) comprend des descriptions de services et des classes templates associées aux fichiers de code et prêtes à accueillir des fabriques concrètes générées au moment de l'exécution grâce à l'appel de la fonction sc_spawn depuis le rappel end_of_elaboration ou lors d'une simulation.

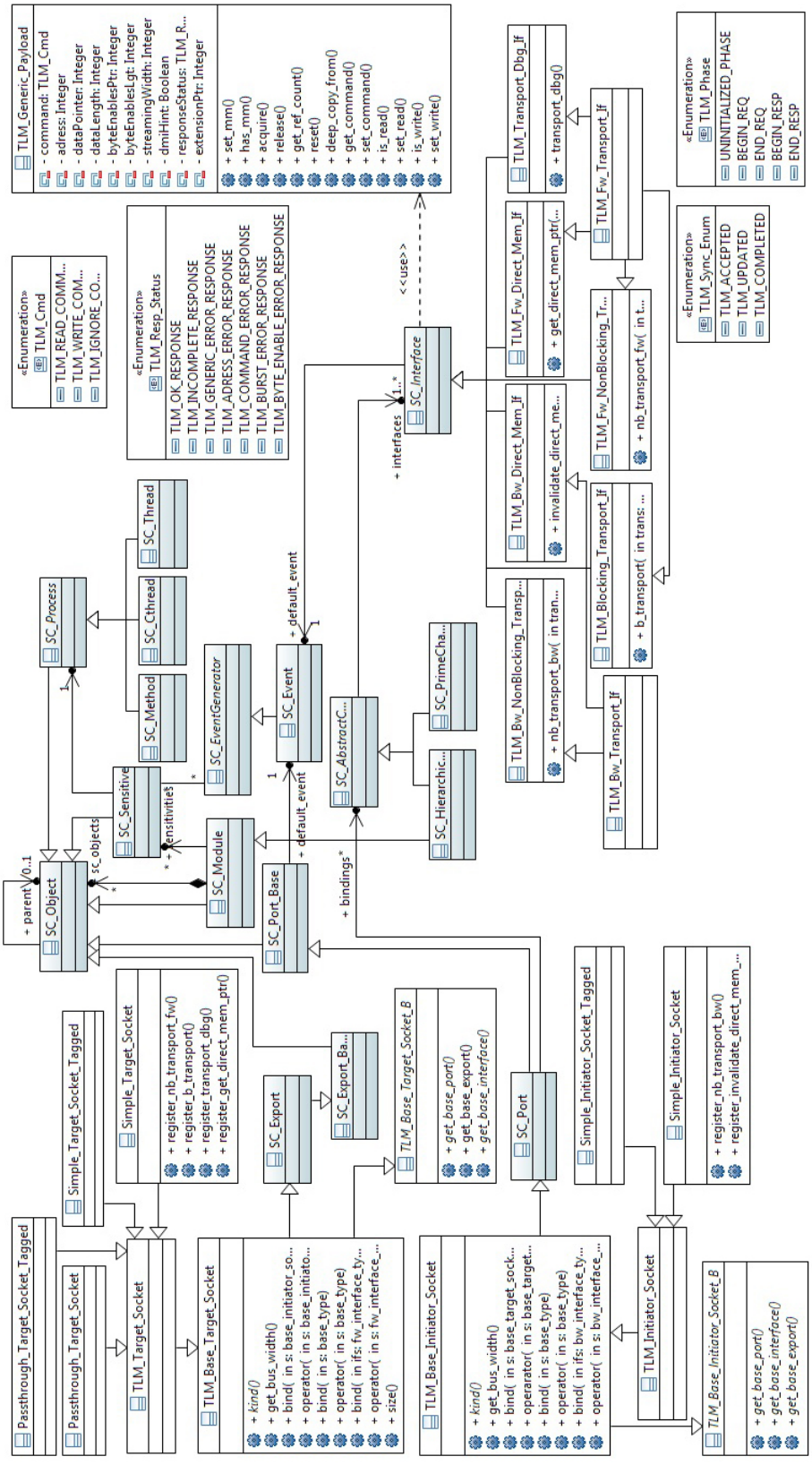



FIGURE 4.22 – *Le métamodèle SystemC/TLM.*

```
SC_MODULE([it.sc_module_factory.name/])
{
[for (itpf :SC_Port | it.sc_module_factory.ports)]
sc_in<[itpf.type/]> [itpf.name/];
[/for]
[for (ite : SC_Event | it.sc_module_factory.ports.event)]
sc_event [ite.name/];
[/for]
void [it.sc_module_factory.threads.function.name/](){ sc_spawn(sc_bind
    (&[it.sc_module_factory.threads.spawn.function.path/]:[it.
    sc_module_factory.threads.spawn.function.name/],this));};
SC_CTOR([it.sc_module_factory.name/]){  SC_THREAD([it.
    sc_module_factory.threads.function.name/]);
sensitive [for (its : SC_Sensitive |  it.sc_module_factory.threads.
    sensitivity)]<<[its.name/][/for];}};
SC_MODULE([it.sc_module_controller.name/]){
[for (itpc : SC_Port | it.sc_module_controller.ports)] sc_in<[itpc.
    type/]> [itpc.name/];
[/for]
[for (itec : SC_Event | it.sc_module_controller.ports.event)]
sc_event [itec.name/];
[/for]};
[for (itf : Function | it.sc_module_controller.threads.function)]
void [itf.name/]([for (itpar : Parameter | itf.parameters)][itpar.type
    /]  [itpar.name/][/for]){
[if (itf.name='publishService')]  [anEClass.containers.name/].
    push_back([itf.parameters.name/]);
[else]
[if (itf.name='deleteService')]
[anEClass.containers.name/]['['/]srvName[']'/].state='Removed';
[/if]
[/if]}
[/for]
SC_CTOR([it.sc_module_controller.name/]){
[for (itth :SC_Thread | it.sc_module_controller.threads)]
SC_THREAD([itth.function.name/]);
sensitive [for (itsen : SC_Sensitive | itth.sensitivity)]<< [itsen.
    name/][/for];
}};
```

Listing 4.2 – *Extrait du template de génération de code SystemC/TLM.*

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/emf/2002/Ecore','http://www.
    afactory.org']
[template public generateElement(anEClass : PatternUseCase)]
[comment @main/]
[file (anEClass.name.concat('.cpp'), false, 'UTF-8')]
class [anEClass.AbstractFactory.name/] // abstract factory
{public:
  [for (it : Operation  |anEClass.AbstractFactory.operation )]
    [it.kind/] [it.reurnType/] *[it.name/]()=0;
  [/for]
}
[for (itcf : ConcreteFactory | anEClass.ConcreteFactory)]
class [itcf.name/] : public [anEClass.AbstractFactory.name/]
{public:
[for (itop : Operation | itcf.operation)]
```

```
16   [itop.reurnType/] *[itop.name/](){ return new [itop.service/](); }
17 [/for]
18 }
19 [/for]
20 [for (itap : AbstractProduct | anEClass.AbstractProduct)]
21 class [itap.name/]
22 {public:
23 [for (itop : Operation | itap.operation)]
24   [itop.kind/] [itop.reurnType/] [itop.name/]()=0;
25 [/for]
26 };
27 [/for]
28 [for (itcp : ConcreteProduct | anEClass.ConcreteProduct)]
29 class [itcp.name/] : public [itcp.parent/]
30 {
31 [for (itpr : Property | itcp.property)]
32   [itpr.type/] [itpr.name/] = [itpr.value/];
33 [/for]
34 [for (itop : Operation | itcp.operation)]
35   [itop.reurnType/] [itop.name/](){}
36 [/for]
37 };
38 [/for]
39 [/file]
40 [/template]
```

Listing 4.3 – *Template de génération de la fabrique en C++.*

## 4.7 Conclusion

Dans ce chapitre, nous avons exposé notre framework dédié à la modélisation et à l'analyse à un haut niveau d'abstraction des SoCs reconfigurables. Ce framework repose sur un processus de transformation progressive des spécifications initiales, exprimées en UML/MARTE étendu, vers des modèles de plus en plus concrets, jusqu'à la génération du code cible en SystemC au niveau transactionnel. Une architecture en couches a été définie, articulant plusieurs niveaux d'abstraction allant du niveau SOCM jusqu'au niveau de modélisation en SystemC. Un flot de conception a également été proposé, intégrant les principes de l'IDM ainsi que les fondements du profil MARTE pour un développement orienté modèle. Par ailleurs, le domaine métier a été spécifié à travers un ensemble de métamodèles couvrant les aspects liés à la plateforme d'exécution, aux composants reconfigurables, au contrôle et à l'analyse de la DPR. Nous avons ensuite introduit des extensions du profil MARTE, incorporant d'une part les concepts liés à la DPR et au SOCM, donnant naissance au profil MARTE4DPR et d'autre part les concepts issus du langage SystemC au niveau TLM ainsi que ceux du patron de conception Abstract Factory, aboutissant aux profils MARTE4SCTLM et MARTE4AF. Ces profils visent à réduire la distance sémantique entre les modèles de simulation et le code généré, en facilitant le raffinement et les transformations successives.

Le chapitre suivant sera consacré à une étude de cas visant à appliquer et à valider notre framework à travers la modélisation et l'analyse d'un système réel.

# 5

# Validation du framework

Sommaire

## 5.1 Introduction

Ce chapitre présente une validation expérimentale de notre framework à travers une étude de cas afin de démontrer sa pertinence et son applicabilité. Nous discutons de la manière dont le framework proposé a été appliqué pour un raffinement progressif des modèles de spécification à différents niveaux d'abstraction. Par ailleurs, un ensemble de métriques d'évaluation est utilisé afin d'apprécier la qualité des transformations de modèles et de quantifier la valeur ajoutée du framework par rapport à une démarche de codage manuel.

## 5.2 Présentation de l'étude de cas

Les filtres adaptatifs jouent un rôle central dans de nombreuses applications de traitement du signal, notamment lorsque le signal à traiter évolue dans le temps ou est affecté par des interférences ou un bruit non stationnaire. Leur pertinence repose sur leur capacité à s'ajuster dynamiquement à l'environnement du système sans intervention manuelle, ce qui en fait des composants critiques dans les systèmes embarqués reconfigurables. Un crossover actif 3-voies (active 3-way crossover) est un dispositif électronique employé dans les systèmes audio pour diviser le spectre audio, qui va d'environ 20 Hz à 20 kHz, en trois bandes de fréquences distinctes : les basses, les moyennes, et les hautes fréquences pour les transmettre aux transducteurs de haut-parleurs spécialement conçus pour ces gammes. L'utilisation d'un crossover est nécessaire en raison de l'impossibilité de créer un transducteur capable de reproduire correctement l'ensemble des dix octaves du spectre audio. Ce défi n'est pas seulement lié aux limitations technologiques des haut-parleurs, mais aussi à des principes physiques fondamentaux. Idéalement, le son d'un haut-parleur émanerait d'un point unique, ce qui garantirait un champ sonore homogène sans effets d'interférence causés par la présence de plusieurs sources ou par la taille même de la source sonore. Un exemple de crossover actif 3-voies est montré dans la figure 5.1.


Figure 5.1 – *Crossover actif 3-voies de Exposure-VXN.*

Au niveau le plus simple, un crossover est un réseau passif qui n'a pas besoin d'une alimentation électrique pour fonctionner, il utilise des composants passifs comme des condensateurs et des inductances et est placé après l'amplification. Les crossovers passifs présentent plusieurs inconvénients. Ils causent des pertes d'énergie, car ils reposent sur des composants comme les résistances, condensateurs et inductances pour filtrer les fréquences, ce qui peut réduire l'efficacité du système audio. Ils génèrent également de la chaleur, ce qui peut nuire aux performances et à la durabilité des haut-parleurs. Leur flexibilité est limitée, car les points de coupure sont fixes, rendant les ajustements difficiles. En outre, ils offrent moins de contrôle sur les bandes de fréquences par rapport

aux crossovers actifs, ce qui limite les possibilités d'optimisation du signal pour des bandes spécifiques, ce qui est souvent nécessaire dans les applications de haute performance. Les crossovers passifs sont placés entre l'amplificateur et les haut-parleurs, ce qui impose une charge variable à l'amplificateur et peut affecter sa performance, surtout à des volumes élevés. En somme, les crossovers passifs sont simples et peu coûteux, mais ils manquent de flexibilité et de précision comparés aux crossovers actifs, ce qui peut limiter la qualité audio dans des systèmes exigeants.

A l'inverse, un crossover actif est situé avant les amplificateurs de puissance, ce qui offre plusieurs avantages en termes de performance et de contrôle. Les crossovers actifs offrent une grande liberté de conception, permettant de créer des pentes de filtrage abruptes et des corrections simples pour chaque haut-parleur sans augmenter les coûts ni ajouter de lourds composants. Ils évitent les problèmes liés aux charges d'impédance et aux pertes de puissance des crossovers passifs, tout en réduisant la distorsion, le besoin en résistances dissipatives, et les effets de compression thermique. Leur conception élimine l'utilisation d'inductances, réduisant ainsi les problèmes de distorsion et de couplage magnétique. Les crossovers actifs permettent également d'ajuster précisément les niveaux sonores, d'ajouter facilement des délais pour compenser les différences de centres acoustiques, et même de modéliser la chaleur de la bobine pour compenser les pertes de puissance. De plus, leur stabilité et leur faible distorsion les rendent idéaux pour des applications exigeantes, comme les systèmes de renforcement sonore et les enceintes de référence.

Afin d'illustrer concrètement notre approche et de démontrer son applicabilité, nous avons choisi de mener une étude de cas portant sur la modélisation et l'analyse d'un crossover actif 3-voies dynamiquement reconfigurable sur FPGA. Comme illustré dans la figure 5.2, un crossover représente un réseau qui effectue le filtrage des signaux audio en fonction de leur fréquence, dirigeant les sons de basse fréquence vers un woofer, les sons de fréquence moyenne vers un haut-parleur médium et les sons de haute fréquence vers un tweeter. L'intégration de ces trois haut-parleurs vise à obtenir la meilleure qualité sonore sur l'ensemble du spectre audio.

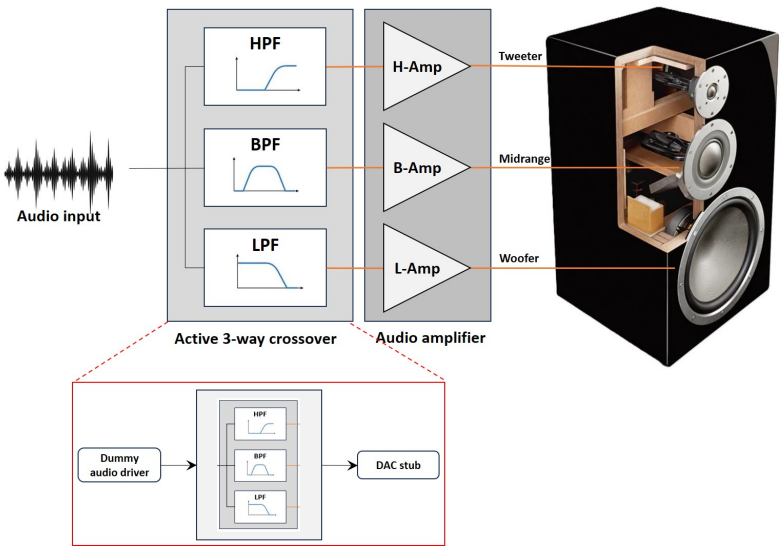



FIGURE 5.2 – *Utilisation du crossover actif 3-voies.*

Un crossover actif 3-voies peut être réalisé au moyen de trois types de filtres : un filtre passe-haut (High-Pass Filter - HPF) permet le passage des fréquences supérieures à une certaine fréquence de coupure tout en atténuant les fréquences inférieures ; un filtre passe-bande (Band-Pass Filter - BPF) permet de laisser passer les signaux dont

les fréquences se situent entre une fréquence inférieure (fréquence de coupure basse) et une fréquence supérieure (fréquence de coupure haute). Les signaux en dehors de cette plage de fréquences sont atténués, c'est-à-dire qu'ils sont soit réduits en amplitude, soit complètement bloqués. Enfin, un filtre passe-bas (Low-Pass Filter - LPF) permet le passage des signaux dont les fréquences sont inférieures à une certaine fréquence de coupure, tout en atténuant ou bloquant les fréquences supérieures.

Les filtres FIR (Finite Impulse Response) sont couramment employés dans de nombreuses applications de traitement du signal numérique (Digital Signal Processing - DSP) grâce à leur stabilité, leur capacité à maintenir la phase, leur flexibilité de conception et leur réponse en fréquence précise. Ces attributs les rendent particulièrement adaptés pour le filtrage audio, le traitement d'images, les systèmes de communication et bien d'autres domaines [Roy et Chandra, 2021], [Esakkirajan et al., 2024].

Le filtre N-Tap FIR a été utilisé pour implémenter des filtres passe-haut, passe-bande et passe-bas du crossover. Un FIR se caractérise par le nombre de prises, les fréquences de transition et la fréquence d'échantillonnage du signal audio à filtrer. La relation entre le signal d'entrée et la sortie d'un filtre FIR N-Tap est décrite par l'équation suivante :

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] \tag{5.1}$$

Où h[k] sont les coefficients de la fonction de transfert du filtre, N est le nombre de coefficients du filtre, x[n-k] représentent les échantillons d'entrée et y[n] est le signal de sortie. Chaque échantillon d'entrée est multiplié par un coefficient, puis les produits résultants sont additionnés pour donner un échantillon de sortie. En adéquation avec l'équation 5.1, la figure 5.3 montre la structure logique typique du filtre du N-Tap FIR.

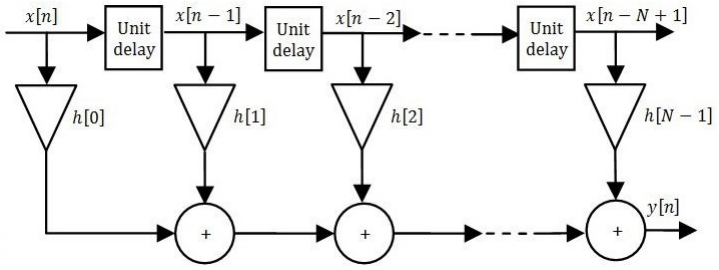



FIGURE 5.3 – *Structure logique du filtre N-Tap FIR.*

La capacité de multiplexage temporel dynamique du matériel sur un même FPGA permet de réduire la taille, le poids, la consommation d'énergie et le coût de développement du crossover en optimisant l'utilisation des ressources matérielles. Telle que montré sur la figure 5.4, la reconfigurabilité du filtre FIR, le dynamisme et la possibilité de substitution des services peuvent être envisagés à deux niveaux distincts :

— Etant donné que les différents types de filtres ont la même structure avec des comportements différents, il est possible de les implémenter via un composant reconfigurable (fournisseur de services) capable de substituer dynamiquement la fonctionnalité lorsque cela est nécessaire.
— Pour répondre aux exigences de performance et de consommation d'énergie, il est possible de modifier le nombre de taps du filtre. Un filtre à N-taps peut être remplacé par un filtre à M-taps du même type. Cela se traduit dans le matériel par une reconfiguration partielle du filtre initial.

Les services fournis sous forme de bitstreams partiels (.bit) sont dynamiquement chargés dans des blocs reconfigurables donnant lieu à chaque fois à une reconfiguration dynamique et partielle sans que cela n'affecte la logique statique du FPGA. Des

exemples de descriptions de services correspondant à des configurations IPs possibles avec des informations relatives à l'utilisation des ressources sont présentés dans le tableau 5.1. Ces descriptions sont enregistrées dans un registre de services et peuvent être exploitées à divers niveaux d'abstraction. Le type du filtre, le nombre de coefficients, la fréquence d'échantillonnage et la fréquence de transition sont par exemple utilisés à un niveau d'abstraction élevé tandis que le nombre de Luts, de canaux, de registres et de DSP slices sont utiles à des niveaux moins abstraits. Bien que nous ciblions une plateforme SystemC/TLM en partant de modèles de haut niveau d'abstraction, toutes les informations disponibles dès les premières étapes de la modélisation et relatives aux niveaux d'abstraction inférieurs doivent être intégrées progressivement à mesure que les modèles sont affinés. Ces informations pourront, par exemple, être utilisées pour la génération automatique de code HDL de bas niveau, contribuant ainsi à compléter notre chaîne de transformation.

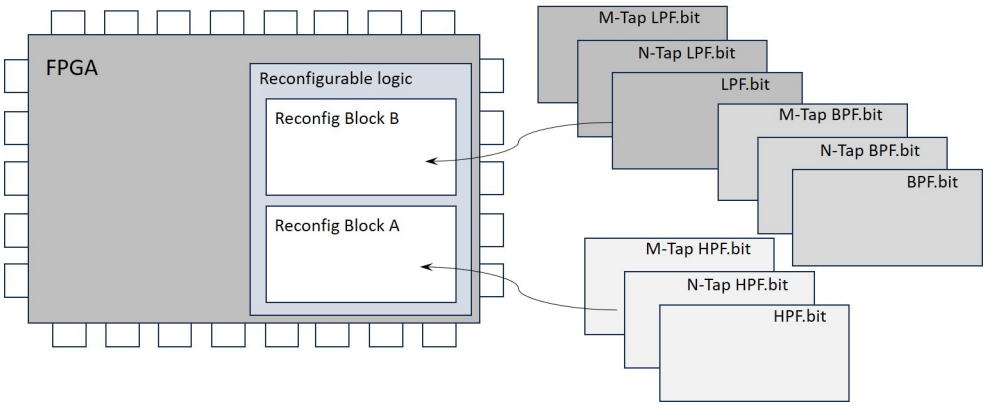



FIGURE 5.4 – *Implémentation du crossover sur FPGA.*

TABLE 5.1 – *Exemples de descriptions de services et des ressources utilisées. Service : Type de filtre, R1 : Nombre de coefficients, R2 : Fréquence d'échantillonnage, R3 : Fréquence de transition R4 : Nombre de multiplicateurs, R5 : Nombre de canaux, R6 : Nombre de Luts, R7 : Nombre de mémoires EBRs, R8 : Nombre de registres et R9 : Nombre de slices DSP.*

| Configuration | Service | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 |
|---|---|---|---|---|---|---|---|---|---|---|
| #1 | LPF | 24 | 44.1 | 10.0 | 6 | 1 | 241 | 4 | 272 | 1 |
| #2 | LPF | 32 | 44.1 | 11.0 | 8 | 1 | 201 | 4 | 303 | 9 |
| #3 | HPF | 32 | 44.1 | 4.0 | 32 | 1 | 202 | 2 | 199 | 6 |
| #4 | HPF | 64 | 44.1 | 5.0 | 1 | 4 | 242 | 3 | 306 | 6 |
| #5 | BPF | 48 | 44.1 | 4.0 | 12 | 1 | 246 | 4 | 281 | 1 |
| #6 | BPF | 48 | 44.1 | 12.0 | 12 | 1 | 246 | 4 | 281 | 1 |

## 5.3 Modélisation du système

En suivant notre flot de conception présenté dans la section 4.3 et afin d'exploiter nos profils d'annotation MARTE4DPR, MARTE4SCTLM et MARTE4AF, il est nécessaire d'appliquer d'abord le profil MARTE aux différents modèles, ce qui rend ensuite ses concepts disponibles pour utilisation.

Principalement, le profil MARTE s'articule autour de deux préoccupations : modéliser les fonctionnalités des systèmes temps réel et embarqués, et annoter les modèles d'application pour faciliter l'analyse des propriétés du système. En appliquant le sous-profil SAM à un modèle avec les données appropriées, il devient possible de prédire

certaines propriétés essentielles à une conception en utilisant des méthodes d'analyse spécifiques au domaine.

### 5.3.1 Modèle de l'application

Pour modéliser les structures internes et les interactions au sein du crossover, nous avons opté pour le diagramme de structure composite UML. Les profils MARTE4DPR et MARTE4SCTLM peuvent être appliqués sur tous les diagrammes UML structurels. Cependant, pour des raisons de flexibilité, nous avons utilisé le diagramme de structure composite pour décrire la structure interne des classificateurs et leurs interactions avec l'environnement, ce qui convient à la fois à la modélisation matérielle et au développement d'une architecture à base de composants orientés services.

Comme le montre la figure 5.5, d'un point de vue structurel, le crossover actif 3-voies simulé est composé de plusieurs parties. Principalement, deux instances du filtre FIR dynamiquement et partiellement reconfigurables, appelées *Filter#1* et *Filter#2* et stéréotypées « ReconfigurableUnit ». *Filter#1* incorpore deux sous-composants *Controller#1* et *Factory#1* stéréotypés respectivement « ReconfigControllerUnit » et « FactoryUnit ». Le sous-composant *Factory#1* offre un service de filtre FIR ainsi que ses implémentations concrètes, à savoir les produits *32TapHPF* et *64TapHPF*. Pour gérer l'exécution de son service généré *64TapHPF*, Factory#1 intègre *FactProc#1*, une ressource ordonnançable dynamiquement stéréotypée *FactoryProcess*. Le sous-composant *controller#1* est responsable de la gestion du processus de reconfiguration conformément à des protocoles spécifiques liés aux modes de reconfiguration du filtre FIR et aux états de connexion des ports reconfigurables. Il englobe une ressource ordonnançable dynamiquement appelée CtrlPro#1 et stéréotypée « ReconfigControllerProcess », permettant de lancer un processus de reconfiguration une fois que l'événement *rcfgRqst* a été déclenché et que les dépendances du service ont été satisfaites. De plus, *CtrlPro#1* fournit des services spécifiques pour gérer la publication, la découverte, l'activation, la reprise et la passivation des services. Les processus embarqués dans *Filter#1* utilisent des appels de méthodes d'interfaces pour communiquer à travers des ports/exports reconfigurables désignés par *p1* et *xp1* et stéréotypés « ReconfigurablePort » et « ReconfigurableExport » respectivement. Les services peuvent être publiés et découverts au moyen d'une unité passive protégée et centralisée appelée *Catalog* et stéréotypée « ServiceRegistry ». Les unités temps réel *DummyAudioDriver* et *DACStub* émulent respectivement les comportements du pilote audio et de l'unité de conversion numérique-analogique.

Pour éviter de surcharger le modèle, nous avons délibérément omis de représenter les détails du second filtre. Il est important de préciser que ce dernier, nommé *Filter#2*, est structurellement identique à *Filter#1*, mais se comporte différemment en offrant d'autres services. Le service de transport de base est fourni par le composant *Interconnect* qui représente une abstraction du bus logique et est stéréotypé « **CommunicationMedia** » (du package GRM). Les composants peuvent ainsi accéder au service de transport par le biais de ports de communication. Lorsque les éléments du modèle sont raffinés en appliquant des stéréotypes, il devient nécessaire d'attribuer des valeurs à leurs propriétés. La figure 5.6 présente les boîtes de dialogue Papyrus qui permettent d'assigner des valeurs aux caractéristiques des services publiés dans le registre de services *Catalog*.

Le comportement modal d'un composant reconfigurable peut être modélisé par un diagramme d'états UML stéréotypé « **ModeBehavior** » (du package MARTE_Foundations : : CoreElements) où chaque mode représente une configuration particulière et qui est connecté à d'autres modes via des transitions tel que illustré par la figure 5.7. Une transition de mode peut se produire en réponse à un déclencheur lié

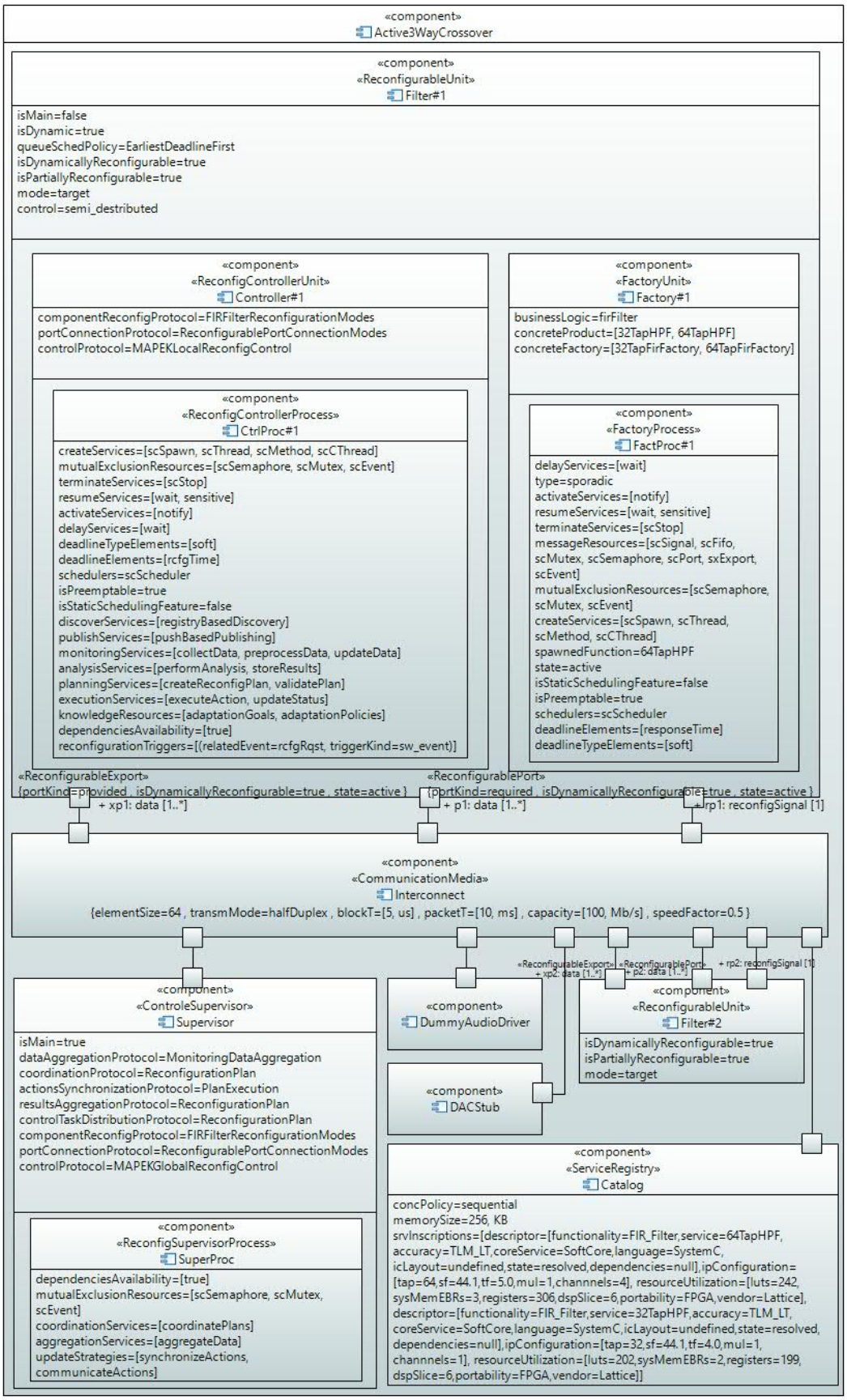



FIGURE 5.5 – *Vue abstraite de haut niveau du système.*

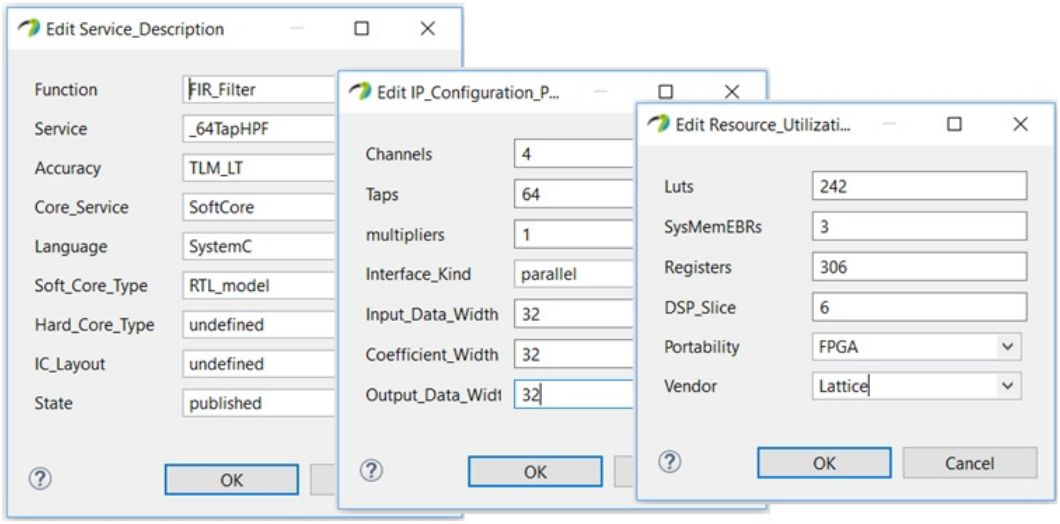



FIGURE 5.6 – *Assignation de valeurs aux attributs du service.*

à un événement qui vérifie les conditions de reconfiguration. Un filtre peut fonctionner en trois modes mutuellement exclusifs : *HighPassFilter*, *LowPassFilter* et *BandPassFilter*, stéréotypés « **Mode** ». Une reconfiguration est déclenchée dès que la condition associée à chaque mode est remplie, la transition est stéréotypée « **ModeTransition** ». Un filtre peut être implémenté en tant que filtre à *N-Tap* ou de *M-Tap*, en fonction du niveau de performance requis et peut être commuté dynamiquement d'un mode à un autre.

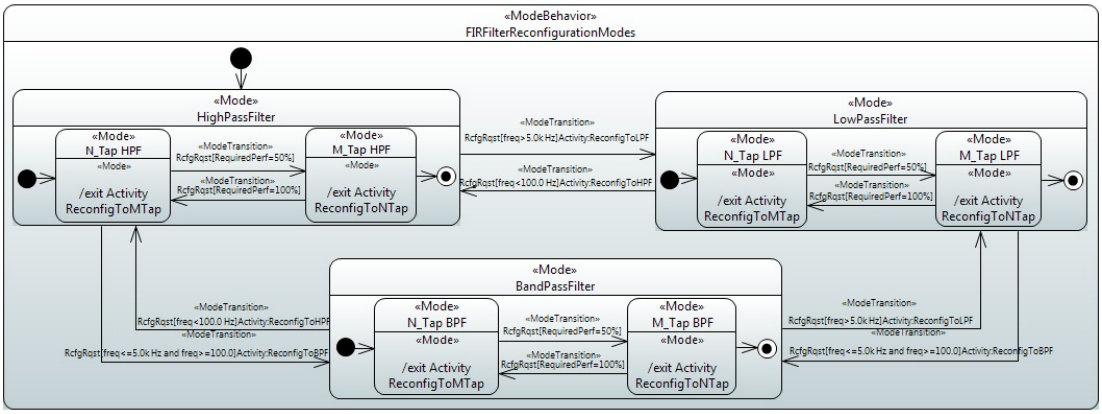



FIGURE 5.7 – *Reconfiguration structurelle/comportementale du filtre.*

### 5.3.2 Modèles de plateformes d'exécution et de mapping

Dans MARTE, la modélisation d'une plateforme matérielle repose sur deux vues complémentaires : une vue logique, qui catégorise les ressources matérielles en fonction de leur rôle fonctionnel et des services qu'elles offrent ou demandent ; et une vue physique, qui se concentre sur leurs caractéristiques physiques, notamment la forme, les dimensions, l'emplacement, la consommation énergétique et la dissipation thermique. Comme ces deux vues partagent de nombreux concepts, elles peuvent être fusionnées en une vue unifiée.

Comme illustré dans la Figure 5.8, le composant *Active3WayCrossover* est raffiné en un composant plus spécialisé en appliquant les stéréotypes « **HwComponent** » (du

package HRM) et « FPGA ». Le composant est une carte (*kind = card*) dotée de propriétés physiques telles que les dimensions cartésiennes (*dimensions = [50.0,62,0]*), la grille rectiligne associée (*grid = [3,3]*) et les exigences environnementales *r_Conditions* telles qu'une température variant dans l'intervalle [0°C,60°C]. Le stéréotype « FPGA » spécifie une architecture reconfigurable à granularité fine avec une capacité de relocalisation à l'exécution modélisées respectivement par l'attribut *granularity = fine_grained* et *isRuntimeReloc = true*. La DPR est réalisé à l'aide d'un flot de conception basé sur des partitions (*designFlow = EAPR*) via l'interface ICAP *(configInterface = ICAP)*. La distribution spatiale des régions reconfigurables est définie par l'attribut multivalué *regions*. Le composant est annoté par le nombre de Luts (*nbLUTs = 9604*), les entrées d'un Lut (*nbLUT_Inputs* = 6) et les Flip-Flops (*nbFlipFlops = 810*). Les blocs de calcul intégrés et la plage de fréquences prises en charge sont également spécifiés par les attributs (*blocksComputing* et *op_Frequencies*) respectivement. Le filtre *Filter#1* est stéréotypé « ReconfigHwComponent » ; il s'agit d'une puce (*kind = chip*) dynamiquement et partiellement reconfigurable (*isDynamicallyReconfig = true* et *isPartiallyReconfig = true*), caractérisée par ses propriétés de disposition et de consommation d'énergie, en l'occurrence *position = [2,2 1,1]*, sa consommation statique *staticConsumption = 5 W* et sa dissipation thermique statique *staticDissipation = 3 W*. Le sous-composant *Factory#1* est stéréotypé « **HwComputingResource** » et « *reconfigHwComponent* » désignant une ressource d'exécution active dynamiquement reconfigurable (*isDynamicallyReconfig = true*), opérant à une fréquence d'horloge *frequency = 300* Mhz.

Les stéréotypes « **HwComputingResource** » (du package GRM) et « **HwComponent** » (du package HRM) sont appliqués au composant *Controller#1* afin de capturer les capacités de traitement des ressources au sein d'un conteneur. Le composant *Catalog* est modélisé à l'aide des stéréotypes « **HwMemory** » (du package GRM) et « **HwComponent** » pour décrire une ressource de stockage de données fournissant des services de lecture et d'écriture. Le composant *Interconnect* est stéréotypé « **HwBus** » (du package GRM) et « **HwComponent** » ; il représente un support matériel spécifique, un canal en l'occurrence (*kind* = channel) caractérisé par la largeur d'adressage *adressWidth = 64 bits* et la largeur du mot de transfert *wordWidth = 64 bits*, ses propriétés temporelles *isSynchronous = true* et son mode de transmission *transmMode = halfDuplex*.

Il est intéressant de noter que dans MARTE, les valeurs étiquetés des stéréotypes peuvent changer en fonction du degré de raffinement des modèles ; en effet, la fréquence d'horloge d'un composant matérielle initialisée à une valeur maximale de 300 Mhz peut être réévaluée à 150 Mhz représentant la fréquence actuellement utilisée à un niveau plus détaillé de la modélisation. Les valeurs de certains attributs tels que la puissance consommée *staticConsumption* et la dissipation thermique *staticDissipation* dépendant fortement des caractéristiques spécifiques de chaque FPGA peuvent être estimées avec des outils d'analyse de consommation fournis par les fabricants de FPGA comme Xilinx Power Estimator (XPE) [1] ou Early Power Estimators (EPE) et Power Analyzer d'Intel [2].

### 5.3.3 Modèle de simulation SystemC/TLM

A partir du modèle d'allocation précédent, un modèle de simulation SystemC/TLM est obtenu soit manuellement en appliquant le profil MARTE4SCTLM, soit automatiquement par transformation de modèles, comme illustré dans la figure 5.9. Le composant de niveau supérieur *Active3WayCrossover* est stéréotypé « Sc_Module », son attribut *isMain = true* signifie que le module sera instancié dans la fonction *sc_main*. Pour

1. https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/power-efficiency/power-estimator.html
2. https://www.intel.com/content/www/us/en/support/programmable/support-resources/power/pow-powerplay.html

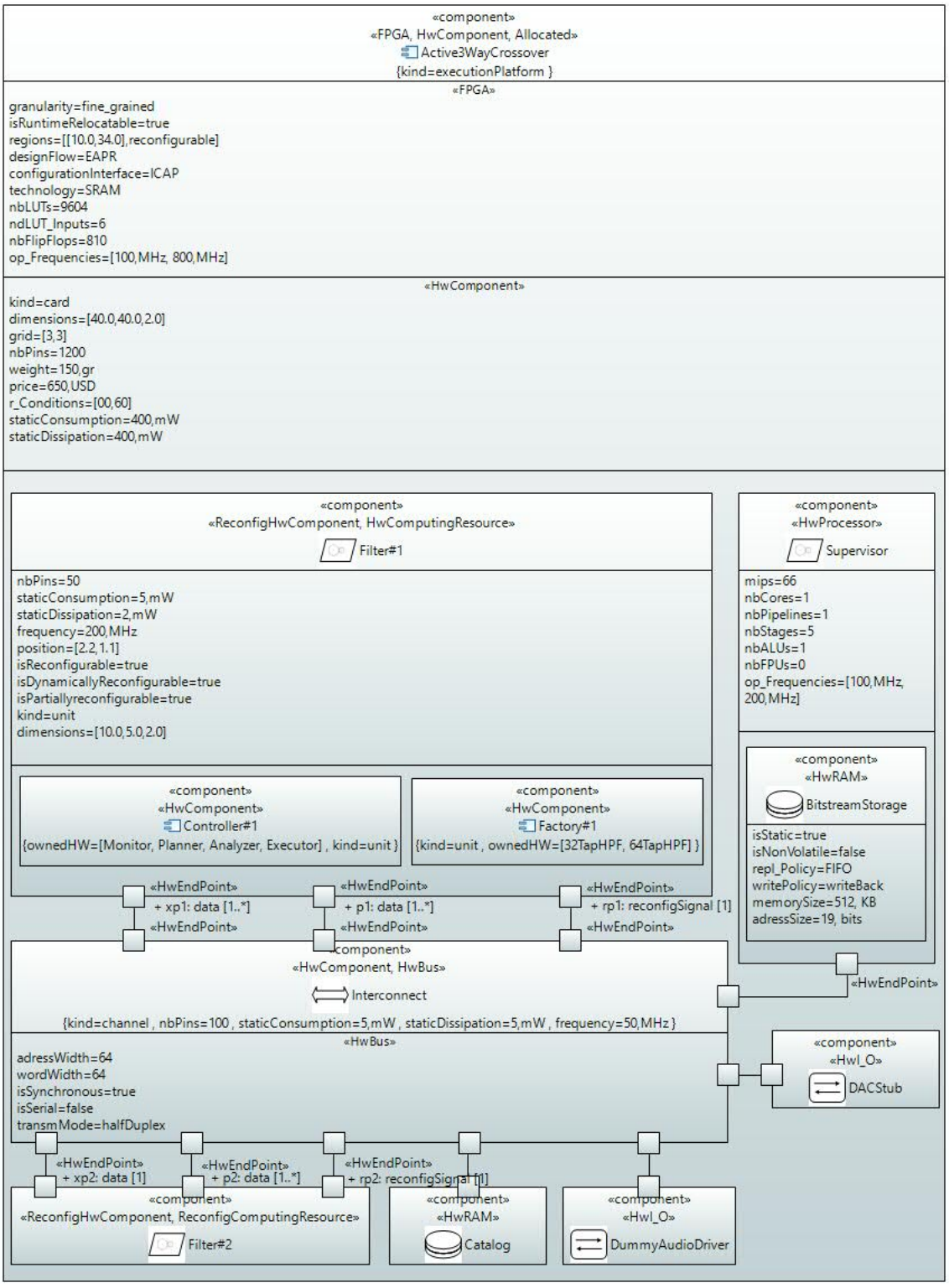



FIGURE 5.8 – *Vues logique et physique fusionnées.*

établir la hiérarchie des modules, le stéréotype « Sc_Module » est appliqué au composant *Filter#1* ce qui permet de créer un sous-module dans le module parent. Son attribut *role = target* spécifiant que le module agira comme une cible dans la modélisation TLM. Les attributs *variables* et *helpers* font partie des membres de données et des fonctions membres devant être déclarés au sein du *Sc_Module*. Les composants imbri-

qués *Factory#1* et *Controller#1* sont également stéréotypés « Sc_Module ». Le premier contient une fonction membre spéciale appelée *64TapHPF* stéréotypée « Sc_Spawned » et « Sc_Thread » pour gérer la création d'une instance de processus engendré appelée à partir d'un processus thread. La second contient une fonction membre stéréotypée « Sc_Thread » pour mapper le service de contrôle. La fonction *64TapHPF* est générée dynamiquement à partir du thread *FactProc#1* et peut être substituée par la fonction *32TapHPF* en fonction de la liste de sensibilité. Les ports et exports sont respectivement stéréotypés « Simple_Initiator_Socket  et « Simple_Target_Socket » pour prendre en charge les chemins aller et retour correspondant à une séquence d'appels de méthode. Une fois le stéréotype « TLM_Fw_Transport_If » est appliqué, ses attributs doivent être explicitement initialisés, ce qui permet de capturer les informations pertinentes pour une future génération automatique de code, comme indiqué dans la figure 5.10. De la même manière que le composant *DummyAudioDriver*, le composant *DACStub* est stéréotypé « Sc_Module » et joue le rôle d'initiateur (*role = initiator*). Enfin, le composant *Catalog* est stéréotypé « Sc_Module » pour spécifier une structure de données assurant des fonctions de gestion et d'accès.

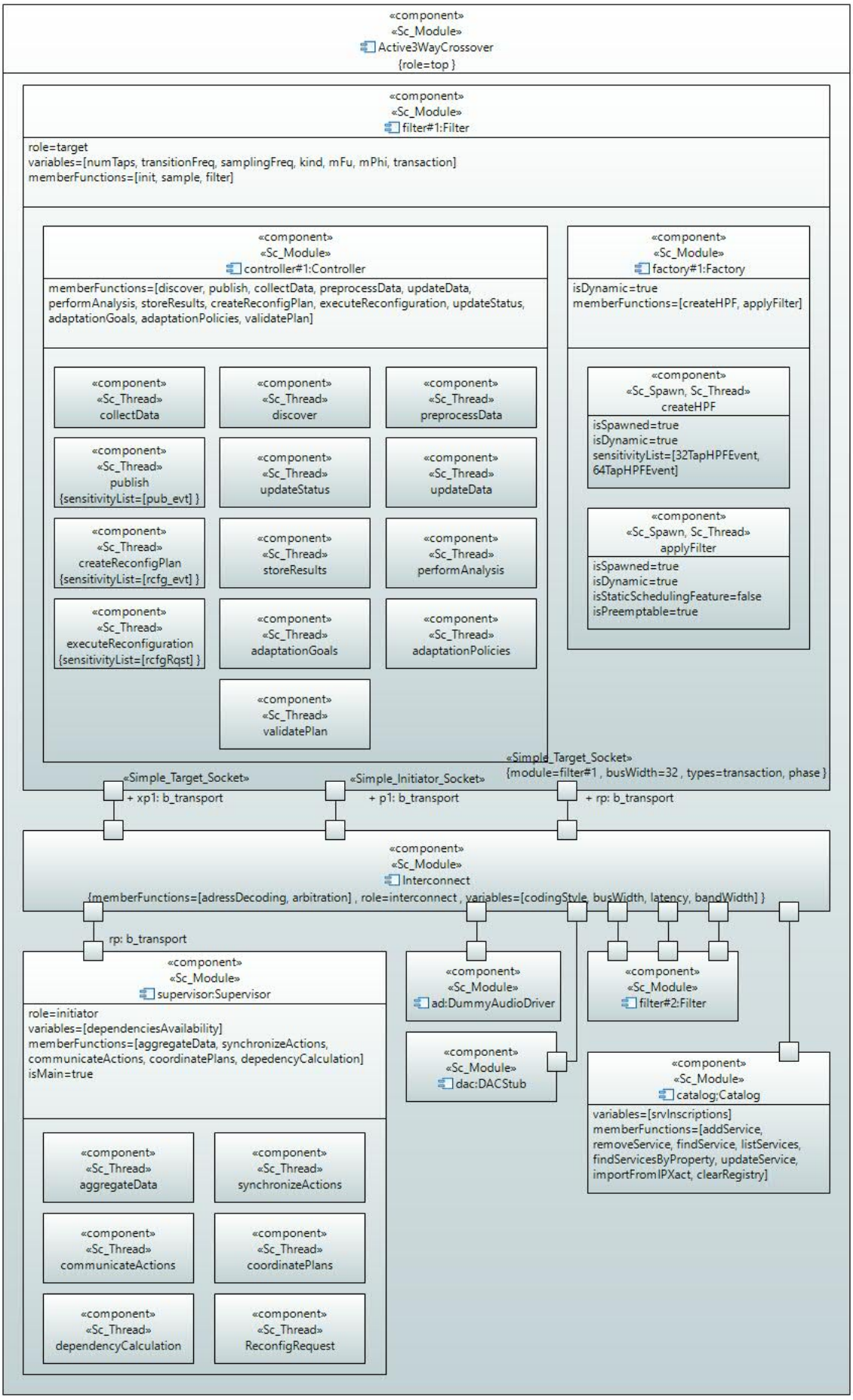



FIGURE 5.9 – *Le modèle de simulation SystemC/TLM.*

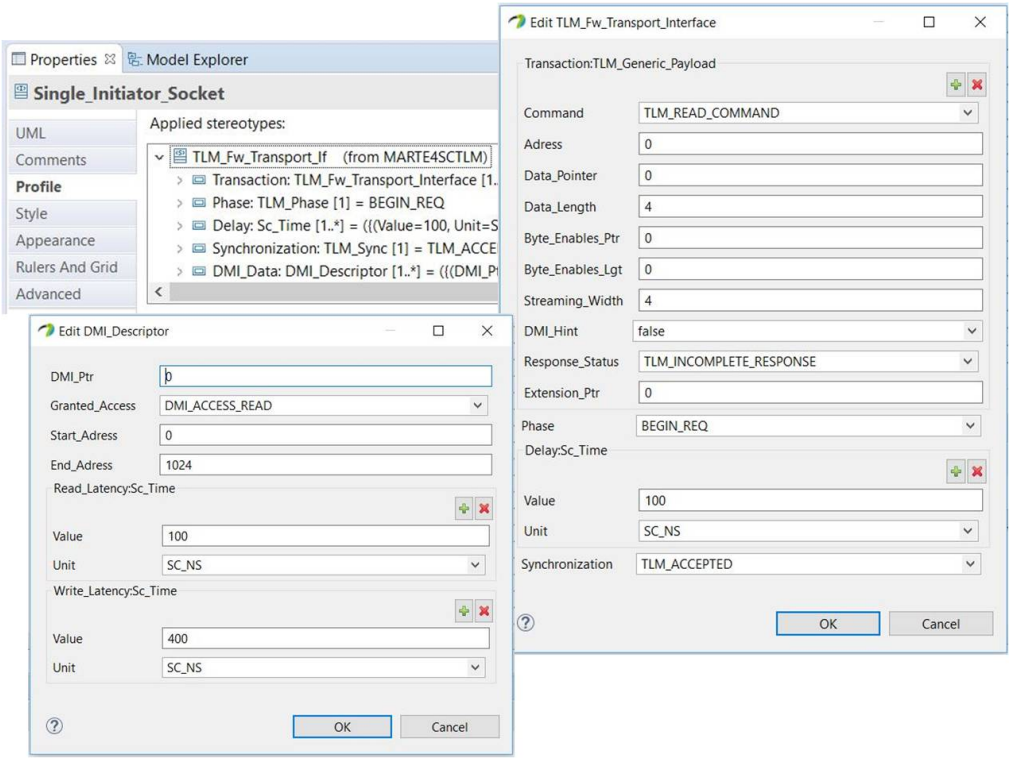



FIGURE 5.10 – *Assignation de valeurs aux attributs du stéréotype TLM_Fw_Transport_If.*

Du fait que l'objectif principal de la modélisation des plateformes est souvent d'évaluer leur capacité à supporter une application, il est généralement nécessaire de les modéliser au niveau des instances plutôt qu'au niveau des classes. Une classe représente une généralisation de ses instances, en mettant en évidence les éléments communs tout en abstrayant leurs spécificités. Un modèle basé sur les instances, en raison de son haut degré de spécificité, demeure statique et ne s'applique qu'à des scénarios particuliers à un moment donné. Cette limitation empêche sa traduction directe en code générique, ce qui le rend peu adapté à la génération automatique de code. Ainsi, les outils de modélisation UML utilisent souvent le diagramme de classes comme point de départ pour générer du code réutilisable. L'application des extensions MARTE4SCTLM et MARTE4AF au diagramme de classes permet de réduire davantage la distance sémantique entre le modèle de simulation SystemC/TLM et le code à générer. Comme le montre la figure 5.11, un filtre est composé de plusieurs classes actives modélisant d'une part les fabriques concrètes stéréotypées « ConcreteFactoty » et les produits concrets stéréotypés « ConcreteProduct » et d'autre part le contrôleur de reconfiguration local. Le superviseur est également modélisé par une classe active et utilise des classes passives (Transaction et Catalog) pour le transfert de données et la gestion des services.

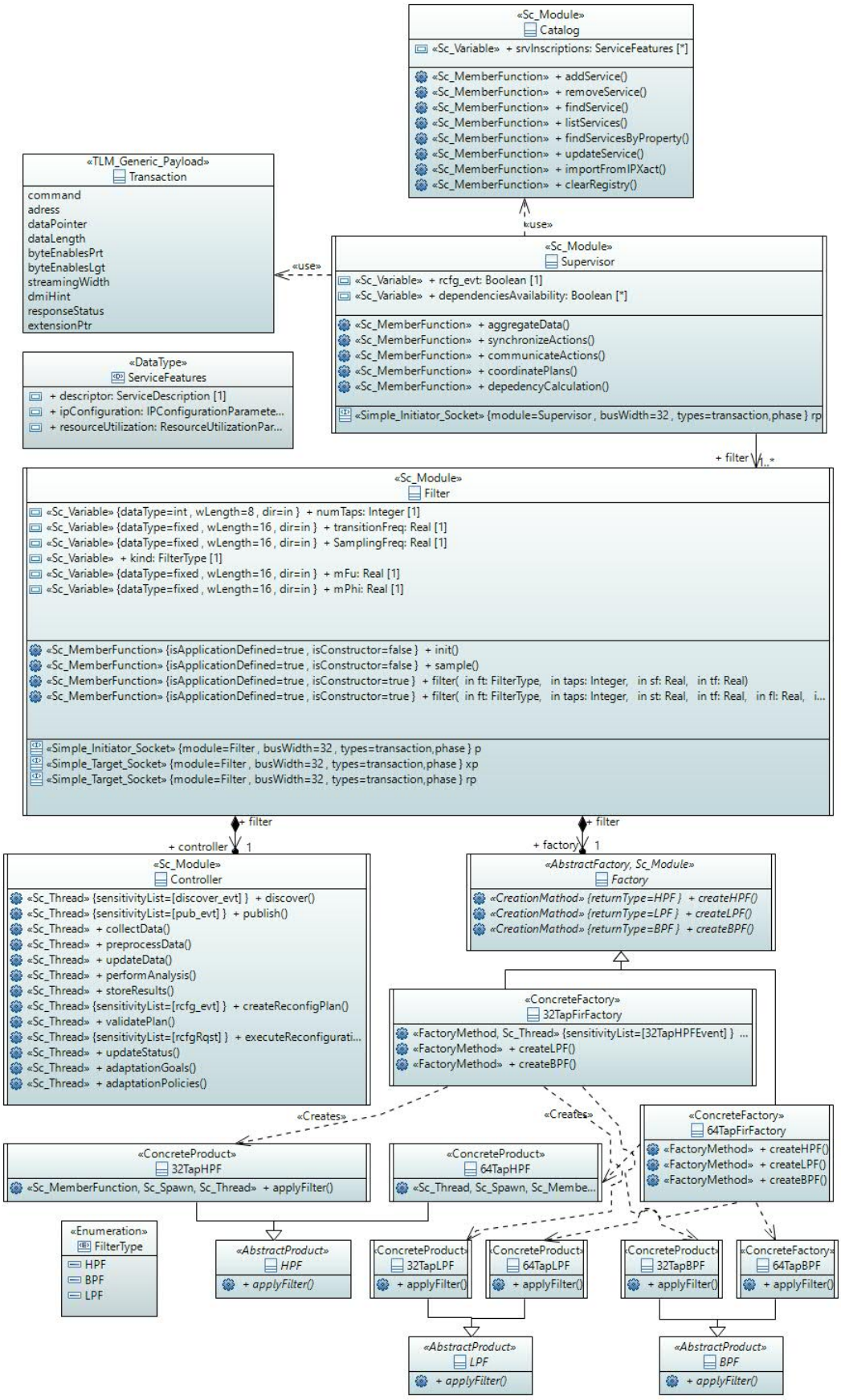



FIGURE 5.11 – *Le modèle de simulation SystemC/TLM d'un filtre pour la génération automatique de code.*

### 5.3.4 Génération automatique de code

À la différence de certains langages tels qu'AADL, qui bénéficie d'outils dédiés comme OSATE (Open Source AADL Tool Environment), MARTE n'offre pas de support natif pour la génération automatique de code et repose sur l'utilisation d'extensions personnalisées adaptées à des cibles spécifiques. Comme il a été mentionné dans la section 4.6, la génération automatique de code est un processus de transformation/raffinement de modèles visant un langage ou une plateforme cible, mis en œuvre par une chaine de compilation. La première transformation que nous avons réalisée prend en entrée un modèle d'allocation conforme au métamodèle MARTE4DPR et le transforme en un modèle cible conforme au métamodèle SystemC/TLM. Nous avons utilisé Eclipse Modeling Framework (EMF) pour la création et l'instanciation de métamodèles au format ecore. Dans la figure 5.12, une partie de la hiérarchie du métamodèle MARTE4DPR avec une instance de modèle conforme sont affichés en utilisant EMF Tree-based Editor.

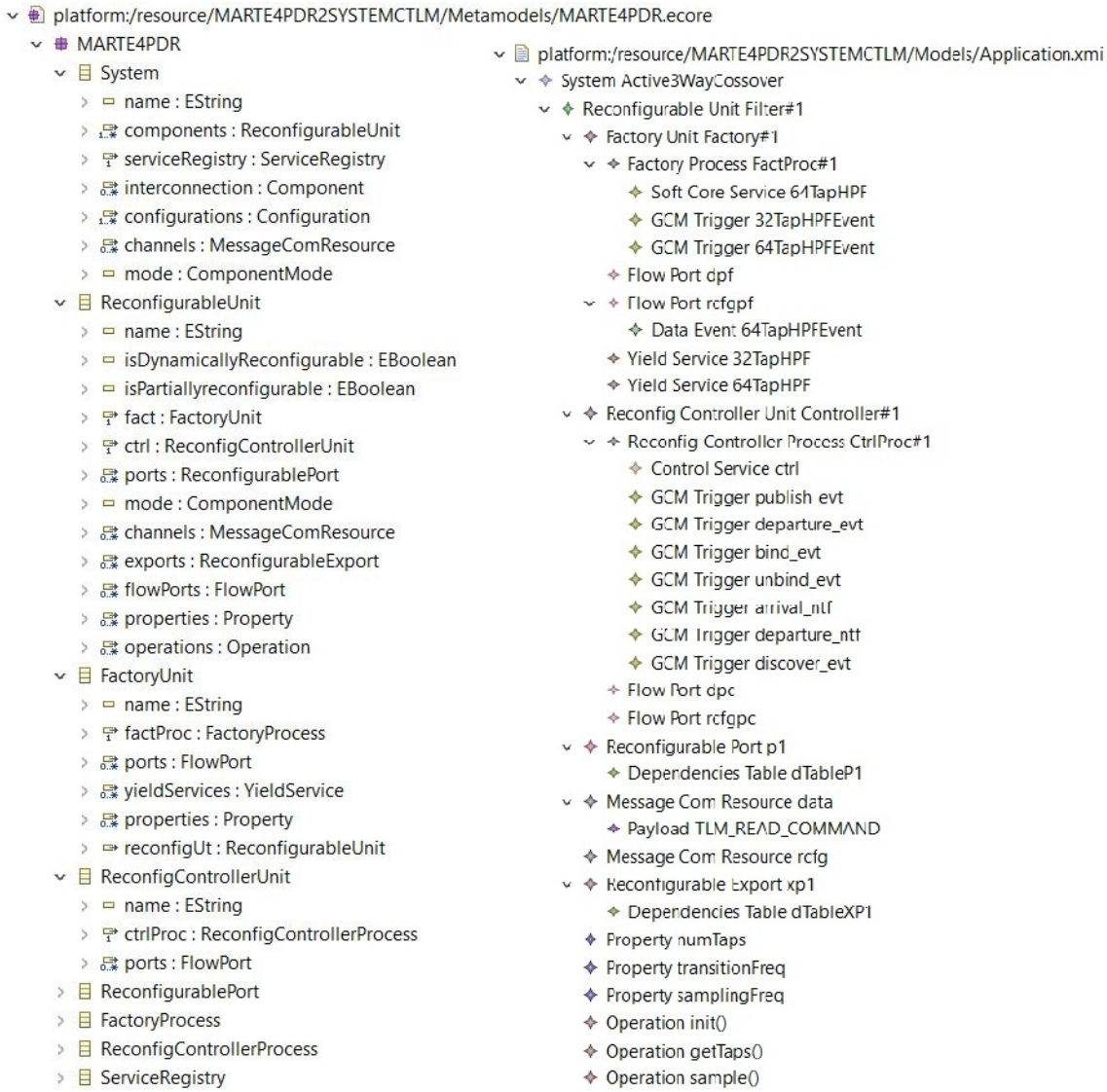



FIGURE 5.12 – *Le métamodèle de MARTE4DPR (à gauche) et le modèle de l'application (à droite).*

Le listing 5.1 montre une partie du modèle SystemC/TLM généré, sérialisé au format xmi, dont la sémantique est clairement définie afin de garantir une meilleure qualité du code à produire ultérieurement.

```
<subModules name="Filter#1">
      <helpers name="init()"/>
      <helpers name="getTaps()"/>
      <helpers name="sample()"/>
      <variables name="numTaps" type="Int" value="64"/>
      <variables name="transitionFreq" type="Float" value="3.0"/>
      <variables name="samplingFreq" type="Float" value="44.1"/>
      <initiatorSocket name="p1">
        <Dependencies dependencyComponent="DummyAudioDriver"
           remotePort="pDUM"      connectionState="Connected"/>
      </initiatorSocket>
      <targetSocket name="xp1">
        <Dependencies dependencyComponent="DACStub" remotePort="pDAC"
             connectionState="Connected"/>
      </targetSocket>
      <primeChannel name="data">
        <payload id="2" command="TLM_READ_COMMAND" adress="32"
           dataPointer="32"   dataLength="4" streamingWidth="4"
           responseStatus="TLM_OK_RESPONSE"  dmiHint="false"/>
      </primeChannel>
      <primeChannel name="rcfg"/>
      <sc_module_factory name="Factory#1">
        <process xsi:type="SYSTEMC_TLM:SC_Thread" name="FactProc#1"
           isSpawned="true"  isDynamic="true">
          <function name="64TapHPF"/>
          <sensitivity name="32TapHPFEvent"/>
          <sensitivity name="64TapHPFEvent"/>
        </process>
        <ports name="dpf"/>
        <ports name="rcfgpf">
          <event name="64TapHPFEvent"/>
        </ports>
        <configuration yieldService="32TapHPF"/>
        <configuration yieldService="64TapHPF"/>
      </sc_module_factory>
      <sc_module_controller name="Controller#1">
        <process xsi:type="SYSTEMC_TLM:SC_Thread" name="CtrlProc#1">
          <function name="ctrl"/>
          <sensitivity name="publish_evt"/>
          <sensitivity name="departure_evt"/>
          <sensitivity name="bind_evt"/>
          <sensitivity name="unbind_evt"/>
          <sensitivity name="arrival_ntf"/>
          <sensitivity name="departure_ntf"/>
        </process>
        <ports name="dpc"/>
        <ports name="rcfgpc"/>
      </sc_module_controller>
    </subModules>
```

Listing 5.1 – *Extrait du modèle SystemC/TLM généré en xmi.*

Afin de simuler la reconfiguration dynamique du filtre FIR, nous devrions pouvoir utiliser une fabrique de filtres appropriée , *32TapFirFactory* ou *64TapFirFactory*, sans avoir à connaitre quelle implémentation du filtre *32TapHPF* ou *64TapHPF* sera effectivement utilisée. Le métamodèle de l'abstract factory et une instance de modèle conforme pré-

sentés dans la figure 5.13 sont utilisés comme entrées pour le moteur de transformation Acceleo employant le template présenté dans le listing 5.2.


FIGURE 5.13 – *Le métamodèle de l'Abstract Factory (à gauche) et le modèle instancié du filtre FIR (à droite).*

```
1  [comment encoding = UTF-8 /]
2  [module generate('http://www.eclipse.org/emf/2002/Ecore','http://www.
       afactory.org']
3  [template public generateElement(anEClass : PatternUseCase)]
4  [comment @main/]
5  [file (anEClass.name.concat('.cpp'), false, 'UTF-8')]
6  class [anEClass.AbstractFactory.name/] // abstract factory
7  {public:
8    [for (it : Operation  |anEClass.AbstractFactory.operation )]
9      [it.kind/] [it.reurnType/] *[it.name/]()=0;
10   [/for]
11 }
```

```
[for (itcf : ConcreteFactory | anEClass.ConcreteFactory)]
class [itcf.name/] : public [anEClass.AbstractFactory.name/]
{public:
[for (itop : Operation | itcf.operation)]
  [itop.reurnType/] *[itop.name/](){ return new [itop.service/](); }
[/for]
}
[/for]
[for (itap : AbstractProduct | anEClass.AbstractProduct)]
class [itap.name/]
{public:
[for (itop : Operation | itap.operation)]
  [itop.kind/] [itop.reurnType/] [itop.name/]()=0;
[/for]
};
[/for]
[for (itcp : ConcreteProduct | anEClass.ConcreteProduct)]
class [itcp.name/] : public [itcp.parent/]
{
[for (itpr : Property | itcp.property)]
  [itpr.type/] [itpr.name/] = [itpr.value/];
[/for]
[for (itop : Operation | itcp.operation)]
  [itop.reurnType/] [itop.name/](){}
[/for]
};
[/for]
[/file]
[/template]
```

Listing 5.2 – *Le template de l'Abstract Factory.*

### 5.3.5 Analyse d'ordonnançabilité basée modèles

Pour mener une analyse d'ordonnançabilité, il est nécessaire de définir des modèles adaptés, dérivés du modèle de conception et en parfaite cohérence avec celui-ci. Notre analyse porte sur le transfert des bitstreams de la mémoire externe vers la mémoire de reconfiguration, identifié comme la principale source de surcharge liée à la DPR. Selon le modèle de coût présenté dans la section 4.4.3.3, les bitstreams se propagent via un chemin de reconfiguration RP composé de deux chemins élémentaires EP1 et EP2. le premier concerne le transfert des bitstreams de la mémoire externe vers la mémoire de stockage du superviseur et le second de la mémoire de stockage du superviseur à la mémoire locale du contrôleur de reconfiguration. Il est essentiel de rappeler que dans MARTE, l'analyse repose sur l'idée que le contexte d'analyse définit une ou plusieurs combinaisons de modèles de plateforme et de workloads. Dans la suite, nous présentons ces éléments ciblant le niveau transactionnel TLM et servant d'entrée aux outils d'analyse.

#### 5.3.5.1 Modèle de comportement du workload

Un **workload** est constitué d'un ou de plusieurs **end-to-end flows**, obtenus en affinant les interactions dans les deux chemins élémentaires EP1 et EP2, comme illustré dans la figure 5.14. Le stéréotype « **GaWorkloadEvent** » est appliqué au message ReconfigEvent, il permet de spécifier le pattern d'arrivée du message déclenchant le processus

de reconfiguration. Dans ce cas, un pattern d'arrivée apériodique est défini. Le stéréotype « **SaEndtoEndFlow** » est appliqué à l'activité consistant à transférer les bitstreams à travers EP1 et EP2 respectivement. Les valeur des variables à calculer *isSched* et *schSlack* indiquant respectivement si le message transmis respecte toutes ses échéances et un pourcentage de variation du temps d'exécution de toutes les étapes s'exécutant sur cet hôte, pouvant être augmenté (valeurs positives) ou réduit (valeurs négatives), afin d'atteindre la limite de l'ordonnaçabilité.

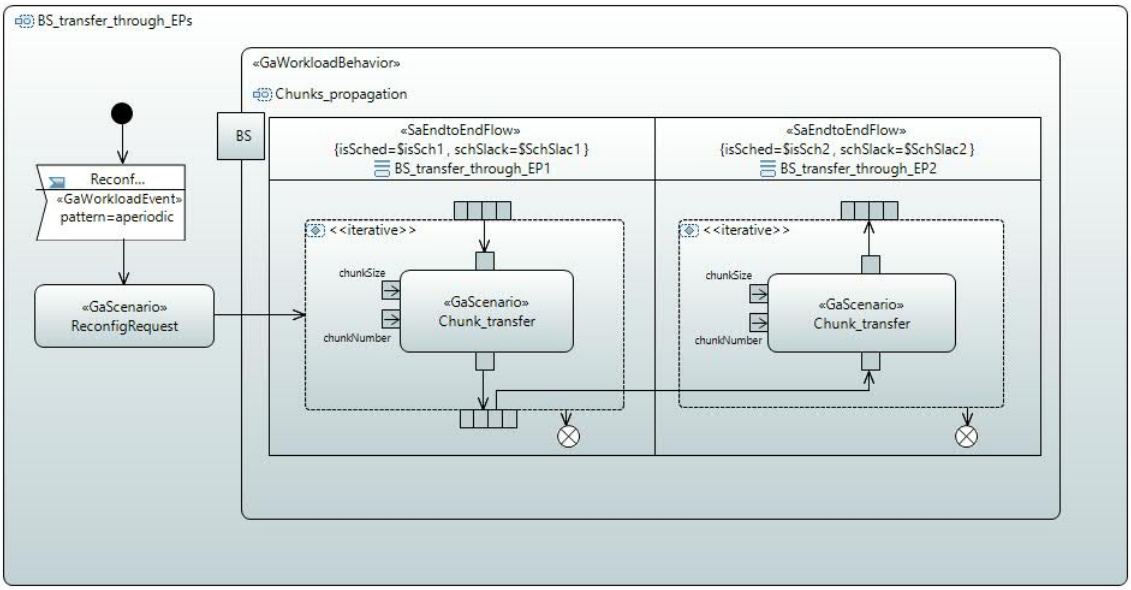



FIGURE 5.14 – *Modèle de scénario comportemental pour le transfert du bitstream via les chemins élémentaire EPs.*

Selon le style de codage AT de TLM, plus adapté à l'analyse, le comportement annoté « **GaScenario** » et les interactions au sein de l'EP1 sont détaillés dans le diagramme de séquence de la figure 5.15. En appliquant le stéréotype « SaCommStep » aux étapes de comportement individuel, le taille du message est spécifié dans l'attribut *msgSize* et les temps d'exécution au pire cas et au meilleur cas sont donnés par l'attribut multivalué *execTime*.

### 5.3.5.2 Modèle de plateforme de ressources pour l'analyse

D'un point de vue analytique, un classificateur structuré peut être employé pour rassembler l'ensemble des instances de ressources de la plateforme à analyser. Tel que montré dans la figure , les ressources de traitement Supervisor stéréotypé « **SaExecHost** » , Controller stéréotypé « **SaExecHost** » et Bus stéréotypé « **SaCommHost** » sont intégrées en tant que composants de cette plateforme et annotés avec des caractéristiques non fonctionnelles essentielles à l'analyse d'ordonnançabilité. L'ensemble des instances de ressources ordonnançables stéréotypées « **SchedulableResource** » est modélisé sous forme de composants alloués aux ressources de traitement avec des niveaux de priorité fixe. Les hôtes d'exécution Supervisor et Controller disposent de ressources partagées, en l'occurrence, BistreamStorage et LocalMemory, respectivement. Les ressources partagées stéréotypées « **SaSharedResource** » sont allouées dynamiquement aux ressources ordonnançables selon une politique d'accès FIFO. La couche logique de communication connectant les ressources ordonnançables est modélisée par les deux canaux TlmBwTransport et TlmFwTransport stéréotypés « **GaCommChannel** » et alloués au hôte de communication Bus.

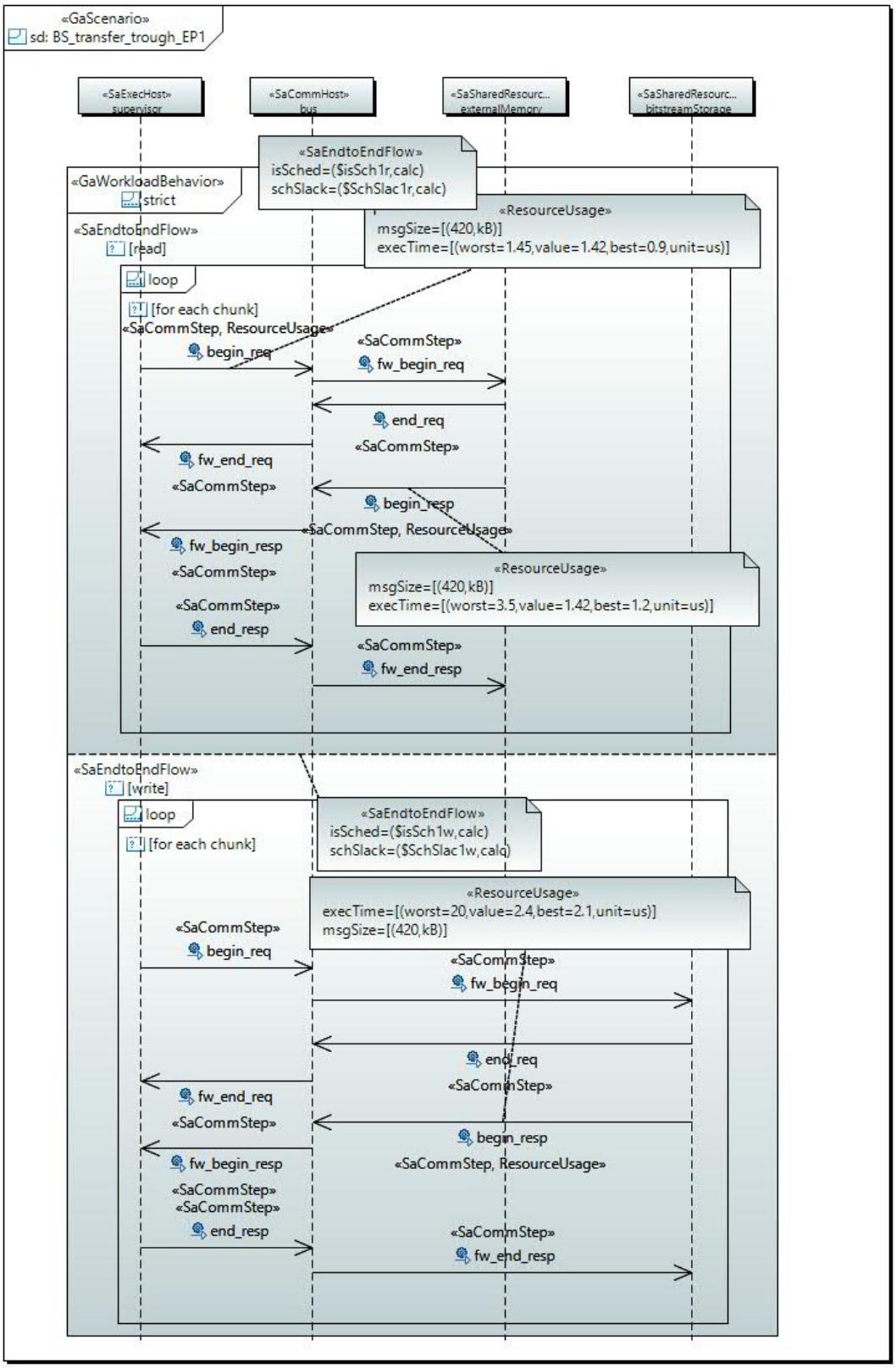



FIGURE 5.15 – *Modèle de scénario comportemental pour le transfert du bitstream via le chemin élémentaire EP1.*

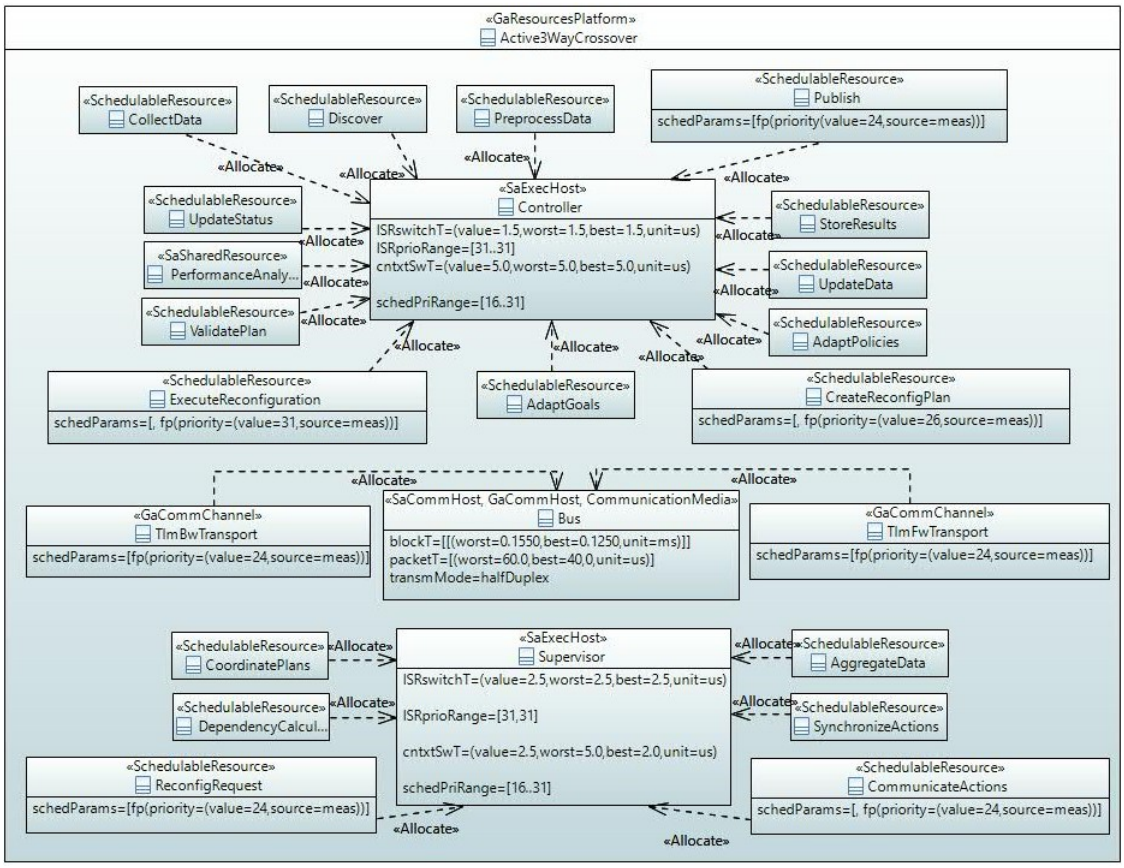



FIGURE 5.16 – *Modèle de la plateforme de ressources pour l'analyse.*

#### 5.3.5.3 Spécification du contexte d'analyse

Un contexte d'analyse constitue le point de départ pour rassembler des données quantitatives pertinentes en vue de réaliser un scénario d'analyse spécifique. En s'appuyant sur ce contexte et ses éléments, un outil d'analyse peut extraire les informations requises pour effectuer l'analyse du modèle. Comme le montre la figure 5.17, notre analyse de contexte permet de vérifier si un ordonnancement respecte toutes les échéances strictes des différentes tâches dans un comportement de workload, d'où *cptCriterion = meetHardDeadlines*. L'attribut *isSched_System* indique si toutes les contraintes temporelles définies pour le contexte d'analyse sont respectées. L'attribut *wcdprt* représente le temps au plus mauvais cas de la DPR. Tous ces attributs sont stéréotypés « var » et correspondent à des valeurs calculées.

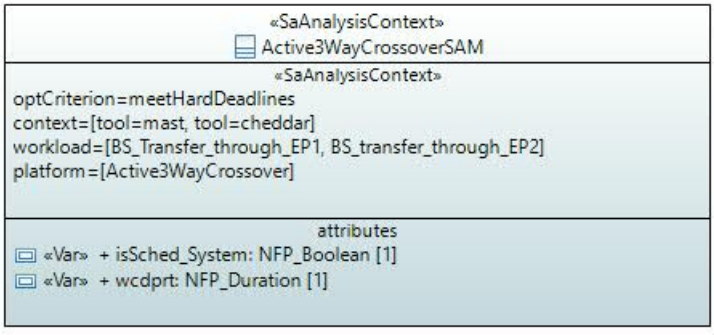



FIGURE 5.17 – *Spécification du contexte de l'analyse d'ordonnançabilité.*

#### 5.3.5.4 Résultats d'analyse

Les modèles de workloads et de plateforme de ressources élaborés précédemment sont transformés automatiquement en des fichiers xml analysables par les outils Mast et Cheddar. Les éléments décrits précédemment pour une analyse d'ordonnnançabilité dans MARTE sont repris dans Mast. Le transfert des bitstreams via EP1 et EP2 correspond à deux transactions dans Mast. Chaque transaction est déclenchée par un ou plusieurs événements externes et regroupe un ensemble d'activités exécutées au sein du système. Ces activités peuvent générer des événements internes à la transaction, susceptibles d'activer d'autres activités en cascade. Les résultats de l'évaluation des temps de réponse dans le pire cas suivant l'algorithme Earliest Deadline First (EDF) sont donnés dans le tableau 5.2. De plus, une analyse de sensibilité basée sur le calcul des marges de temps (Slack Times) met en évidence des valeurs positives, indiquant l'augmentation maximale possible des temps d'exécution des transactions BS_transfer_through_EP1 et BS_transfer_through_EP2 sans compromettre l'ordonnançabilité du système.

Table 5.2 – *Les résultats de l'analyse sous Mast.*

| **Transaction** | **WC Response times ($\mu s$)** | **Slack (%)** |
|---|---|---|
| BS_transfer_through_EP1 | 332.276 | 180.47 |
| BS_transfer_through_EP2 | 248.121 | 266.41 |

De la même manière, les concepts SAM de MARTE trouvent leurs équivalents sémantiques dans Cheddar grâce aux éléments processeur, tâche et ressource. La figure 5.18 présente les résultats de la simulation de l'ordonnancement en appliquant un ordonnanceur de type Least Laxity First (LLF) qui attribue les priorités aux tâches sporadiques en fonction de leur laxisme. Ces tâches s'exécutent sur deux processeur monocœur Supervisor et Controller. Il en ressort que les pires temps de réponse calculés par simulation sur l'intervalle de faisabilité confirment que les échéances des différentes tâches sont respectées pour les deux processeurs.

## 5.4 Analyse des performances du framework

L'évaluation d'un framework de modélisation s'appuie sur divers critères permettant de juger son efficacité, sa pertinence et sa convivialité pour répondre aux exigences des utilisateurs et des systèmes visés. Ces critères doivent être ajustés en fonction des exigences spécifiques du projet ou de l'organisation afin de choisir le framework de modélisation approprié.

Le framework proposé s'appuie sur les avantages offerts par l'écosystème Eclipse, en tirant parti de ses outils de modélisation, de transformation et de génération de code. Cette intégration propose une expressivité graphique avancée facilitant la compréhension, l'édition et la validation des modèles complexes. Elle facilite l'interopérabilité avec des standards industriels et permet une réutilisation et une extensibilité naturelle via les technologies comme EMF et Papyrus. Avec l'ajout de plus de 50 nouveaux stéréotypes pour intégrer les concepts DPR manquants et les constructions SystemC/TLM dans MARTE, les extensions proposées visent à assurer une complétude du profil tout en minimisant le surcoût de modélisation. Papyrus propose un support exhaustif pour définir et appliquer des incréments conformes au standard UML/MARTE, facilitant ainsi la réutilisation des outils disponibles et des connaissances existantes, tout en réduisant les efforts d'apprentissage et les coûts associés.

Pour évaluer l'efficacité du framework proposé en termes d'artefacts générés, une analyse comparative est réalisée entre la génération automatique de code et l'implémentation manuelle du crossover.

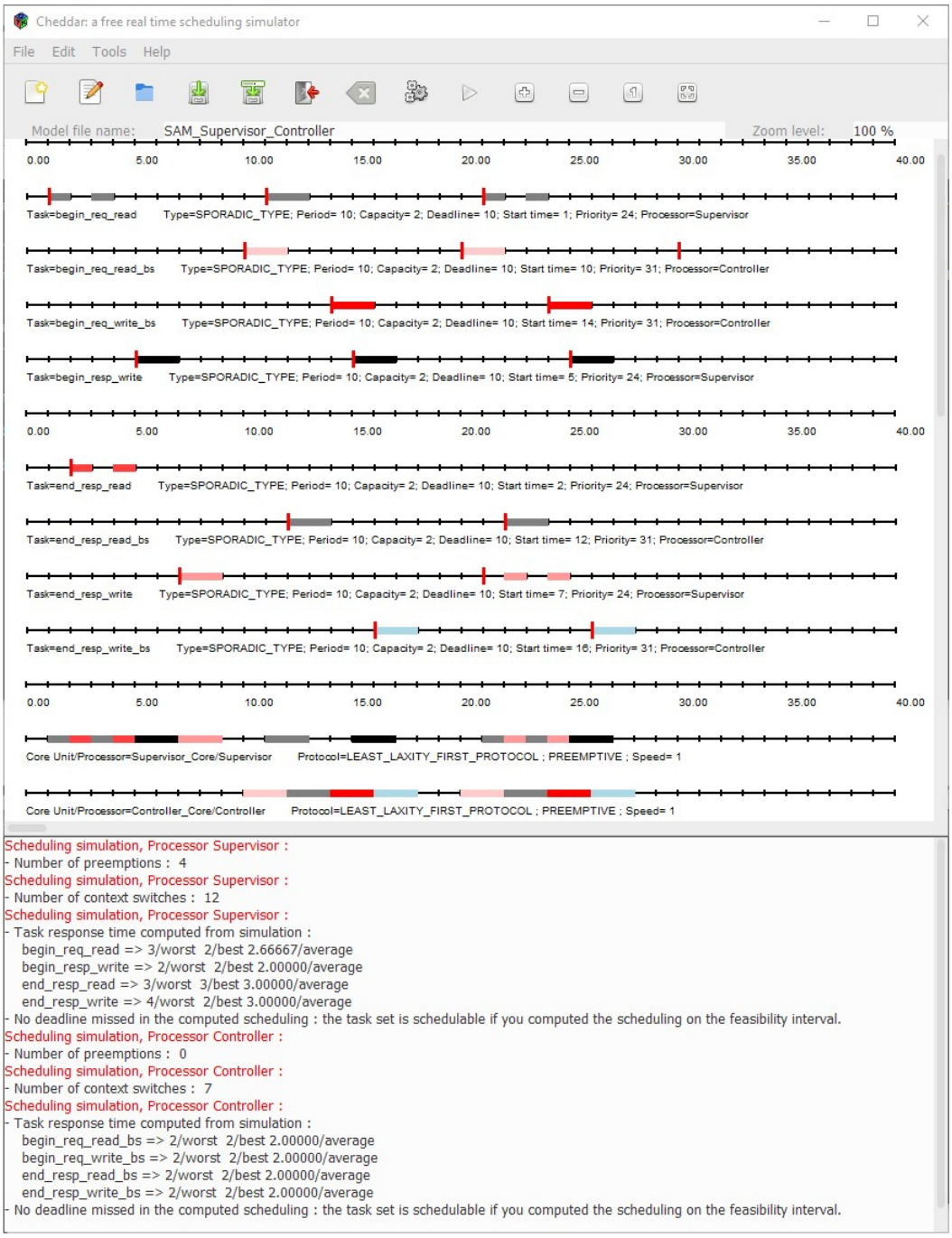



FIGURE 5.18 – *Résultats de l'analyse d'ordonnançabilité en utilisant l'outil Cheddar.*

Les tests de performance ont été effectués sur un processeur Intel(R) Core (TM) i5-8265U à 1,6 GHz, équipé de 4 Go de RAM et fonctionnant sous Windows 10.

Le tableau 5.3 présente une évaluation quantitative de l'implémentation manuelle du système, réalisée à l'aide de l'outil LocMetrics [3].

Le système développé totalise 1256 lignes de code réparties sur 6 fichiers sources, nécessitant un effort de développement estimé à 6,041 mois-personne. Une complexité cyclomatique de 117 indique que le code est particulièrement complexe à tester et à

3. www.locmetrics.com

maintenir. En outre, le codage manuel est réputé pour être un processus long, fastidieux et sujet aux erreurs.

TABLE 5.3 – *Les résultats mesurés de l'implémentation manuelle.*

| **Metric** | **Value** |
|---|---|
| Source Files | 6 |
| Directories | 2 |
| LOC, Lines of Code | 1256 |
| BLOC, Blank Lines | 91 |
| SLOC-P, Physical Executable Lines of Code | 1024 |
| SLOC-L, Logical Executable Lines of Code | 759 |
| McCabe VG Complexity | 117 |
| C&SLOC, Code and Comment Lines of Code | 79 |
| CLOC, Comment Only Lines of Code | 141 |
| CWORD, Commentary Words | 1272 |
| HCLOC, Header Comment Lines of Code | 12 |
| HCWORD, Header Commentary Words | 27 |

Pour évaluer la qualité interne des transformations de modèles ATL, nous nous sommes appuyés sur un ensemble de métriques spécifiques définies à partir des attributs de qualité présentés dans [van Amstel et al., 2011] et [Vignaga, 2009]. Ces derniers incluent notamment la compréhensibilité, la modifiabilité, la réutilisabilité, la modularité, l'exhaustivité, la cohérence et la complexité.

Le tableau 5.4 montre que la transformation repose sur seulement 286 lignes de code, ce qui fait augmenter sa compréhensibilité, sa modifiabilité et réduit sa complexité.

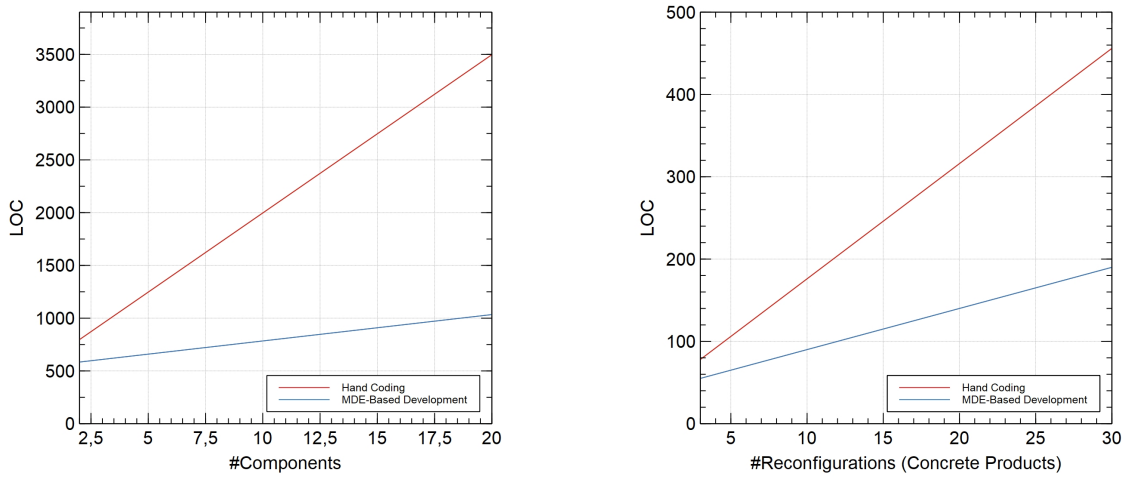
La transformation comprend 26 règles appariées, chacune reliée à un élément d'entrée, ainsi que 2 helpers dotés d'un code de faible complexité. Les métamodèles d'entrée et de sortie affichent des taux de couverture élevés : 96,29 % et 100 % respectivement, ce qui témoigne de la complétude de cette transformation de modèles.

La transformation est définie dans un style déclaratif qui dissimule les détails de l'encodage des relations entre les motifs source et cible, favorisant ainsi une meilleure compréhensibilité et modifiabilité des algorithmes de transformation.

TABLE 5.4 – *L'évaluation des transformations ATL en fonctions des attributs de qualité :A1 : Understandability, A2 : Modifiability, A3 : Reusability, A4 : Modularity, A5 : Completeness, A6 : Consistency, A7 : Complexity. + : Positive effect, - : Negative effect.*

| Metric | Value | A1 | A2 | A3 | A4 | A5 | A6 | A7 |
|---|---|---|---|---|---|---|---|---|
| # Lines of Code | 286 | + | + | | | | | - |
| # Lines of Comments | 8 | + | | | | | | |
| Balance of a Unit | 1 | + | | + | + | | | |
| # Matched Rules | 26 | + | + | | | | | - |
| # Lazy Matched Rules | 0 | | | | | | | |
| # Called Rules | 0 | | | | | | | |
| Average number of Bindings per Rule | 3.42 | + | + | | | - | + | |
| # Rules with a Filter Condition on an Input Pattern | 1 | + | + | | | | | |
| Rule Complexity Increase | 0.69 | + | | | | | | - |
| # Helpers | 2 | + | + | - | | | | |
| # Calls to OCL Functions | 1 | + | + | | | | | |
| Average Helper Cyclomatic complexity | 1 | + | | | | | | |
| # Input Models | 1 | | + | + | | | | - |
| # Output Models | 1 | | + | + | | | | |
| Input Metamodel Coverage | 96.29 | | | | | + | | + |
| Output Metamodel Coverage | 100 | | | | | + | | + |
| Transformation Approach | Declarative | + | + | | | | | - |

Comme le montre la figure 5.19, les résultats suggèrent que notre approche réduit significativement le nombre de LOC nécessaires, diminuant ainsi l'effort de dévelop-




(a) *La relation entre le nombre de LOC et le nombre de composants.*

(b) *La relation entre le nombre de LOC et le nombre de reconfigurations.*

FIGURE 5.19 – *Analyse du nombre de LOC.*

pement comparé au codage manuel. De plus, l'augmentation des LOC en fonction du nombre de composants du système est beaucoup plus faible avec notre framework. Une tendance similaire est observée lorsqu'on examine la relation entre les LOC et le nombre de reconfigurations.

Le profileur intégré Acceleo offre la possibilité de suivre les évaluations et d'identifier les goulots d'étranglement dans le processus de génération de code. Comme le montrent les données de profilage de la figure 5.20, le temps consacré à la génération du code source pour deux composants reconfigurables est de 336 ms, tandis que moins de 90 s sont nécessaires pour le personnaliser et l'ajuster afin de répondre aux exigences finales de l'application.

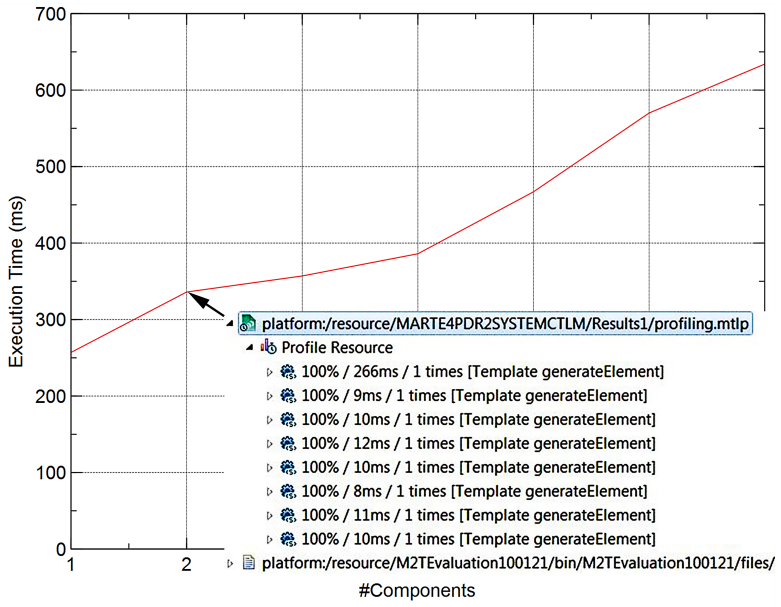



FIGURE 5.20 – *Le résultat du profilage du processus de génération de code.*

Comparé aux approches traditionnelles, l'alignement des pratiques de l'IDM sur les

concepts fondamentaux de la co-conception logicielle/matérielle permet de raccourcir le cycle de développement tout en réduisant les coûts (voir la figure 5.21).

L'enrichissement sémantique des modèles par des annotations et des constructions supplémentaires pour effectuer des transformations entraîne le prolongation de la phase de modélisation de l'application et de la plateforme d'exécution. En revanche, en initiant le processus d'automatisation, l'ensemble du cycle de vie est impacté, ce qui raccourcit logiquement les phases suivantes. Ainsi, les modèles précis élaborés auparavant contribuent à réduire le temps de développement et l'effort nécessaire à l'implémentation, comparé à une approche de codage manuel.

Des informations de traçabilité transparentes entre la conception et l'exécution offrent un support solide pour le suivi et la vérification des artefacts. En effet, une classe SystemC peut être retracée jusqu'à sa classe stéréotypée UML/MARTE, facilitant ainsi la vérification, l'automatisation des tests et fournissant un retour plus rapide sur l'analyse des performances.

Il peut être nécessaire de passer par une phase de personnalisation du code afin d'adapter et de mettre à jour le code généré en fonction des exigences spécifiques de conception. Comme anticipé, bien que notre approche demande un effort de modélisation légèrement supérieur, elle permet de réaliser un gain de temps moyen de 33 % tout en améliorant considérablement la productivité en matière de conception.

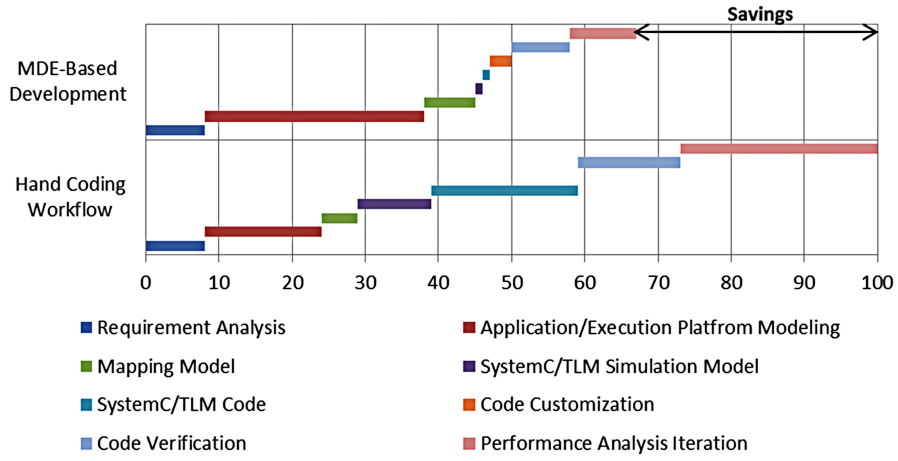



FIGURE 5.21 – *Comparaison entre un workflow de codage manuel classique et le processus de développement des SoCs reconfigurables basé modèles.*

## 5.5 CONCLUSION

Dans ce chapitre, nous avons validé le framework de modélisation et d'analyse proposé dans ce travail à travers un système reconfigurable de filtres audio. Nous avons présenté les différents modèles annotés avec les profils MARTE4DPR, MARTE4SCTLM et MARTE4AF couvrants toutes les étapes du processus de développement à différents niveau d'abstraction jusqu'à la génération automatique du code SystemC/TLM. Une analyse d'ordonnançabilité basée modèles à été également effectuée. Le framework a été évalué et les résultats ont montré l'efficacité de l'approche adoptée par rapport à une approche manuelle classique.

# Conclusions et perspectives

## Rappel des objectifs

Afin de maîtriser la complexité intrinsèque des systèmes sur puce dynamiquement reconfigurables, et d'améliorer la productivité des concepteurs, il devient impératif d'élever le niveau d'abstraction au-delà du niveau RTL dès les phases amont du processus de conception. En s'appuyant sur une analyse approfondie des limites des approches existantes, cette thèse avait donc pour objectif principal de proposer un framework dédié à la modélisation à un haut niveau d'abstraction et à l'analyse précoce des systèmes sur puce dynamiquement reconfigurables reposant sur des architectures FPGA. Plus spécifiquement, le travail visait à fournir aux concepteurs un outil permettant la génération automatique de code SystemC au niveau transactionnel à partir de spécifications exprimées en UML/MARTE, conformément aux principes de l'ingénierie dirigée par les modèles.

## Bilan des travaux effectués

L'ensemble des travaux menés au cours de cette thèse a permis d'apporter des contributions significatives à la problématique ciblée, tant sur le plan théorique que méthodologique.

Un flot de conception de haut niveau, structuré selon le modèle en Y, a été mis en œuvre en combinant les principes de l'IDM avec le développement basé sur les modèles tel que défini par le profil MARTE. Ce flot intègre des techniques de transformation de modèles et s'appuie sur des outils dédiés à la modélisation et à l'analyse, permettant ainsi une formalisation progressive et outillée du processus de conception des systèmes reconfigurables.

Afin d'assurer une prise en charge efficace de la disponibilité dynamique au sein des composants, nous avons opté pour un modèle de composant orienté service (SOCM). Ce modèle concilie les avantages des architectures à base de composants, notamment la modularité et la séparation des préoccupations, avec les principes fondamentaux du paradigme orienté services, tels que le couplage faible, la liaison dynamique et la découverte de services. Cette fusion permet ainsi de répondre aux exigences de flexibilité, de réutilisabilité et de reconfigurabilité propres aux systèmes dynamiquement reconfigurables.

Dans le but de garantir l'autonomie locale des composants et réduire ainsi la dépendance à un point de contrôle centralisé, nous avons proposé un modèle de contrôle semi-distribué assurant une coordination globale et facilitant la collaboration entre les contrôleurs locaux tout en déchargeant le coordinateur central. Ce modèle repose sur l'intégration de la boucle de rétroaction MAPE-K, offrant une application décentralisée mais cohérente de la logique de reconfiguration.

En nous appuyant sur une méthode holistique existante, nous avons élaboré un modèle de coût visant à estimer, de manière anticipée, le temps de reconfiguration attendu. Ce modèle constitue un outil d'aide à la décision permettant de faciliter l'évaluation

des performances potentielles du système dès les premières phases du processus de conception. L'objectif est double : éviter au concepteur de s'engager inutilement dans un processus complexe de mise en œuvre de la DPR lorsque les gains escomptés sont limités, et permettre un ajustement précoce des paramètres de configuration, en fonction des contraintes temporelles et structurelles identifiées.

Afin de pallier l'absence de certains concepts nécessaires à la modélisation de la DPR, du paradigme de conception des composants orientés services, du pattern de conception Abstract Factory et du langage SystemC au niveau transactionnel, nous avons procédé à l'extension du profil MARTE. Cette extension repose sur l'introduction de constructions spécifiques, de stéréotypes dédiés et de valeurs étiquetées appropriées, référencées dans trois sous profils d'annotation, à savoir : MARTE4DPR, MARTE4SCTLM et MARTE4AF.

Enfin, ces contributions ont été validées à travers une étude de cas représentative d'un crossover actif 3-voies dynamiquement reconfigurable, implémenté au moyen de filtres audio de type FIR. Cette validation a permis de démontrer non seulement l'applicabilité concrète et la pertinence du framework proposé, mais également sa capacité à accroître la productivité des concepteurs, en automatisant certaines tâches complexes de modélisation et de génération de code, tout en assurant une meilleure maîtrise des choix architecturaux grâce à une analyse précoce dès les phases amont du développement.

## Apports, limites et perspectives

À notre connaissance, les travaux menés dans cette thèse constituent la seule contribution ayant proposé une couverture sémantique aussi étendue de la DPR dans UML/-MARTE, permettant d'intégrer de manière cohérente les concepts de la DPR au sein des modèles d'architecture, de contrôle et de communication, en vue de supporter la génération automatique de code SystemC/TLM à partir de spécifications de haut niveau. Cette contribution comble ainsi un manque notable dans l'état de l'art en établissant un lien formel et outillé entre modélisation abstraite et implémentation transactionnelle des systèmes reconfigurables.

Toutefois, certains aspects de la DPR n'ont pas été traités dans cette thèse, par exemple la prise en charge de la migration des tâches dépendantes et indépendantes au sein des modules reconfigurables ou encore la prise en charge des techniques de reconfiguration imbriquée et fractionnée (nested and split reconfiguration) telles que introduites dans [Haase et al., 2024]. La reconfiguration imbriquée telle que défini dans [Xilinx Inc., 2024] désigne une approche consistant à insérer une ou plusieurs régions dynamiquement reconfigurables à l'intérieur d'une région reconfigurable existante, permettant ainsi de subdiviser le dispositif matériel en sous-régions reconfigurables plus fines. Cette capacité offre un niveau de granularité élevé dans la reconfiguration, en autorisant la division d'une région partiellement reconfigurable en plusieurs sous-régions, chacune étant reconfigurable de manière indépendante. Ce mécanisme accroît la flexibilité du système, en permettant le chargement dynamique de modules reconfigurables de tailles, de formes et de ressources hétérogènes. Alors que dans [Charaf et al., 2022], la reconfiguration fractionnée permet de générer une nouvelle configuration en extrayant les segments souhaités d'un bitstream partiel et en les insérant dans un autre. Ainsi, il devient possible de combiner deux bitstreams partiels distincts pour produire un nouveau bitstream partiel intégrant les comportements fonctionnels des deux configurations initiales. Il serait donc pertinent d'intégrer ces nouveaux concepts à un niveau élevé d'abstraction, tant au sein du profil MARTE4DPR que dans MARTE4SCTLM, afin

de permettre la modélisation de la reconfiguration imbriquée et fractionnée visant la génération automatique de code SystemC.

Egalement, pour une meilleure réutilisation et intégration des propriétés intellectuelles (IPs) hétérogènes , il est souhaitable d'utiliser un standard qui permet la descriptions des composants et des services en automatisant leur intégration dans des systèmes reconfigurables. En effet, non seulement la norme IEEE IP-XACT répond bien à ce besoin mais il est tout à fait possible d'établir une correspondance entre les concepts de MARTE et les balises XML d'IP-XACT. Cela dit, la norme IP-XACT n'inclut pas explicitement la notion de service telle qu'elle est utilisée dans notre travail. En effet, IP-XACT est principalement conçu pour décrire les composants matériels, leurs interfaces, et leurs interconnexions dans les systèmes électroniques. Cependant, il est possible d'étendre IP-XACT pour inclure les descriptions de services en utilisant des extensions personnalisées via l'élément <*spirit :vendorExtensions*>. Cela permet d'ajouter des propriétés spécifiques aux services, comme l'identité, la description fonctionnelle, l'interface, les paramètres de QoS, etc.

Issu d'un besoin concret identifié notamment lors de l'étude de la littérature, il apparaît pertinent de proposer une ontologie unificatrice des concepts relevant à la fois de l'ingénierie logicielle et matérielle, en particulier dans le contexte de la reconfiguration partielle dynamique, de l'adaptabilité et de l'auto-adaptabilité logicielle. Une telle ontologie permettrait de détecter les équivalences sémantiques, de formaliser les correspondances conceptuelles et de construire un vocabulaire commun, contribuant ainsi à lever certaines ambiguïtés terminologiques. À titre d'exemple, le terme SoC peut désigner alternativement System on Chip, Separation of Concerns, ou encore Service-oriented Computing, ce qui illustre la nécessité d'une clarification sémantique rigoureuse.

Au cours des dernières années, l'intégration de l'apprentissage automatique (Machine Learning - ML) dans le domaine de la conception assistée par ordinateur (Electronic Design Automation - EDA) s'est imposée comme une thématique de recherche majeure. De nombreuses contributions scientifiques ont été proposées dans ce contexte, visant à exploiter les techniques d'apprentissage automatique pour améliorer les différentes étapes du flot de conception des systèmes électroniques. [Huang et al., 2021] classifie ces contributions en quatre catégories : la prise de décision dans les méthodes traditionnelles, la prédiction de performance, l'optimisation en boîte noire et la conception automatisée. De la meme façon, l'intégration de l'apprentissage automatique à la modélisation des domaines métiers dans un contexte d'IDM a été abordé dans [Hartmann et al., 2019] et [Rädler et al., 2024]. Dans notre framework, Il serait donc pertinent d'intégrer l'apprentissage automatique au niveau modélisation en augmentant les profils MARTE4DPR et MARTE4SCTLM avec des concepts de ML pour générer du code SystemC. Ce code qui servira comme entrée au niveau HLS (High Level Synthesis) permettra d'extraire des paramètres d'entrée à des algorithmes de ML afin de prédire le temps de reconfiguration, l'utilisation des ressources et les délais des opérations, ainsi que d'autres métriques.

# Mes contributions scienfifiques

— K. Allem, E-B. Bourennane et Y.Khelfaoui. A SystemC/TLM Service-Oriented Component-Based Approach for Partial Dynamically Reconfigurable Systems Modeling. Dans *Proceedings of the International Conference on Embedded Systems in Telecommunications and Instrumentation (ICESTI'16)*, Annaba, Algeria, October 2016.

— K. Allem, E-B. Bourennane et Y.Khelfaoui. A Service-Oriented Component-Based Framework for Dynamic Reconfiguration Modeling targeting SystemC/TLM. *International Journal of reconfigurable Computing*, 2021 : 1-25, 2021. https://doi.org/10.1155/2021/5584391.

# A

# Aperçu du profil MARTE

Sommaire

## A.1 Introduction

MARTE (Modeling and Analysis of Real-Time and Embedded systems) est un profil UML conçu pour enrichir le langage UML de capacités avancées dédiées au développement dirigée par les modèles des systèmes temps réel et embarqués. Ces concepts fondamentaux établis par MARTE sont affinés selon deux perspectives complémentaires : la modélisation et l'analyse. Ainsi, le volet modélisation fournit un cadre structuré permettant de couvrir l'ensemble des phases, depuis la spécification jusqu'à la conception détaillée des aspects temps réel et embarqués des systèmes. Par ailleurs, MARTE aborde également les problématiques d'analyse fondée sur les modèles. Son objectif n'est pas d'introduire de nouvelles méthodes d'analyse, mais plutôt d'offrir les moyens nécessaires à leur réalisation. À cette fin, MARTE propose des mécanismes d'annotation des modèles, permettant de recueillir les informations indispensables à des analyses ciblées, en particulier celles portant sur les performances et l'ordonnançabilité [Obj, 2023b].

L'utilisation du profil MARTE offre plusieurs avantages notables, parmi lesquels :

— La définition d'un cadre commun de modélisation qui intègre à la fois les dimensions matérielles et logicielles des systèmes temps réel et embarqués, facilitant ainsi la communication et l'échange entre développeurs.
— L'amélioration de l'interopérabilité entre les différents outils de développement mobilisés lors des phases de spécification, conception, vérification ou génération automatique de code.
— La promotion de modèles permettant d'effectuer des prédictions quantitatives précises relatives aux caractéristiques temps réel et embarquées des systèmes, en prenant simultanément en compte les spécificités tant matérielles que logicielles.

## A.2 Les usages possibles de MARTE

La norme MARTE distingue deux grandes catégories d'utilisateurs : d'une part, les méthodologistes et fournisseurs d'infrastructures, et d'autre part, les utilisateurs du langage. La première catégorie regroupe ceux qui, en se basant sur MARTE, adaptent son utilisation aux besoins d'un domaine spécifique, apportent un support à l'exécution des modèles obtenus, ou en spécialisent ou étendent certains aspects en fonction d'une technologie ou d'un domaine particulier. Le travail que nous avons réalisé relève principalement de cette catégorie. La seconde catégorie désigne ceux qui mettent en œuvre les résultats produits par les participants de la première catégorie. Il est essentiel de souligner que, tout comme UML, MARTE est un langage de modélisation dont la spécification ne décrit pas un usage particulier. Il revient donc aux membres de la première catégorie d'utilisateur de fournir des informations aux ingénieurs et utilisateurs de la norme.

## A.3 L'architecture de MARTE

MARTE est un profil UML conçu pour aider à développer des systèmes temps réel embarqués en utilisant une approche d'IDM. Il est constitué d'un ensemble d'extensions adaptées aux concepts généraux d'UML, offrant aux concepteurs des constructions de langage de première classe pour leur domaine d'application. Le profil MARTE repose sur deux principales préoccupations :

— d'une part, la modélisation des caractéristiques des système temps réel et embarqués ;

— d'autre part, l'annotation des modèles d'application pour analyser les propriétés du système.

Pour répondre à toutes ces exigences, MARTE est organisé en une structure hiérarchique de sous-profils, comme illustré dans la figure A.1.

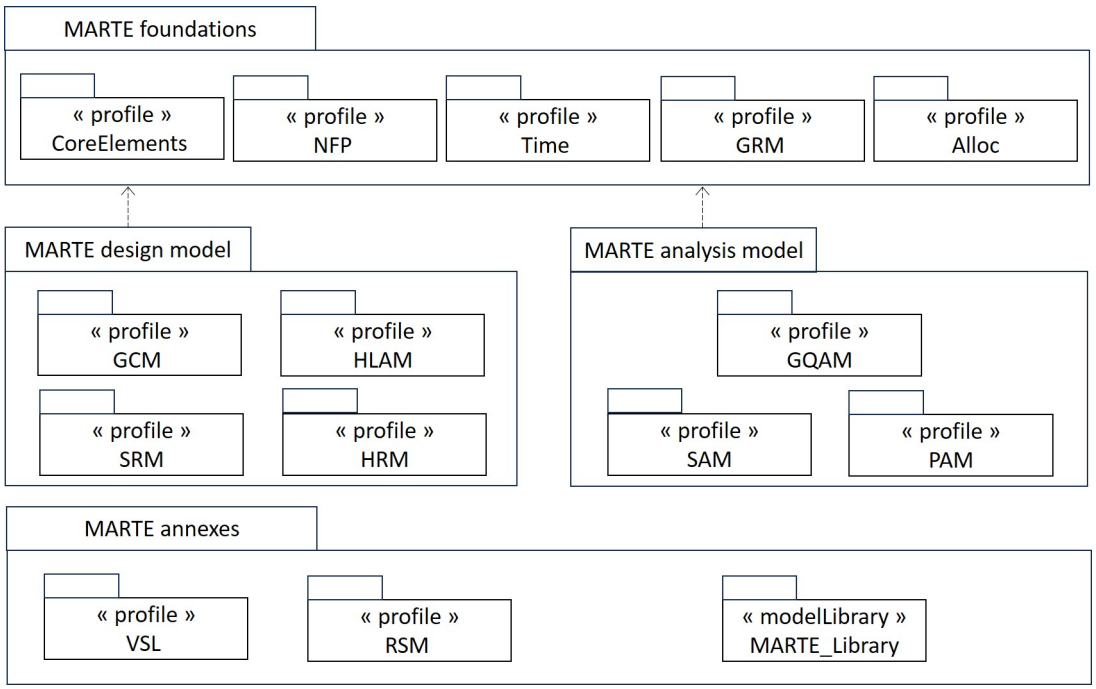



FIGURE A.1 – *Architecture globale du profil MARTE.*

### A.3.1 Concepts de base

Les concepts de base de MARTE sont définis par le paquetage *foundations*, qui se compose des quatre sous-profiles suivants :

— *CoreElements* : ce sous-profil reprend les principes fondamentaux de la modélisation pour la conception et l'analyse, tels que la différenciation entre les éléments de classification, les types ou les classes, et les instances générées à partir de ces éléments de classification. Les concepts fondamentaux qui permettent d'exprimer le comportement de ces classifieurs ou instances sont également présents, tels que les actions, les évènements et les déclencheurs. Ces éléments sont utilisés pour établir les liens de causalité entre les changements dynamiques des éléments modélisés. L'ensemble de ces concepts est en accord avec les définitions similaires d'UML et ne crée donc pas de stéréotypes. Dans ce sous-profil, les seuls stéréotypes définis portent sur la modélisation des comportements en fonction des modes d'opération. Ces modes peuvent illustrer les différentes opérations d'un système concernant le traitement des pannes, les paramètres de qualité de service ou les différentes phases d'opération. Les règles de transition entre les différentes configurations représentant les modes d'opération sont ainsi représentées par un diagramme de machine à état stéréotypée *ModeBehavior*.

— *NFP (Non-Functional Properties)* et *VSL (Value Specification Language)* : le sous-profil NFP fournit les constructions de modélisations nécessaires pour déclarer, qualifier et appliquer à un modèle UML des informations non fonctionnelles. Les valeurs des annotations non fonctionnelles définies dans le sous-profil NFP sont spécifiées dans le langage VSL, qui est indispensable pour NFP.

— *Time* et *CCSL (Clock-Constraint Specification Language)* : le concept de temps est défini par le sous-profil Time, qui est essentiel pour les systèmes temps réel embarqués. Trois modèles de temps sont proposés par MARTE : un modèle chronométrique, un modèle logique et un modèle de temps qui correspond aux méthodes de développement basées sur le paradigme de la programmation synchrone. Ce sous-profil désigne également un ensemble de mécanismes de bas niveau pour gérer le temps, tels que des horloges, des observateurs ou encore des événements temporisés. CCSL est un langage textuel qui s'ajoute au profil Time pour décrire les contraintes entre des horloges.
— *GRM (Generic Resource Modeling)* : ce sous-profil permet la modélisation des plateformes d'exécution, qu'elles soient logicielles ou matérielles. Le concept de plateforme a été abstrait dans la norme sous la forme de ressources fournissant des services et des instances de ces ressources exécutant ces services. Les différents types de ressources définis permettent de modéliser des ressources de calcul, de stockage, de communication, de gestion du temps, de synchronisation, de concurrence et d'accès à des périphériques. Pour compléter la modélisation des ressources, MARTE introduit des concepts permettant la modélisation des services de gestion des ressources, d'ordonnancement et d'exclusion mutuelle. La taxonomie des ressources du profil GRM permet donc de modéliser tous les aspects d'une plateforme à un niveau d'abstraction système, indépendamment des préoccupations logicielles et/ou matérielles. Ces deux aspects sont traités dans deux autres sous-profils spécialisés dans MARTE, respectivement *SRM (Software Resource Modeling)* et *HRM (Hardware Resource Modeling)*.
— *Alloc (Allocation Modeling)* : ce sous-profil définit les concepts nécessaires à la description explicite d'un modèle d'allocation mettant en relation les éléments du modèle de l'application avec les éléments du modèle de la plateforme d'exécution. L'allocation modélise des liens entre des modèles représentés au même niveau d'abstraction, à la différence des liens de raffinement ou d'abstraction qui relient des éléments modélisant les mêmes artefacts, mais à des niveaux d'abstraction distincts. MARTE fournit une notation pour des allocations totales ou partielles, spatiales ou temporelles, mais ne permet pas de vérifier leur consistance ou leur complétude. Cette tâche incombe à des outils d'analyse susceptibles d'exploiter cette notation.

Les sous-profils décrits précédemment constituent le socle conceptuel de MARTE. Ainsi, des concepts plus spécialisés sont définis pour modéliser plus finement les applications temps réel embarquées d'une part, et d'autre part, pour intégrer des approches permettant des analyses quantitatives des modèles.

### A.3.2 Modélisation

Le développement des systèmes temps réel embarqués basés sur des modèles est essentiellement supporté de façon déclarative avec MARTE, en utilisant des annotations associées aux éléments du modèle pour préciser leurs propriétés. Les quatre sous-profils de MARTE dédiés à cette préoccupation sont :

— *GCM (Generic Component Model)* : ce sous-profil reprend les structures composites d'UML et en étend un sous-ensemble afin de répondre au mieux aux besoins du domaine en matière de modélisation à base de composants. Son apport principal par rapport à UML est la possibilité de modéliser des composants communiquant en s'échangeant des flots de données en plus du modèle client-serveur d'UML.
— *HLAM (High Level Application Modeling)* : ce sous-profil fournit un ensemble d'ex-

tensions à UML permettant d'annoter les éléments d'un modèle afin de décrire des caractéristiques aussi bien quantitatives (période, délais) que qualitatives (comportement, communication et concurrence) des systèmes temps réel. Le concept clé proposé est celui de l'unité temps réel *(RtUnit)* qui encapsule une ressource d'exécution autonome et constitue le concept central de la gestion de la concurrence (voir figure A.2). Une *RtUnit* gère les propriétés temps réel des messages qu'elle reçoit, peut traiter plusieurs messages en concurrence et est capable d'invoquer les services d'autres *RtUnit*.

— *SRM (Software Resource Modeling)* et *HRM (Hardware resource Modeling)* : ces deux sous-profils étendent le sous-profil GRM afin de proposer, respectivement, une modélisation détaillée des ressources logicielles (principalement les systèmes d'exploitation temps réel) et une modélisation approfondie des ressources matérielles (cartes, puces, processeurs, mémoires, moyens de communication, etc.).

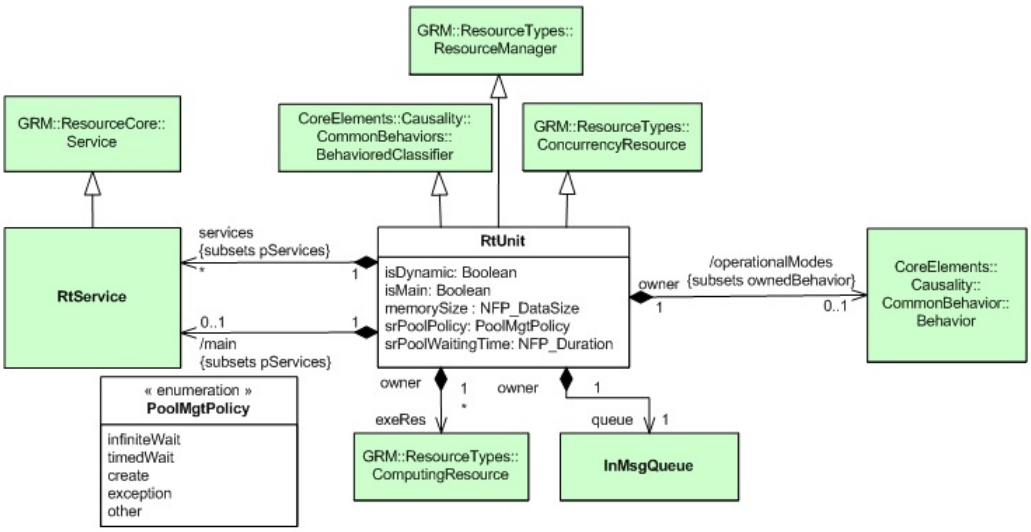



FIGURE A.2 – *L'unité temps réel du paquetage HLAM [Obj, 2023b].*

### A.3.3 Analyse

Les paquetages définis dans cette section permettent d'annoter les modèles de conception, élaborés à l'aide des paquetages de modélisation décrits précédemment, avec des informations quantitatives. Ces annotations facilitent des analyses formelles statiques directement sur les modèles, sans recourir à la simulation, afin de détecter les problèmes tôt dans le cycle de développement, réduisant ainsi les risques et les coûts associés. La partie analyse du profil MARTE contient trois sous-profils :

— *GQAM (Generic Quantative Analysis Modeling)* : ce sous-profil permet de décrire comment le comportement d'un système utilise les ressources. Les différents types d'analyses quantitatives proposés déterminent les valeurs des propriétés non fonctionnelles de sortie (comme le temps de réponse, les dépassements d'échéances, les taux d'utilisation des ressources, les tailles des files d'attentes, etc.) en fonction des valeurs des propriétés non fonctionnelles d'entrée (comme le taux de requêtes, les exigences d'exécution, les délais, les objectifs de qualité de service, etc.). Pour faciliter ces analyses, le profil suggère de modéliser, dans un cadre analytique, diverses charges de travail incluant d'une part les événements déclencheurs, et d'autre part les scénarios de comportement induits par ces événements. Ces scénarios sont constitués d'étapes précisant l'utilisa-

tion des ressources. Les annotations quantitatives peuvent être définies à l'aide d'un ensemble varié d'attributs de type NFP. Les figures A.3 et A.4 représentent respectivement le contexte d'analyse sous les aspects **WorkloadBehavior** et **ResourcePlateform**.

— *SAM (Schedulability Analysis Modeling)* : ce sous-profil fournit des annotations dédiées à l'analyse d'ordonnançabilité basée sur des modèles. Une analyse précoce d'un modèle de conception aide les développeurs à détecter des architectures temps réel potentiellement non viables et prévient des erreurs de conception coûteuses, en particulier celles liées au comportement temporel. En revanche, une analyse tardive d'un système implémenté permet aux analystes de découvrir (avec des informations quantitatives plus précises sur le système) des défauts liés au temps, ou d'évaluer l'impact de possibles migrations ou modifications de plateforme sur les stratégies d'ordonnancement. Des annotation détaillées permettent donc la spécification des différentes propriétés temporelles de l'utilisation des ressources selon le type d'ordonnancement et le modèle d'analyse associé : Rate Monotonic Analysis, Deadline Monotonic Analysis ou encore Earliest Deadline First.

— *PAM (Performance Analysis Modeling)* : ce sous-profil concerne l'analyse des propriétés temporelles des systèmes à meilleur effort et des systèmes embarqués en temps réel souple. L'analyse s'applique à divers domaines comme les services web, les télécommunications et les services en réseau. Les mesures de performance (sorties de l'analyse) sont statistiques, telles que le débit moyen et le délai ou la probabilité de manquer un temps de réponse cible, et les paramètres d'entrée peuvent être probabilistes, tels qu'un processus d'arrivée aléatoire ou un temps d'exécution aléatoire pour une trame de donnée. Les techniques d'analyse de performance incluent les simulations, les modèles de files d'attente étendus et les réseaux de Petri stochastiques. L'analyse peut porter sur un cas unique ou sur plusieurs cas, comme l'analyse de sensibilité pour explorer des paramètres opérationnels ou l'analyse de l'évolutivité pour évaluer les capacités de la conception.

### A.3.4 Annexes

La norme MARTE comprend plusieurs annexes qui sont soit d'intérêt général (ne sont pas dédiées uniquement à MARTE) comme le sous profil *RSM (Repetitive Structure Modeling)* qui permet de modéliser des structures répétitives dans les architectures matérielles et logicielles complexes ; soit utilisées à la fois par le profil et par les modèles utilisant le profil comme le langage VSL (Value Specification Language) qui est un langage d'expression textuelle permettant de modéliser des valeurs, des contraintes et des paramètres dans des modèles UML, à cela s'ajoute une bibliothèque de types, unités et valeurs normalisés pour les systèmes temps réel et embarqués. Enfin, plusieurs annexes sont consacrées à la documentation.

## A.4 Quelques outils IDM

L'essor de l'IDM s'est accompagné d'une prolifération d'outils visant à en faciliter la mise en œuvre. Bien que nombreux, les outils développés dans le cadre académique restent généralement difficiles d'accès, tant en termes de disponibilité que d'utilisabilité, ce qui limite leur diffusion et participe à une faible reconnaissance de ces outils au sein de la communauté. Dans [Jácome-Guerrero et al., 2017] et [Soukaina et al., 2018], une étude comparative entre les différents outils académique et commerciaux est fournie.

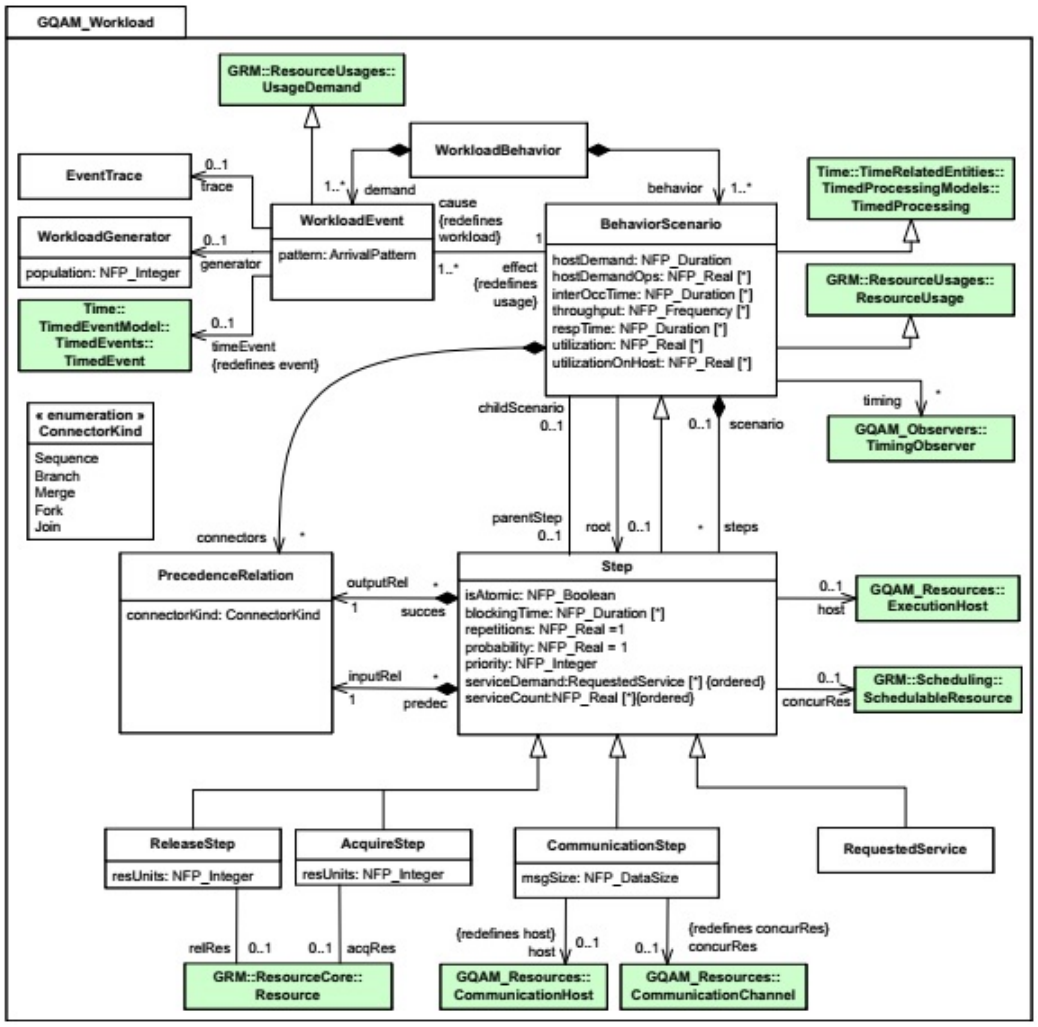



FIGURE A.3 – *Le package GQAM_Workload du modèle de domaine GQAM [Obj, 2023b].*

Nous présentons ci-après les outils les plus largement adoptés, qui permettent non seulement une modélisation conforme au profil MARTE, mais qui intègrent également des mécanismes d'extension, d'analyse, de vérification ainsi que de génération de code.

— **Outils de modélisation :** Papyrus [1] , RSA (Rational Software Architect) [2], MagicDraw [3], Cheddar [4], Modelio [5], Entreprise Architect [6], IBM Rhapsody [7], etc.

— **Outils de métamodélisation :** Eclipse Modeling Framework (EMF) [8], MetaEdit+ [9], Kermeta [10], etc.

1. https://eclipse.dev/papyrus/. Il fournit l'implémentation de référence de MARTE.
2. https://www.ibm.com/products/rational-software-architect-designer
3. https://www.magicdraw.com/main.php
4. https://beru.univ-brest.fr/cheddar/
5. https://www.modelio.org/index.htm
6. https://sparxsystems.com/
7. https://www.ibm.com/products/engineering-rhapsody
8. https://projects.eclipse.org/projects/modeling.emf.emf
9. https://www.metacase.com/products.html
10. https://diverse-project.github.io/k3/

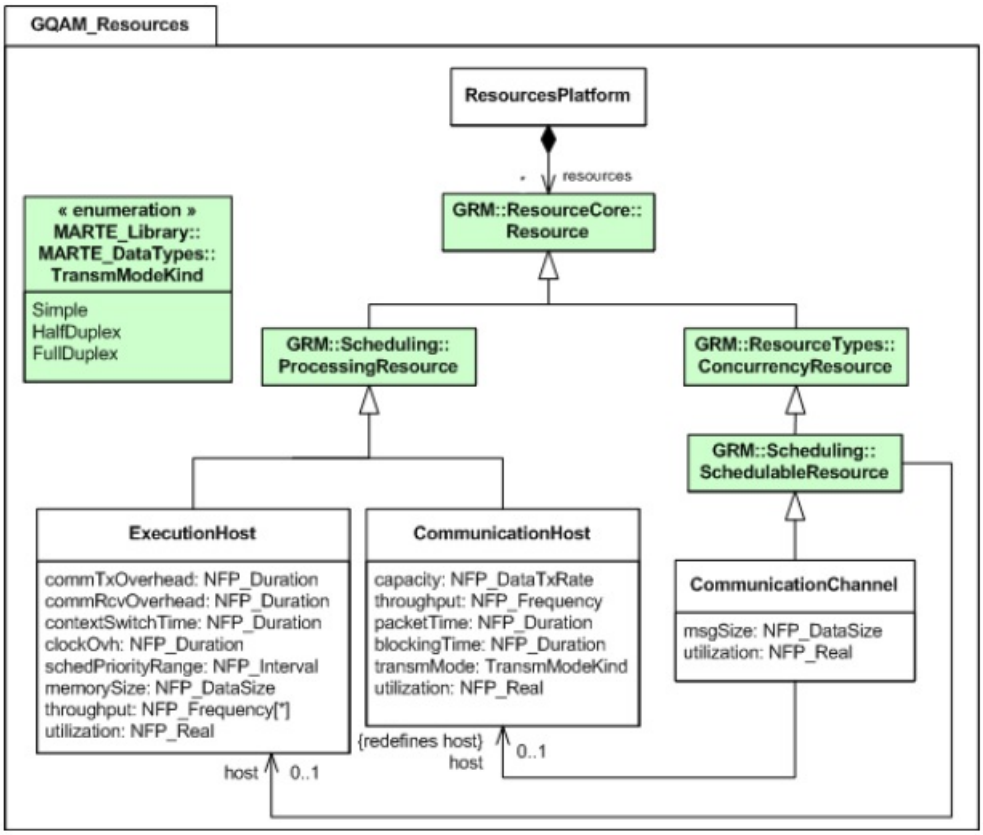



FIGURE A.4 – *Le package GQAM_resources du modèle de domaine GQAM [Obj, 2023b].*

— **Outils de transformation de modèles :** ATL [11], QVT (Query/View/Transformation) [12], Acceleo [13], VIATRA [14], Epsilon [15], etc.

11. https://eclipse.dev/atl/
12. https://www.omg.org/spec/QVT
13. https://eclipse.dev/acceleo/
14. https://projects.eclipse.org/projects/modeling.viatra
15. https://projects.eclipse.org/projects/modeling.epsilon

# B

# Processus engendrés dans SystemC

Sommaire

## B.1 Le fonction sc_spawn

La fonction sc_spawn est utilisée pour instancier un processus engendré, qu'il soit de nature statique ou dynamique. La fonction sc_spawn peut être appelée durant la phase d'élaboration. Dans ce cas, le processus engendré devient un sous-processus (enfant) de l'instance de module au sein de laquelle l'appel à sc_spawn est effectué, ou bien un objet de niveau supérieur (top-level) si l'appel est réalisé depuis la fonction sc_main.

La fonction sc_spawn peut être appelée durant la simulation. Dans ce cas, le processus engendré devient un sous-processus (enfant) du processus à l'origine de l'appel à sc_spawn. Cette fonction peut être invoquée à partir d'un processus de type méthode (method process), d'un processus thread (thread process) ou d'un processus thread cadencé (clocked thread process).

Le listing B.1 présente deux versions surchargées de la fonction sc_spawn, implémentées comme fonctions templates pour permettre la flexibilité sur le type d'objet ou de fonction à exécuter.

```
1  template <typename T>
2  sc_process_handle sc_spawn(
3      T object,
4      const char* name_p = 0,
5      const sc_spawn_options* opt_p = 0 );
6
7  template <typename T>
8  sc_process_handle sc_spawn(
9      typename T::result_type* r_p,
10     T object,
11     const char* name_p = 0,
12     const sc_spawn_options* opt_p = 0 );
13
14 #define sc_bind boost::bind
15 #define sc_ref(r) boost::ref(r)
16 #define sc_cref(r) boost::cref(r)
```

Listing B.1 – *Déclaration de la fonction* `sc_spawn`

Le processus ou le module à partir duquel la fonction sc_spawn est invoquée est considéré comme le parent du processus engendré. Ainsi, un ensemble d'instances de processus dynamiques peut entretenir une relation hiérarchique, analogue à celle de la hiérarchie des modules, cette relation étant reflétée dans les noms hiérarchiques attribués aux instances de processus.

Si la fonction sc_spawn est appelée durant la phase d'évaluation, le processus engendré devient exécutable au cours de cette même phase d'évaluation (sauf si la méthode dont_initialize a été invoquée pour cette instance de processus). En revanche, si sc_spawn est appelée pendant la phase de mise à jour, le processus engendré sera rendu exécutable dès la phase d'évaluation suivante (à moins que dont_initialize n'ait été appelée pour cette instance).

L'argument de type T doit être soit un pointeur de fonction, soit un objet fonction, c'est-à-dire un objet d'une classe surchargeant l'opérateur operator() en tant que fonction membre. Cet argument désigne la fonction associée à l'instance de processus engendré, c'est-à-dire la fonction qui sera exécutée par le processus engendré. Il s'agit du seul argument obligatoire requis par la fonction sc_spawn.

Lorsqu'il est présent, l'argument de type T result_type doit transmettre un pointeur vers une zone mémoire destinée à recevoir la valeur retournée par la fonction associée à l'instance du processus. Dans ce cas, l'argument de type T doit être un objet fonction appartenant à une classe définissant un type imbriqué nommé result_type. De plus,

l'opérateur operator() de cet objet fonction doit avoir pour type de retour result_type. Il incombe à l'application de garantir que la zone mémoire pointée reste valide au moment où la fonction engendrée termine son exécution.

Les macros sc_bind, sc_ref et sc_cref sont fournies à titre de commodité lors de l'utilisation des bibliothèques libres Boost C++ pour lier des arguments aux fonctions engendrées. Le passage d'arguments aux processus engendrés constitue un mécanisme puissant permettant de paramétrer ces processus au moment de leur création, et d'autoriser leur mise à jour progressive à travers des arguments passés par référence. La bibliothèque Boost.Bind offre une méthode pratique pour transmettre des arguments par valeur, par référence ou par référence constante aux fonctions engendrées, bien que son utilisation ne soit pas obligatoire.

L'argument de type const char* doit spécifier le nom, sous forme de chaîne de caractères, de l'instance de processus engendré. Ce nom est transmis par l'implémentation au constructeur de la classe sc_object, laquelle constitue la superclasse de base de l'instance du processus engendré. Si aucun nom n'est fourni, ou si la chaîne est vide, l'implémentation générera automatiquement un nom pour l'instance de processus en appelant la fonction sc_gen_unique_name, en utilisant la chaîne "thread_p" comme préfixe dans le cas d'un processus de type thread, ou "method_p" dans le cas d'un processus de type méthode.

L'argument de type sc_spawn_options permet de définir les options de génération associées à l'instance du processus engendré. En l'absence de cet argument, l'instance du processus adopte les valeurs par défaut telles que définies par les fonctions membres de la classe sc_spawn_options. L'application n'est pas tenue de conserver l'objet sc_spawn_options valide après le retour de la fonction sc_spawn.

Si un argument de type sc_spawn_options est fourni, un nom de processus sous forme de chaîne de caractères doit également être spécifié, même si cette chaîne peut être vide.

La fonction sc_spawn retourne un descripteur de processus valide correspondant à l'instance du processus engendré. Une instance de processus peut être suspendue, reprise, désactivée, réactivée, terminée ou réinitialisée à l'aide des fonctions membres de la classe sc_process_handle.

La directive SC_INCLUDE_DYNAMIC_PROCESSES dans une simulation SystemC/TLM est utilisée pour activer la gestion des processus dynamiques.

## B.2 La classe sc_spawn_options

La classe sc_spawn_options est utilisée pour créer un objet destiné à être passé en argument à la fonction sc_spawn lors de l'instanciation d'un processus engendré. Les options de génération définissent certaines propriétés du processus ainsi créé. L'appel des fonctions membres d'un objet sc_spawn_options n'a aucun effet sur un processus donné, sauf si cet objet est explicitement transmis en paramètre à la fonction sc_spawn. La fonction sc_spawn peut être appelée soit durant la phase d'élaboration, soit à partir d'un processus statique, dynamique, engendré (spawned) ou non engendré (unspawned) au cours de la simulation. De manière analogue, les objets de la classe sc_spawn_options peuvent être créés ou modifiés aussi bien pendant l'élaboration que durant la simulation.

## B.3 Exemple

Le listing B.2 [IEEE, 2023] illustre diverses manières d'utiliser la fonction sc_spawn pour engendrer dynamiquement des processus dans un module SystemC, en mettant en œuvre des fonctions classiques, des foncteurs, des objets liés via Boost, ainsi que des options de génération (sc_spawn_options).

```
int f();

struct Functor {
    typedef int result_type;
    result_type operator() ();
};

Functor::result_type Functor::operator() () {
    return f();
}

int h(int a, int& b, const int& c);

struct MyMod : sc_core::sc_module {
    sc_core::sc_signal<int> SC_NAMED(sig);

    void g();

    SC_CTOR(MyMod) {
        SC_THREAD(T);
    }

    int ret;

    void T() {
        using namespace sc_core;

        sc_spawn(&f); // Spawn a function without arguments and
            discard any return value.

        // Spawn a similar process and create a process handle.
        sc_process_handle handle = sc_spawn(&f);

        Functor fr;
        sc_spawn(&ret, fr); // Spawn a function object and catch the
            return value.

        sc_spawn_options opt;
        opt.spawn_method();
        opt.set_sensitivity(&sig);
        opt.dont_initialize();

        sc_spawn(f, "f1", &opt); // Spawn a method process named "f1",
             sensitive to sig, not initialized.

        // Spawn a similar process named "f2" and catch the return
            value.
        sc_spawn(&ret, fr, "f2", &opt);

        // Spawn a member function using Boost bind.
        sc_spawn(sc_bind(&MyMod::g, this));
```

```
48
49          int A = 0, B, C;
50
51          // Spawn a function using Boost bind, pass arguments
52          // and catch the return value.
53          sc_spawn(&ret, sc_bind(&h, A, sc_ref(B), sc_cref(C)));
54      }
55  };
```

Listing B.2 – *Exemple d'utilisation de la fonction sc_spawn.*

# Bibliographie


Nour Alhouda Aboud, Eric Cariou, Eric Gouardères, et Philippe Aniorté. Correspondances sémantiques entre des modèles de services, de composants et d'agents. 2012. URL https://api.semanticscholar.org/CorpusID:149172516.

Felwah Al-Hamed, Sara Al-Doweesh, Rana Al-Omar, Wejdan Al-Doweesh, et Abeer Najjar. Service oriented software engineering (sose) :a survey and gap analysis. Dans *2018 21st Saudi Computer Society National Computer Conference (NCC)*, pages 1–6, 2018. URL https://doi.org/10.1109/NCG.2018.8593028.

Sarah Alhozaimy, Daniel A. Menascé, et Massimiliano Albanese. Resilience and performance quantification of dynamic reconfiguration. *Future Generation Computer Systems*, 160 :120–130, 2024. ISSN 0167-739X. URL https://doi.org/10.1016/j.future.2024.05.040.

K. Allem, E-B. Bourennane, et Y. Khelfaoui. A service-oriented component-based framework for dynamic reconfiguration modeling targeting systemc/tlm. *International Journal of reconfigurable Computing*, 2021 :1–25, 2021. URL https://doi.org/10.1155/2021/5584391.

K. Allem, E-B. Bourennane, et Y.Khelfaoui. A systemc/tlm service-oriented component-based approach for partial dynamically reconfigurable systems modeling. Dans *Proceedings of the International Conference on Embedded Systems in Telecommunications and Instrumentation (ICESTI'16)*, Annaba, Algeria, October 2016.

OSGi Alliance. *The OSGi Alliance, Release 8*, 2020. URL https://docs.osgi.org/download/r8/osgi.core-8.0.0.pdf.

*Vitis High-Level Synthesis User Guide*. AMD, 2023. URL https://docs.amd.com/r/en-US/ug1399-vitis-hls.

João Claudio Américo et Didier Donsez. Service component architecture extensions for dynamic systems. Dans Chengfei Liu, Heiko Ludwig, Farouk Toumani, et Qi Yu, éditeurs, *Service-Oriented Computing*, pages 32–47, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-34321-6. URL https://doi.org/10.1007/978-3-642-34321-6_3.

Jesper Andersson, Mauro Caporuscio, Mirko D'Angelo, et Annalisa Napolitano. Architecting decentralized control in large-scale self-adaptive systems. *Computing*, 105 : 1849–1882, 2023. URL https://doi.org/10.1007/s00607-023-01167-9.

Jesper Andersson, Rogério de Lemos, Sam Malek, et Danny Weyns. Modeling dimensions of self-adaptive software systems. Dans Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, et Jeff Magee, éditeurs, *Software Engineering for Self-Adaptive Systems*, pages 27–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-02161-9. URL https://doi.org/10.1007/978-3-642-02161-9_2.

Paolo Arcaini, Elvinia Riccobene, et Patrizia Scandurra. Modeling and analyzing mape-k feedback loops for self-adaptation. Dans *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 13–23. IEEE Press, 2015. URL https://doi.org/10.1109/SEAMS.2015.10.

C. Atkinson, R. Gerbig, et B. Kennel. Symbiotic general-purpose and domain-specific languages. Dans *Proceedings of the 34th International Conference on Software Engineering*, volume 10563 de *ICSE '12*, page 1269–1272. IEEE Press, 2012. URL https://doi.org/10.1007/978-3-319-66854-3_1.

R. Iris Bahar, Alex K. Jones, Srinivas Katkoori, Patrick H. Madden, Diana Marculescu, et Igor L. Markov. Workshops on extreme scale design automation (ESDA) challenges and opportunities for 2025 and beyond. *CoRR*, abs/2005.01588, 2020. URL https://arxiv.org/abs/2005.01588.

A. Basu, M. Bozga, et J. Sifakis. Modeling heterogeneous real-time components in bip. Dans *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)*, pages 3–12, 2006.

Christian Beckhoff, Dirk Koch, et Jim Torresen. Go ahead : A partial reconfiguration framework. Dans *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 37–44, 2012. URL https://doi.org/10.1109/FCCM.2012.17.

Alizadeh Bijan et Shiroei Masoud. Automatic correction of rtl designs using a lightweight partial high level synthesis. *Integration*, 91 :173–181, 2023. ISSN 0167-9260. URL https://doi.org/10.1016/j.vlsi.2023.04.001.

D. C. Black, J. Donovan, B. Bunton, et A. Keist. *SystemC : From the Ground Up, Second Edition*. Springer édition, 2010. ISBN 978-0-387-69958-5.

C. Bocka, R. Barbaua, et A. Narayanana. Bpmn profile for operational requirements. *Journal of Object Technology*, 13(2) :1–35, 2014. URL https://doi.org/10.5381/jot.2014.13.2.a1.

F. Boutekkouk, M. Benmohammed, S. Bilavarn, et M. Auguin. Uml2.0 profiles for embedded systems and systems on a chip (socs). *Journal of Object Technology*, 8(1) :135–157, Janvier 2009. ISSN 1660-1769. URL http://www.jot.fm/contents/issue_2009_01/article1.html.

Fateh Boutekkouk. Automatic systemc code generation from uml models at early stages of systems on chip design. *International Journal of Computer Applications*, 8(6) :10–17, October 2010. ISSN 0975-8887. URL https://ijcaonline.org/archives/volume8/number6/1215-1744/.

Andrew Boutros et Vaughn Betz. *Field-Programmable Gate Array Architecture*. Springer Nature Singapore, Singapore, 2022. ISBN 978-981-15-6401-7. URL https://doi.org/10.1007/978-981-15-6401-7_49-1.

Hongyu Pei Breivold et Magnus Larsson. Component-based and service-oriented software engineering : Key concepts and principles. Dans *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)*, pages 13–20, 2007. URL https://doi.org/10.1109/EUROMICRO.2007.25.

A. V. Brito, M. Kuhnle, M. Hubner, J. Becker, et E. U. K. Melcher. Modelling and simulation of dynamic and partially reconfigurable systems using systemc. Dans *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, ISVLSI'07, pages 35–40. IEEE, 2007. URL https://doi.org/10.1109/ISVLSI.2007.69.

A. V. Brito, G. Silveira, et E. U. K. Melcher. A methodology for modelling and simulation of dynamic and partially reconfigurable systems. Dans Alisson V. Brito, éditeur, *Dynamic Modelling*, Chapitre 3. IntechOpen, 2010. URL https://doi.org/10.5772/7094.

Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, et Jean-Bernard Stefani. The fractal component model and its support in java. *Software : Practice and Experience*, 36(11-12) :1257–1284, 2006. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.767.

Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, et Günter Kniesel. Towards a taxonomy of software change. *Journal of Software Maintenance*, 17(5) :309–332, 2005. URL https://doi.org/10.1002/smr.319.

Alex R. Bucknall et Suhaib A. Fahmy. Zypr : End-to-end build tool and runtime manager for partial reconfiguration of fpga socs at the edge. *ACM Trans. Reconfigurable Technol. Syst.*, 16(3), jun 2023. ISSN 1936-7406. URL https://doi.org/10.1145/3585521.

T. Bures, P. Hnetynka, et F. Plasil. Sofa 2.0 : Balancing advanced features in a hierarchical component model. Dans *Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*, pages 40–48, 2006. URL https://doi.org/10.1109/SERA.2006.62.

Rajkumar Buyya, Christian Vecchiola, et S. Thamarai Selvi. Chapter 1 - introduction. Dans Rajkumar Buyya, Christian Vecchiola, et S. Thamarai Selvi, éditeurs, *Mastering Cloud Computing*, pages 3–27. Morgan Kaufmann, Boston, 2013. ISBN 978-0-12-411454-8. URL https://doi.org/10.1016B978-0-12-411454-8.00001-2.

C. Canal, J. M. Murillo, et P. Poizat. Software adaptation. *Obj. Logiciel, Base données, Réseaux*, 12(1) :9–31, 2006. URL https://doi.org/10.3166/objet.12.1.9-31.

João M. P. Cardoso. Programming strategies for runtime adaptability. Dans *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–8, 2012. URL https://doi.org/10.1109/ReCoSoC.2012.6322875.

Ewerson Carvalho, Ney Calazans, Fernando Moraes, et Daniel Mesquita. Reconfiguration control for dynamically reconfigurable systems. Dans *DCIS*, volume 4, pages 405–410. Citeseer, 2004.

R. Carvalho, R. Alencar, et A. Sarmento. Generation of systemc simulation models from service level uml diagrams. Dans *Proceedings of the VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 114–121. IEEE, 2018. URL https://doi.org/10.1109/SBESC.2018.00025.

H. Cervantes et R.S. Hall. Autonomous adaptation to dynamic availability using a service-oriented component model. Dans *Proceedings. 26th International Conference on Software Engineering*, pages 614–623, 2004. URL https://doi.org/10.1109/ICSE.2004.1317483.

T. Cervero, J. Dondo, A. Gómez, X. Peña, S. Lopez, F. Rincon, R. Sarmiento, et J.C. Lopez. A resource manager for dynamically reconfigurable fpga-based embedded systems. Dans *Proceedings of the 2013 Euromicro Conference on Digital System Design)*, pages 633–640. IEEE, 2013. URL https://doi.org/10.1109/DSD.2013.75.

Najdet Charaf, Christoph Tietz, et Diana Goehringer. Manabit : A versatile tool for manipulating and analyzing fpga bitstreams. Dans *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 1–1, 2022. URL https://doi.org/10.1109/FCCM53951.2022.9786106.

S. Cherif. *A model-based approach for the conception of dynamically reconfigurable systems : From MARTE to RecoMARTE*. PhD thesis, Lille University of Science and Technology, France, 2013.

Benoit Combemale. *Approche de métamodélisation pour la simulation et la vérification de modèle : Application à l'ingénierie des procédés*. PhD thesis, Université de Toulouse, France, 2008.

Katherine Compton et Scott Hauck. Reconfigurable computing : a survey of systems and software. *ACM Comput. Surv.*, 34(2) :171–210, jun 2002. ISSN 0360-0300. URL https://doi.org/10.1145/508352.508353.

Ivica Crnkovic, Severine Sentilles, A. Vulgarakis, et Michel R.V. Chaudron. A classification framework for software component models. *IEEE Transactions on Software Engineering*, 37(5) :593–615, 2011. URL https://doi.org/10.1109/TSE.2010.83.

K. Czarnecki et S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3) :621–645, 2006. URL https://doi.org/10.1147/sj.453.0621.

Alberto Rodrigues da Silva. Model-driven engineering : A survey supported by the unified conceptual mode. *Computer Languages, Systems and Structures*, 43 :139–155, 2015. URL https://doi.org/10.1016/j.cl.2015.06.001.

Roukaya Dalbouchi, Chiraz Trabelsi, Majdi Elhajji, et Abdelkrim Zitouni. A model-driven platform for dynamic partially reconfigurable architectures : A case study of a watermarking system. *Micromachines*, 14(2), 2023. ISSN 2072-666X. URL https://doi.org/10.3390/mi14020481.

D. de la Fuente, J. Barba, J. Caba, P. Penil, J. C. Lopez, et P. Sanchez. Building a dynamically reconfigurable system through a high-level development flow. *Languages, Design Methods, and Tools for Electronic System Design : Selected Contributions from FDL 2015*, 385 :51–73, 2016. URL https://doi.org/10.1007/978-3-319-31723-6_3.

Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976. ISBN 013215871X.

Alexis Duhamel. *Development of a dynamic resource allocation controller for partially reconfigurable FPGAs with service guarantee approach*. PhD thesis, Université de Nantes, France, 2022.

F. Duhem, F. Muller, R. Bonamy, et S. Bilavarn. Fortress : a flow for design space exploration of partially reconfigurable systems. *Design Automation for Embedded Systems*, 19 (3) :301–326, 2015. URL https://doi.org/10.1007/s10617-015-9160-2.

F. Duhem, F. Muller, et P. Lorenzini. Methodology for designing partially reconfigurable systems using transaction-level modeling. Dans *Proceedings of the 2011 IEEE Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–7. IEEE, 2011. URL https://doi.org/10.1109/DASIP.2011.6136897.

E. Ebeid, F. Fummi, et D. Quaglia. Hdl code generation from uml/marte sequence diagrams for verification and synthesis. *Design Automation for Embedded Systems*, 19 (3) :277–299, 2015. URL https://doi.org/10.1007/s10617-014-9158-1.

Eclipse Foundation. *Gemini Blueprint*, 2024. [Accessed : 01-08-2024].

S. Esakkirajan, T. Veerakumar, et Badri N. Subudhi. Fir filter design. Dans *Digital Signal Processing : Illustration Using Python*, pages 263–302. Springer Nature Singapore, Singapore, 2024. ISBN 978-981-99-6752-0. URL https://doi.org/10.1007/978-981-99-6752-0_7.

Clement Escoffier, Richard S. Hall, et Philippe Lalanda. ipojo : an extensible service-oriented component framework. Dans *IEEE International Conference on Services Computing (SCC 2007)*, pages 474–481, 2007. URL https://doi.org/10.1109/SCC.2007.74.

Huascar Espinoza, Daniela Cancila, Bran Selic, et Sébastien Gérard. Challenges in combining sysml and marte for model-based design of embedded systems. Dans Richard F. Paige, Alan Hartman, et Arend Rensink, éditeurs, *Model Driven Architecture - Foundations and Applications*, pages 98–113, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-02674-4. URL https://doi.org/10.1007/978-3-642-02674-4_8.

R. S. Fabry. How to design a system in which modules can be changed on the fly. Dans *Proceedings of the 2nd International Conference on Software Engineering*, ICSE '76, page 470–476, Washington, DC, USA, 1976. IEEE Computer Society Press.

Ghenassia Frank. *Transaction-Level Modeling with SystemC : TLM Concepts and Applications for Embedded Systems*. Springer New York, NY, 2005. ISBN 978-0-387-26232-1. URL https://doi.org/10.1007/b137175.

Nissaf Fredj, Yessine Hadj Kacem, et Mohamed Abid. A model driven-based approach for managing unanticipated runtime adaptation of rte systems. Dans *2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 314–319, 2019. URL https://doi.org/10.1109/PDCAT46702.2019.00064.

Nissaf Fredj, Yessine Hadj Kacem, et Mohamed Abid. Runtime uml marte extensions for the design of adaptive rte systems. Dans Ajith Abraham, Aswani Kumar Cherukuri, Patricia Melin, et Niketa Gandhi, éditeurs, *Intelligent Systems Design and Applications*, pages 78–87, Cham, 2020. Springer International Publishing. ISBN 978-3-030-16660-1. URL https://doi.org/10.1007/978-3-030-16660-1_8.

Abdoulaye Gamatié, Sébastien Le Beux, Eric Piel, Rabie Ben Atitallah, Anne Etien, Philippe Marquet, et Jean-Luc Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Trans. Embed. Comput. Syst.*, 10(4), 2011. ISSN 1539-9087. URL https://doi.org/10.1145/2043662.2043663.

Erich Gamma, Richard Helm, Ralph Johnson, et John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN 0201633612.

Enrico Giordano, Federico Di Marco, et Graziano Pravadelli. A model-based design flow for dynamic partial reconfigurable fpgas. Dans *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3099–3103, 2019. URL https://doi.org/10.1109/SMC.2019.8914616.

L. Gong et O. Diessel. Modeling dynamically reconfigurable systems for simulation-based functional verification. Dans *Proceedings of the 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 9–16. IEEE, 2011. URL https://doi.org/10.1109/FCCM.2011.18.

Björn Gottschall, Thomas Preußer, et Akash Kumar. Reloc — an open-source vivado workflow for generating relocatable end-user configuration tiles. Dans *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 211–211, 2018. URL https://doi.org/10.1109/FCCM.2018.00045.

S. Guillet, F. De Lamotte, N. Le Griguer, E. Rutten, G. Gogniat, et J. P. Diguet. Extending uml/marte to support discrete controller synthesis, application to reconfigurable systems-on-chip modeling. *ACM Trans. Reconfigurable Technol. Syst.*, 7(3) :1–17, 2014. URL https://doi.org/10.1145/2629628.

D. Gupta, P. Jalote, et G. Barua. A formal framework for on-line software version change. *IEEE Transactions on Software Engineering*, 22(2) :120–131, 1996. URL https://doi.org/10.1109/32.485222.

Julian Haase, Najdet Charaf, Alexander Groß, et Diana Göhringer. Nc-library : Expanding systemc capabilities for nested reconfigurable hardware modelling. *ACM Trans. Reconfigurable Technol. Syst.*, 17(3), Juillet 2024. ISSN 1936-7406. URL https://doi.org/10.1145/3662001.

Kaj Hanninen, Jukka Maki-Turja, Mikael Nolin, Mats Lindberg, John Lundback, et Kurt-Lennart Lundback. The rubus component model for resource constrained real-time systems. Dans *2008 International Symposium on Industrial Embedded Systems*, pages 177–183, 2008. URL https://doi.org/10.1109/SIES.2008.4577697.

Thomas Hartmann, Assaad Moawad, Francois Fouquet, et Yves Le Traon. The next evolution of mde : a seamless integration of machine learning into domain modeling. *Software and Systems Modeling*, 18(2), 2019. ISSN 1285-1304. URL https://doi.org/10.1007/s10270-017-0600-2.

F. Herrera, I. Ugarte, et E. Villar. Towards automated implementation of adaptive systems from abstract systemc specifications. *Design Automation for Embedded Systems*, 16 (3) :129–160, 2012. URL https://doi.org/10.1007/s10617-012-9099-5.

Anthony A. Hock-koon et Mourad Chabane Oussalah. Toward a conceptual comparison framework between cbse and sose. Dans *CAiSE Forum*, 2012. URL https://api.semanticscholar.org/CorpusID:1219036.

Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, Xuefei Ning, Yuzhe Ma, Haoyu Yang, Bei Yu, Huazhong Yang, et Yu Wang. Machine learning for electronic design automation : A survey. *ACM Trans. Des. Autom. Electron. Syst.*, 26(5), 2021. ISSN 1084-4309. URL https://doi.org/10.1145/3451179.

K. Hölldobler, A. Roth, B. Rumpe, et A. Wortmann. Advances in modeling language engineering. Dans *Model and Data Engineering. MEDI 2017. Lecture Notes in Computer Science*, volume 10563 de *AISC*, pages 3–17. Springer, Cham, 2017. URL https://doi.org/10.1007/978-3-319-66854-3_1.

IEEE. *IEEE Standard for Standard SystemC Language Reference Manual*. IEEE Standards, 2023. URL https://doi.org/10.1109/IEEESTD.2023.10246125.

*ISO/IEC/IEEE 42010 :2022 Systems and software engineering — Architecture description*. ISO, International Organization for Standardization, 2022. URL https://www.iso.org/standard78700.html.

A. Jiménez-Pastor, A. Garmendia, et J. de Lara. Scalable model exploration for model-driven engineering. *The Journal of Systems and Software*, 132 :204–225, 2017. URL http://dx.doi.org/10.1016/j.jss.2017.07.011.

Santiago P. Jácome-Guerrero, Marcelo Ferreira, et Alexandra Corral. Software development tools in model-driven engineering. Dans *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 140–148, 2017. URL https://10.1109/CONISOFT.2017.00024.

Jean-Marc Jézéquel, Benoit Combemale, et Didier Vojtisek. *Ingénierie dirigée par les modèles : Des concepts à la pratique*. Références sciences. Ellipses édition, Janvier 2012. ISBN 978-2-7298-71963.

N. Kahani. A domain-specific language for automatic modeling of real-time embedded systems. Dans *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, ICSE '18, pages 515–517. ACM/IEEE, 2018. URL https://doi.org/10.1145/3183440.3190333.

Selim Kebir. Jacac : An aspect oriented framework for the development of self-adaptive software systems. Dans *2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*, pages 74–80, 2012. URL https://doi.org/10.1109/SETIT.2012.6481893.

J.O. Kephart et D.M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, 2003. URL https://doi.org/10.1109/MC.2003.1160055.

SeongKi Kim et Sang-Yong Han. Performance comparison of dcom, corba and web service. Dans *International Conference on Parallel and Distributed Processing Techniques and Applications*, 2006. URL https://api.semanticscholar.org/CorpusID:16468934.

Michael Kirchhoff, Philipp Kerling, Detlef Streitferdt, et Wolfgang Fengler. A real-time capable dynamic partial reconfiguration system for an application-specific soft-core processor. *International Journal of Reconfigurable Computing*, 2019(1) :4723838, 2019. URL https://doi.org/10.1155/2019/4723838.

A. Kleppe, J. Warmer, et W. Bast. *MDA Explained : The Model Driven Architecture : Practice and Promise*. Object Technology. Addison-wesley longman publishing co. édition, April 2003. ISBN 978-0-321-19442-8.

Wolfgang Klingauf. *Systematic Transaction Level Communication Modeling with SystemC*. PhD thesis, Technical University of Braunschweig, Germany, 2008. URL https://doi.org/10.24355/dbbs.084-200810140200-7.

Dirk Koch. *Partial Reconfiguration on FPGAs : Architectures, Tools and Applications*. Springer New York, NY, 2013. ISBN 978-1-4614-1224-3. URL https://doi.org/10.1007/978-1-4614-1225-0.

George Kornaros, Svoronos Leivadaros, et Filippos Kolimbianakis. Flexible updating of internet of things computing functions through optimizing dynamic partial reconfiguration. *ACM Trans. Embed. Comput. Syst.*, 23(2), Mars 2024. ISSN 1539-9087. URL https://doi.org/10.1145/3643825.

A. Kraas. *On the Automated Derivation of Domain-Specific UML Profiles*. PhD thesis, University of Bamberg, Germany, 2019.

J. Kramer et J. Magee. The evolving philosophers problem : dynamic change management. *IEEE Transactions on Software Engineering*, 16(11) :1293–1306, 1990. URL https://doi.org/10.1109/32.60317.

P. Kukkala, J. Riihimaki, M. Hannikainen, T.D. Hamalainen, et K. Kronlof. Uml 2.0 profile for embedded system design. Dans *Design, Automation and Test in Europe*, pages 710–715 Vol. 2, 2005. URL https://doi.org/10.1109/DATE.2005.321.

André Lalevée, Pierre-Henri Horrein, Matthieu Arzel, Michael Hübner, et Sandrine Vaton. Autoreloc : Automated design flow for bitstream relocation on xilinx fpgas. Dans *2016 Euromicro Conference on Digital System Design (DSD)*, pages 14–21, 2016. URL https://doi.org/10.1109/DSD.2016.92.

M. Leite et M. A. Wehrmeister. System-level design based on uml/marte for fpga-based embedded real-time systems. *Design Automation for Embedded Systems*, 20(2) :127–153, 2016. URL https://doi.org/10.1007/s10617-016-9172-6.

JD. Lesage et B. Raffin. A hierarchical component model for large parallel interactive applications. *Journal of Supercomputing*, 60 :389–409, 2012. URL https://doi.org/10.1007/s11227-008-0228-7.

Hugh Maaskant. A robust component model for consumer electronic products. Dans Peter van der Stok, éditeur, *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, pages 167–192. Springer Netherlands, Dordrecht, 2005. ISBN 978-1-4020-3454-1. URL https://doi.org/10.1007/1-4020-3454-7_7.

Amina Magdich, Yessine Hadj Kacem, Bouthaina Dammak, Adel Mahfoudhi, et Mohamed Abid. From dynamic uml/marte models to early schedulability analysis of rtes with dependent tasks. Dans Ajith Abraham, Aswani Kumar Cherukuri, Patricia Melin, et Niketa Gandhi, éditeurs, *Intelligent Systems Design and Applications*, pages 192–201, Cham, 2020. Springer International Publishing. ISBN 978-3-030-16660-1. URL https://doi.org/10.1007/978-3-030-16660-1_19.

Scott Malabarba, Raju Pandey, Jeff Gragg, Earl Barr, et J. Fritz Barnes. Runtime support for type-safe dynamic java classes. Dans Elisa Bertino, éditeur, *ECOOP 2000 - Object-Oriented Programming*, pages 337–361, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45102-0.

Jim Marino et Michael Rowley. *Understanding SCA (Service Component Architecture)*. Addison-Wesley Professional, 1st édition, 2009. ISBN 0321515080.

Grant Martin, Brian Bailey, et Andrew Piziali. *ESL Design and Verification : A Prescription for Electronic System Level Methodology*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. ISBN 9780080488837.

Tom Mens et Pieter Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152 :125–142, 2006. URL https://doi.org/10.1016/j.entcs.2005.10.021.

Bertrand Meyer. *Object-oriented software construction (2nd ed.)*. Prentice-Hall, Inc., USA, 1997. ISBN 0136291554.

Mansureh Shahraki Moghaddam, Jae-Min Cho, et Kiyoung Choi. Reconfigurable architectures. Dans Soonhoi Ha et Jürgen Teich, éditeurs, *Handbook of Hardware/Software Codesign*, pages 335–376. Springer Netherlands, Dordrecht, 2017. URL https://doi.org/10.1007/978-94-017-7267-9_12.

Brice Morin, Olivier Barais, Gregory Nain, et Jean-Marc Jezequel. Taming dynamically adaptive systems using models and aspects. Dans *2009 IEEE 31st International Conference on Software Engineering*, pages 122–132, 2009. URL https://doi.org/10.1109/ICSE.2009.5070514.

P-A. Muller, F. Fondement, B. Baudry, et B. Combemale. Modeling modeling modeling. *Software and Systems Modeling*, 11 :347–359, 2012. URL https://doi.org/10.1007/s10270-010-0172-x.

Gunter Mussbacher, Daniel Amyot, Ruth Breu, Jean-Michel Bruel, Betty H. C. Cheng, Philippe Collet, Benoit Combemale, Robert B. France, Rogardt Heldal, James Hill, Jörg Kienzle, Matthias Schöttle, Friedrich Steimann, Dave Stikkolorum, et Jon Whittle. The relevance of model-driven engineering thirty years from now. Dans *Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E. (eds) Model-Driven Engineering Languages and Systems. MODELS 2014. Lecture Notes in Computer Science*, volume 8767 de *LNCS*, pages 183–200. Springer, Cham, 2014. URL https://doi.org/10.1007/978-3-319-11653-2_12.

Mohamed Naija et Samir Ben Ahmed. Extending uml/marte-sam for integrating adaptation mechanisms in scheduling view. Dans *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering*, ENASE 2016, page 84–90, Setubal, PRT, 2016. SCITEPRESS - Science and Technology Publications, Lda. ISBN 9789897581892. URL https://doi.org/10.5220/0005822300840090.

OASIS. *Service Component Architecture Assembly Model Specification Version 1.1*, 2009. URL https://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.pdf.

*UML Profile for System on a Chip (SoC)*. Object Management Group (OMG), 2006. URL https://www.omg.org/spec/SoCP/1.0.1/About-SoCP. Version 1.0.1.

*Unified Modeling Language (UML) Specification*. Object Management Group (OMG), 2017. URL https://www.omg.org/spec/UML/.

*OMG System Modeling Language (SysML)*. Object Management Group (OMG), 2023a. URL https://www.omg.org/spec/SysML/2.0/Beta1/About-SysML. Version 2.0 beta.

*UML Profile for MARTE : Modeling and Analysis of Real-Time Embedded Systems*. Object Management Group (OMG), 2023b. URL https://www.omg.org/spec/MARTE/1.3/About-MARTE. Version 1.3.

Gilberto Ochoa-Ruiz, Sébastien Guillet, Florent De Lamotte, Eric Rutten, El-Bay Bourennane, Jean-Philippe Diguet, et Guy Gogniat. An mde approach for rapid prototyping and implementation of dynamic reconfigurable systems. *ACM Trans. Des. Autom. Electron. Syst.*, 21(1) :1–25, 2015. URL https://doi.org/10.1145/2800784.

OMG. *CORBA Component Model Specification*, 2006. URL https://www.omg.org/spec/CCM/4.0/PDF.

P. Oreizy, N. Medvidovic, et R.N. Taylor. Architecture-based runtime software evolution. Dans *Proceedings of the 20th International Conference on Software Engineering*, pages 177–186, 1998. URL https://doi.org/10.1109/ICSE.1998.671114.

OSGi Alliance. *Declarative Services Specification*, 2019. [Accessed : 2024-08-01].

Kyprianos Papadimitriou, Apostolos Dollas, et Scott Hauck. Performance of partial reconfiguration in fpga systems : A survey and a cost model. *ACM Trans. Reconfigurable Technol. Syst.*, 4(4), dec 2011. ISSN 1936-7406. URL https://doi.org/10.1145/2068716.2068722.

M.P. Papazoglou. Service-oriented computing : concepts, characteristics and directions. Dans *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*, pages 3–12, 2003.

Luca Pezzarossa, Andreas Toftegaard Kristensen, Martin Schoeberl, et Jens Sparsø. Using dynamic partial reconfiguration of fpgas in real-time systems. *Microprocessors and Microsystems*, 61 :198–206, 2018. ISSN 0141-9331. URL https://doi.org/10.1016/j.micpro.2018.05.017.

Luca Pezzarossa, Martin Schoeberl, et Jens Sparsø. A controller for dynamic partial reconfiguration in fpga-based real-time systems. Dans *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 92–100, 2017. URL https://doi.org/10.1109/ISORC.2017.3.

X. Peña, F. Rincon, J. Dondo, J. Caba, et J. C. Lopez. Run-time partial reconfiguration simulation framework based on dynamically loadable components. Dans *Proceedings of the 11th International Symposium of Applied Reconfigurable Computing (ARC 2015)*, volume 9040, pages 153–164. Springer, Cham, 2015. URL https://doi.org/10.1007/978-3-319-16214-0_13.

Martin Pfannemüller, Martin Breitbach, Christian Krupitzer, Markus Weckesser, Christian Becker, Bradley Schmerl, et Andy Schürr. React : A model-based runtime environment for adapting communication systems. Dans *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 65–74, 2020. URL https://doi.org/10.1109/ACSOS49614.2020.00027.

Jason Porter, Daniel A. Menascé, et Hassan Gomaa. A decentralized approach for discovering runtime software architectural models of distributed software systems. *Information and Software Technology*, 131 :106476, 2021. ISSN 0950-5849. URL https://doi.org/10.1016/j.infsof.2020.106476.

I. R. Qadri, H. Yu, A. Gamatie, E. Rutten, S. Meftali, et J. L. Dekeyser. Targeting reconfigurable fpga based socs using the uml marte profile : from high abstraction levels to code generation. *International Journal of Embedded Systems*, 4(3-4) :204–224, 2010. URL https://doi.org/10.1504/IJES.2010.039025.

I.R. Quadri. *MARTE based model driven design methodology for targeting dynamically reconfigurable FPGA based SoCs*. PhD thesis, Université de Lille, France, 2010. URL https://tel.archives-ouvertes.fr/tel-00486483v2.

A. Raabe et A. Felke. A systemc language extension for high-level reconfiguration modeling. Dans *Proceedings of the 2008 Forum on Specification, Verification and Design Languages (FDL 2008)*, pages 55–60. IEEE, 2008. URL https://doi.org/10.1109/FDL.2008.4641421.

B. Radunovic. An overview of advances in reconfigurable computing systems. Dans *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32.*, volume Track3, pages 10 pp.–, 1999. URL https://doi.org/10.1109/HICSS.1999.772878.

B. Radunovic et V. Milutinovic. A survey of reconfigurable computing architectures. Dans Reiner W. Hartenstein et Andres Keevallik, éditeurs, *Field-Programmable Logic and Applications From FPGAs to Computing Paradigm*, pages 376–385, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-68066-6. URL https://doi.org/10.1007/BFb0055265.

Jens Rettkowski, Konstantin Friesen, et Diana Göhringer. Repabit : Automated generation of relocatable partial bitstreams for xilinx zynq fpgas. Dans *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2016. URL https://doi.org/10.1109/ReConFig.2016.7857186.

F. G. C. Ribeiro, A. Rettberg, C. E. Pereira, S. S. C. Botelho, et M. S. Soares. Guidelines for using marte profile packages considering concerns of real-time embedded systems. Dans *Proceedings of the IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 917–922. IEEE, 2017. URL https://doi.org/10.1109/INDIN.2017.8104894.

E. Riccobene, P. Scandurra, A. Rosti, et S. Bocchio. A uml 2.0 profile for systemc : toward high-level soc design. Dans *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT '05, page 138–141, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930914. URL https://doi.org/10.1145/1086228.1086254.

Alfonso Rodríguez, Juan Valverde, Jorge Portilla, Andrés Otero, Teresa Riesgo, et Eduardo De la Torre. Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems : The artico3 framework. *Sensors*, 18(6), 2018. ISSN 1424-8220. URL https://www.mdpi.com/1424-8220/18/6/1877.

Guillaume Rousseau, Roberto Di Cosmo, et Stefano Zacchiroli. Software provenance tracking at the scale of public source code. *Empirical Software Engineering*, 25 :2930–2959, 2020. URL https://doi.org/10.1007/s10664-020-09828-5.

Romain Rouvoy, Frank Eliassen, Jacqueline Floch, Svein Hallsteinsen, et Erlend Stav. Composing components and services using a planning-based adaptation middleware. Dans Cesare Pautasso et Éric Tanter, éditeurs, *Software Composition*, pages 52–67, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

Subhabrata Roy et Abhijit Chandra. A survey of fir filter design techniques : Low-complexity, narrow transition-band and variable bandwidth. *Integration*, 77 :193–204, 2021. ISSN 0167-9260. URL https://www.sciencedirect.com/science/article/pii/S0167926020303102.

Simon Rädler, Luca Berardinelli, Karolin Winter, Abbas Rahimi, et Stefanie Rinderle-Ma. Bridging mde and ai : a systematic review of domain-specific languages and model-driven practices in ai software systems engineering. *Software and Systems Modeling*, 2024. URL https://doi.org/10.1007/s10270-024-01211-y.

Mazeiar Salehie et Ladan Tahvildari. Self-adaptive software : Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2), may 2009. ISSN 1556-4665. URL https://doi.org/10.1145/1516533.1516538.

Mohamed Oussama Ben Salem, Olfa Mosbahi, Mohamed Khalgui, et Georg Frey. R-uml : An uml profile for verification of flexible control systems. Dans Pascal Lorenz, Jorge Cardoso, Leszek A. Maciaszek, et Marten van Sinderen, éditeurs, *Software Technologies*, pages 118–136, Cham, 2016. Springer International Publishing. ISBN 978-3-319-30142-6. URL https://doi.org/10.1007/978-3-319-30142-6_7.

Andreas Schallenberg, Wolfgang Nebel, Andreas Herrholz, Philipp A. Hartmann, et Frank Oppenheimer. Osss+r : a framework for application level modelling and synthesis of reconfigurable systems. Dans *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, page 970–975, Leuven, BEL, 2009. European Design and Automation Association. ISBN 9783981080155.

M.E. Segal et O. Frieder. Dynamic program updating in a distributed computer system. Dans *Proceedings. Conference on Software Maintenance, 1988.*, pages 198–203, 1988. URL https://doi.org/10.1109/ICSM.1988.10162.

Lionel Seinturier, Philippe Merle, Damien Fournier, Nicolas Dolet, Valerio Schiavoni, et Jean-Bernard Stefani. Reconfigurable sca applications with the frascati platform. Dans *2009 IEEE International Conference on Services Computing*, pages 268–275, 2009. URL https://doi.org/10.1109/SCC.2009.27.

Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, et Jean-Bernard Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Software : Practice and Experience*, 42(5) :559–583, 2012. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.1077.

B. Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5) :19–25, 2003. URL https://doi.org/10.1109/MS.2003.1231146.

Bran Selić et Sébastien Gérard. *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE*. The mk/omg press édition, 2014. ISBN 978-0-12-416619-6.

Biruk Seyoum, Marco Pagani, Alessandro Biondi, et Giorgio Buttazzo. Automating the design flow under dynamic partial reconfiguration for hardware-software co-design in fpga soc. Dans *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, page 481–490, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450381048. URL https://doi.org/10.1145/3412841.3441928.

Afreen Siddiqi et Olivier L. de Weck. Modeling methods and conceptual design principles for reconfigurable systems. *Journal of Mechanical Design*, 130(10) :101102, 09 2008. ISSN 1050-0472. URL https://doi.org/10.1115/1.2965598.

Ha Soonhoi et Teich Jürgen. *Handbook of Hardware/Software Codesign*. Springer Dordrecht., 2017. ISBN 978-94-017-7266-2. URL https://doi.org/10.1007/978-94-017-7267-9.

E. Sotiriou-Xanthopoulos, K. Siozios, G. Economakos, et D. Soudris. A process-based reconfigurable systemc module for simulation speedup. Dans *Proceedings of the 2013 International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS)*, pages 72–79. IEEE, 2013. URL https://doi.org/10.1109/SAMOS.2013.6621108.

Moujtahid Soukaina, Belangour Abdessamad, et Marzak Abdelaziz. Model driven engineering (mde) tools : A survey. *American Journal of Science, Engineering and Technology*, 3(2) :29–33, 2018. URL https://doi.org/10.11648/j.ajset.20180302.11.

E. Suvorova, N. Matveeva, A. Rabin, et V. Rozanov. System level modeling of dynamic reconfigurable system-on-chip. Dans *Proceedings of the 2015 17th Conference of Open Innovations Association (FRUCT)*, pages 222–229. IEEE, 2015. URL https://doi.org/10.1109/FRUCT.2015.7117996.

E. Syriani, L. Luhunu, et H. Sahraoui. Systematic mapping study of template-based code generation. *Computer Languages, Systems & Structures*, 52 :43–62, 2018. URL https://doi.org/10.1016/j.cl.2017.11.003.

Clemens Szyperski. *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd édition, 2002. ISBN 0201745720.

Richard N. Taylor, Nenad Medvidovic, et Eric Dashofy. *Software Architecture : Foundations, Theory, and Practice*. Wiley, 2009. ISBN 978-0-470-16774-8.

Richard N. Taylor et Andre van der Hoek. Software design and architecture the once and future focus of software engineering. Dans *Future of Software Engineering (FOSE '07)*, pages 226–243, 2007. URL https://doi.org/10.1109/FOSE.2007.21.

Grotker Thorsten, Liao Stan, Martin Grant, et Swan Stuart. System design with systemc. *Microelectronics Reliability*, 43(4) :683–684, 2003. ISSN 0026-2714. URL https://doi.org/10.1016/S0026-2714(03)00032-5.

T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk, et P.Y.K. Cheung. Reconfigurable computing : architectures and design methods. Dans *Computers and Digital Technique*, volume 152, pages 193–207. IEE, 2005. URL https://doi.org/10.1049/ip-cdt:20045086.

Maamar Touiza, Gilberto Ochoa-Ruiz, El-Bay Bourennane, Abderrezak Guessoum, et Kamel Messaoudi. A novel methodology for accelerating bitstream relocation in partially reconfigurable systems. *Microprocessors and Microsystems*, 37(3) :358–372, 2013. ISSN 0141-9331. URL https://doi.org/10.1016/j.micpro.2012.07.004.

Chiraz Trabelsi. *Contrôle matériel des systèmes partiellement reconfigurables sur FPGA : de la modélisation à l'implémentation*. PhD thesis, Université des Sciences et Technologies de Lille, France, 2013.

S. R. Koteshwar V. S. Chakravarthi. *System on Chip (SOC) Architecture*. Springer Cham, August 2023. ISBN 978-3-031-36241-5. URL https://doi.org/10.1007/978-3-031-36242-2.

Giacomo Valente, Tania Di Mascio, Luigi Pomante, et Gabriella D'Andrea. Dynamic partial reconfiguration profitability for real-time systems. *IEEE Embedded Systems Letters*, 13(3) :102–105, 2021. URL https://doi.org/10.1109/LES.2020.3004302.

P. Vallejo. *Réutilisation de composants logiciels pour l'outillage de DSML dans le contexte des MPSoC*. PhD thesis, Université de Bretagne Occidentale, France, 2015.

M. van Amstel, S. Bosems, I. Kurtev, et L. Ferreira Pires. Performance in model transformations : Experiments with atl and qvt. Dans *Proceedings of the 4th international conference on Theory and practice of model transformations (ICMT'11)*, volume 6707 de *LNPSE*, pages 198–212. Springer Berlin Heidelberg, 2011. URL https://doi.org/10.1007/978-3-642-21732-6_14.

J. Vidal, F. de Lamotte, G. Gogniat, J. P. Diguet, et S. Guillet. Dynamic applications on reconfigurable systems : From uml model to fpga implementation. Dans *Proceedings of the 2011 Design, Automation and Test in Europe (DATE 2011)*, volume 9040, pages 1–4. IEEE, 2011. URL https://doi.org/10.1109/DATE.2011.5763315.

A. Vignaga. Metrics for measuring atl model transformations. Rapport technique, Universidad de Chile, 2009. URL https://www.dcc.uchile.cl/TR/2009/TR_DCC-20090430-006.pdf.

Kizheppatt Vipin et Suhaib A. Fahmy. Mapping adaptive hardware systems with partial reconfiguration using copr for zynq. Dans *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 1–8, 2015. URL https://doi.org/10.1109/AHS.2015.7231169.

Kizheppatt Vipin et Suhaib A. Fahmy. Fpga dynamic and partial reconfiguration : A survey of architectures, methods, and applications. *ACM Comput. Surv.*, 51(4), jul 2018. ISSN 0360-0300. URL https://doi.org/10.1145/3193827.

Qianxiang Wang, Junrong Shen, Xiaopeng Wang, et Hong Mei. A component-based approach to online software evolution. *Journal of Software Maintenance and Evolution : Research and Practice*, 18(3) :181–205, 2006. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.324.

Yuanlong Xiao, Dongjoon Park, Zeyu Jason Niu, Aditya Hota, et André Dehon. Exhipr : Extended high-level partial reconfiguration for fast incremental fpga compilation. *ACM Trans. Reconfigurable Technol. Syst.*, 17(2), mar 2024. ISSN 1936-7406. URL https://doi.org/10.1145/3617837.

Xilinx Inc. *Zynq-7000 SoC Technical Reference Manual*, 2018. URL https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf. UG585 (v1.12.2).

Xilinx Inc. *Vivado Design Suite User Guide : Dynamic Function eXchange*. Xilinx, 2024. URL https://docs.amd.com/viewer/book-attachment/EaOg7VinYmGITgsfaOacvg/QjemFKCjiZOkVXVBbuzQZA-EaOg7VinYmGITgsfaOacvg. UG909 (v2024.2), December 13, 2024.

Rafael Zamacola, Alberto García Martínez, Javier Mora, Andrés Otero, et Eduardo de La Torre. Impress : Automated tool for the implementation of highly flexible partial reconfigurable systems with xilinx vivado. Dans *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2018. URL https://doi.org/10.1109/RECONFIG.2018.8641703.

Rafael Zamacola, Andrés Otero, Alberto García, et Eduardo De La Torre. An integrated approach and tool support for the design of fpga-based multi-grain reconfigurable systems. *IEEE Access*, 8 :202133–202152, 2020. URL https://doi.org/10.1109/ACCESS.2020.3036541.

I. Zečević, P. Bjeljac, B. Perišić, V. Maruna, et D. Venus. Domain-specific modeling environment for developing domain specific modeling languages as lightweight general purpose modeling language extensions. Dans *Recent Advances in Information Systems and Technologies. WorldCIST 2017. Advances in Intelligent Systems and Computing*, volume 569 de *AISC*, pages 183–200. Springer, Cham, 2017. URL https://doi.org/10.1007/978-3-319-56535-4_85.

Qiang Zhu, Ryosuke Oishi, Takashi Hasegawa, et Tsuneo Nakata. System-on-chip validation using uml and cwl. Dans *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '04, page 92–97, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 158113 9373. URL https://doi.org/10.1145/1016720.1016745.

**ملخص** : في ظل التعقيد المتزايد الذي يميّز عمليات تصميم الأنظمة على الرقاقة (SoC) القابلة لإعادة التكوين الديناميكي والجزئي، تبرز الحاجة الملحّة إلى رفع مستوى التجريد المستخدم في نمذجة هذه الأنظمة. إذ يُسهم هذا المستوى العالي من التجريد في تجاوز القيود المرتبطة بتفاصيل التنفيذ ذات المستوى المنخفض. وفي هذا السياق، تبرز الهندسة الموجهة بالنماذج (IDM) كمنهجية قوية، توفّر إطارًا رسميًا لإنشاء النماذج البنيوية، وتحويلها، وتنقيحها تدريجيًا عبر مختلف مستويات التجريد. وتنسجم هذه الدراسة مع هذا التوجه من خلال اعتمادها على مستوى النظام الإلكتروني (ESL) ، وتحديدًا عند طبقة النمذجة على المستوى التبادلي (TLM). حيث تقترح إطارًا نمذجيًا يدعم التوليد التلقائي لشفرة SystemC/TLM انطلاقًا من نماذج موسّعة UML/MARTE . ولمعالجة غياب الدعم الأصلي لإعادة التكوين الديناميكي الجزئي (DPR) في كل من MARTE و SystemC ، تقترح هذه المقاربة توسيعًا لملف التعريف يشمل نماذج نمطية مخصصة، وبنى تركيبية، وقيمًا معنونة مناسبة، منظمة ضمن ثلاثة ملفات فرعية : MARTE4DPR ، MARTE4SCTLM و MARTE4AF . وقد أظهرت عملية التحقق التجريبية، التي أُجريت على نظام Crossover نشط وقابل لإعادة التكوين بثلاث قنوات، انخفاضًا ملحوظًا في زمن التصميم، وتحسنًا كبيرًا في الإنتاجية مقارنة بالأساليب اليدوية التقليدية.

**Résumé :** Face à la complexité croissante des processus de conception des systèmes sur puce (SoC) dynamiquement et partiellement reconfigurables, il devient essentiel d'élever le niveau d'abstraction utilisé pour leur modélisation. Un tel niveau d'abstraction permet de s'affranchir des contraintes liées aux détails d'implémentation bas niveau. Dans ce contexte, l'Ingénierie Dirigée par les Modèles (IDM) s'impose comme une approche méthodologique robuste, offrant un cadre formel pour la création, la transformation et le raffinement progressif de modèles structurés à différents niveaux d'abstraction. Ce travail s'inscrit dans cette démarche en se plaçant au niveau système (ESL), et plus spécifiquement au niveau transactionnel (TLM). Il propose un framework de modélisation facilitant la génération automatique de code SystemC/TLM à partir de modèles UML/MARTE étendus. Afin de pallier l'absence de support natif de la Reconfiguration Dynamique Partielle (DPR) dans MARTE et SystemC, une extension du profil est introduite, intégrant des stéréotypes, constructions et valeurs étiquetées spécifiques, organisés en trois sous-profils : MARTE4DPR, MARTE4SCTLM et MARTE4AF. La validation expérimentale, conduite sur un système crossover actif 3-voies reconfigurable, montre un gain significatif en temps de conception et une amélioration notable de la productivité par rapport aux méthodes manuelles.

**Abstract:** In light of the growing complexity of the design processes for dynamically and partially reconfigurable systems-on-chip (SoC), it becomes crucial to raise the abstraction level used for their modeling. Such a level of abstraction helps to overcome the constraints associated with low-level implementation details. In this context, Model-Driven Engineering (MDE) stands out as a robust methodological approach, providing a formal framework for the creation, transformation, and progressive refinement of structured models across various abstraction levels. This research aligns with this paradigm by operating at the Electronic System Level (ESL), and more specifically at the Transaction-Level Modeling (TLM) layer. It introduces a modeling framework that supports the automatic generation of SystemC/TLM code from extended UML/MARTE models. To address the lack of native support for Dynamic Partial Reconfiguration (DPR) in both MARTE and SystemC, a profile extension is proposed, incorporating dedicated stereotypes, constructs, and tagged values, organized into three sub-profiles: MARTE4DPR, MARTE4SCTLM, and MARTE4AF. Experimental validation, carried out on a reconfigurable 3-way active crossover system, demonstrates a significant reduction in design time and a substantial improvement in productivity compared to traditional manual methods.