Université d'Evry
Laboratoire d'Ingénierie Dirigée par les Modèles
des Systèmes Temps Réel Embarqués

Thèse

présentée en première version en vue d'obtenir le grade de
Docteur, spécialité « Informatique »

par

Frédéric Thomas

Contribution à la prise en compte des plates-formes logicielles d'exécution dans une ingénierie générative dirigée par les modèles

Thèse soutenue le 21 novembre 2008 devant le jury composé de :

M.	Jean Bézivin	LINA	(Rapporteur)
M.	Jean-Philippe Babau	LISyC	(Rapporteur)
M.	Patrick Albert	ILOG	(Invité)
M.	François Terrier	LISE	(Directeur)
M.	Sébastien Gérard	LISE	(Encadrant)
M.	Jérôme Delatour	ESEO	(Encadrant)

Remerciements

Je ne sais pas s'il existe des statistiques sur le sujet, mais il me semble que la page la plus lue dans une thèse est celle des remerciements. L'exercice est donc difficile. En espérant qu'ils ne soit pas trop rébarbatifs, voici les remerciements de cette thèse.

Contrairement au ressenti d'un doctorant à certaines étapes de ce projet de trois ans : ***Maman, je suis perdu !***, une thèse n'est pas le fruit d'une seule tête pensante, mais d'un ensemble de têtes qui échangent, doutent, rigolent et même parfois font la fête ensembles. Mes premiers remerciements sont donc destinés à mes encadrants : Séb et Jérôme avec qui j'ai eu plaisir à travailler. Jérôme a été le premier à me faire confiance. Séb m'a permis de découvrir la recherche internationale. J'ai eu la chance d'être encadré et soutenu par vous deux tout au long de mon doctorat. Je tiens donc à vous remercier pour vos compétences techniques, votre disponibilité et vos nombreuses qualités humaines.

Outre Sébastien Gérard et Jérôme Delatour, je remercie également François Terrier pour avoir été mon directeur de thèse et m'avoir accueilli dans son Laboratoire d'Ingénierie Dirigée par les Modèles des Systèmes Temps Réel Embarqués (LISE) du CEA Saclay. Merci, à toi, François pour m'avoir fourni les moyens matériels et financiers pour réaliser ce projet.

Même si la rédaction est un but en soi, ce n'est pas le but final d'un doctorant. Je remercie donc également les membres de mon jury qui ont accepté d'assister à ma soutenance. Ainsi, je remercie Jean-philippe Babau et Jean Bézivin pour avoir étudié et évalué ce manuscrit. Leurs retours m'ont permis d'avancer dans ma réflexion. Je remercie également Patrick Albert pour avoir participé au jury en apportant ses compétences industrielles de recherche et développement.

Les remerciements ne sont jamais exhaustifs. Pour simplifier, je remercie l'ensemble des équipes LISE du CEA Saclay et TRAME de l'ESEO. Je remercie tout particulièrement : Arnaud, Brahim, Damien, François, Fred, Guillaume, Gonzague, Huascar, Matthias, Rémy, Romain, Safouan, les Sebs, Wassim... Les beautés de discussions, de pauses cafés et de barbecues que nous avons partagés me manqueront.

Enfin, je compte, bien évidemment, remercier mon entourage pour m'avoir soutenu durant ces trois ans, en particulier, Isa et Chloé. Vous m'avez accompagné, supporté et encouragé dans ce long projet. Comme dirait ***Lalé*** : ***Appi*** !

Saclay, le 18 janvier 2009.

Table des matières

Note aux lecteurs

Conventions

Ce mémoire utilise les conventions typographiques et graphiques suivantes :

Conventions typographiques

- Les noms et les sigles des langages de programmation, des méthodes, des outils, et des institutions sont en petites majuscules : Java, ATL, OMG,
- Les noms des constituants d'un métamodèle ou d'un modèle sont en police non- proportionnelle avec empattement : Class, Task,
- Les mots en langue étrangère sont en italique : *design pattern*,
- Le terme «plate-forme» réfère à la plate-forme d'exécution. Les termes «plate-forme» et «plate-forme d'exécution» sont donc utilisés,
- Le sigle UML réfère à la version 2.1 de la norme UML,
- Dans la description des codes sources, une flèche $\searrow$ précise que la ligne a été coupée dans ce document pour des raisons éditoriales. Cette flèche n'apparait pas dans la description finale.

Conventions graphiques

- La couleur bleue utilisée dans les diagrammes UML référence des informations qui n'appartiennent pas au modèle décrit. Ce sont soit des méta-informations, c'est-à-dire des informations renseignées dans le métamodèle, ou soit des commentaires facilitant la compréhension du diagramme,
- La couleur verte utilisée dans les diagrammes référence un élément qui est importé d'un autre package du modèle,
- Dans les illustrations des hypothèses et des besoins de cette étude, une flèche, $\rightarrow$, signifie une décision prise à une étape du raisonnement. Une ligne terminée par un rond, $\multimap$, signifie la non prise en compte du cas pointé,
- Dans les diagrammes de classes, certains niveaux d'héritage sont omis pour faciliter la compréhension des concepts. Le signe $\doteq$ est utilisé pour spécifier que des classes parentes ont été omises.

Plan de lecture

Ce mémoire a été écrit en vue d'un parcours linéaire puisqu'il concrétise une démarche de thèse. Cependant chaque chapitre est développé de façon indépendante, ainsi un lecteur averti et/ou pressé peut commencer la lecture où bon lui semble. Pour faciliter une telle lecture, la figure 1 explicite les dépendances entre les sections de chaque chapitre. De même, le lecteur dispose aussi d'un glossaire, d'une bibliographie et d'un index à la fin de ce mémoire.

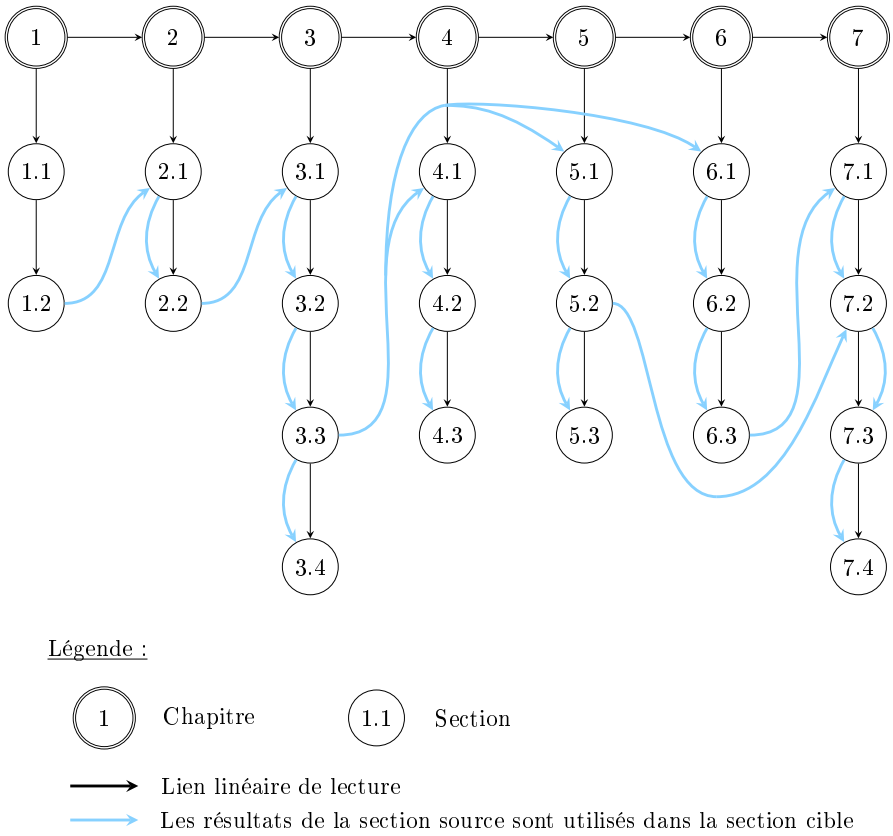

Fig. *1 – Plan de lecture du mémoire*

Introduction

Contexte

Dans le domaine des systèmes embarqués temps réel, des études industrielles empiriques constatent que l'empreinte mémoire des logiciels croit exponentiellement depuis le début des années quatre-vingt [1, 2]. Cette augmentation s'explique, en partie, par l'élargissement du champ d'application des systèmes. Du contrôleur d'injection à la navigation par satellite, les fonctionnalités logicielles se sont en effet largement diversifiées. Cette diversification a inévitablement complexifié les logiciels en eux-mêmes ainsi que celle des cycles de développement [3].

Face à cette complexité, des abstractions logicielles ont été mises en œuvre. Par exemple, des couches logicielles telles que les systèmes d'exploitation temps réel embarqués ont été utilisés. Ces systèmes ont, d'une part, automatisé le portage des applications sur différents supports matériels et, d'autre part, capitalisé des services programmés nécessaires à l'exécution des applications. Ce sont en fin de compte des plates-formes logicielles d'exécution utilisées pour faciliter, automatiser et capitaliser les implantations logicielles.

Bien que les plates-formes logicielles d'exécution aient facilité les étapes d'implantation ; la diversité des solutions qu'elles proposent a paradoxalement complexifié les étapes de conception. Ces étapes requièrent désormais une connaissance accrue des mécanismes et des services fournis par les plates-formes ainsi qu'une maitrise poussée de leurs mises en œuvre. Cette connaissance et cette maitrise constituent alors le savoir et le savoir-faire [4] nécessaires au développement des applications logicielles temps réel embarquées.

Les solutions classiques de programmation ne permettent pas de capitaliser à la fois le savoir et le savoir-faire. Ainsi, les cinq instructions génériques que sont l'affectation, la déclaration, la séquence, le test et la boucle [5] ne capturent pas aisément les différentes approches d'utilisation des mécanismes et des services fournis par les plates-formes d'exécution. L'ingénierie dirigée par les modèles (IDM) [6], discipline informatique émergente, vise justement à proposer un cadre méthodologique et technologique pour faciliter la manipulation de ce savoir et de ce savoir-faire. Pour cela, elle prône l'utilisation de modèles et de transformations de modèles. Les modèles capitalisent le savoir et le savoir-faire. Les transformations automatisent leurs intégrations dans les développements.

Problématique

Dans l'IDM appliqué au temps réel embarqué, des travaux s'intéressent à modéliser les plates-formes logicielles d'exécution [7, 8, 9, 10]. Néanmoins, les concepts qu'ils proposent sont très abstraits. Ils sont souvent distants des mécanismes et des services réellement fournis par les plates-formes. Ainsi, pour réduire le fossé d'abstraction entre la modélisation et l'implantation, les ingénieries résultantes sont dédiées à un ensemble de plates-formes cibles. Elles sont figées pour un ensemble de choix d'implantation spécifiques à la plate-forme visée. Les ingénieries produites sont alors efficaces pour un ensemble de préoccupations homogènes, figées et répétitives.

Dans les systèmes embarqués temps réel, les besoins sont certes répétitifs mais ils sont surtout hétérogènes et évolutifs. Par exemple, la production d'une application implique la prise en compte de diverse contraintes (de temps et d'empreinte mémoire par exemple), le respect de plusieurs règles d'implantation et enfin la mise en œuvre de différentes plates-formes d'exécution adéquates à différentes étapes d'un même développement. Toute cette hétérogénéité impose donc aux ingénieries d'être adaptables et réutilisables pour différents contextes de développement, en particulier, pour différentes plates-formes d'exécution. C'est ce qui justifie cette thèse.

Objectifs de recherche

Cette thèse consiste à caractériser la notion de modèle de plate-forme d'exécution dans l'IDM, c'est-à-dire de définir les caractéristiques d'un langage permettant de décrire des modèles de plates-formes d'exécution. Elle a pour objectif d'externaliser les formalismes propres aux plates-formes logicielles d'exécution dans des modèles explicites, c'est-à-dire des modèles de plates-formes explicites. Ces modèles paramètrent alors l'exécution des transformations et non plus leurs descriptions.

Outre une meilleur compréhension des plates-formes d'exécution, cette modélisation et cette intégration explicite sont importantes parce qu'elles doivent permettre de capitaliser et donc de réutiliser les transformations outillant les ingénieries de développement.

Contributions

Cette thèse contribue sur trois points à une ingénierie générative dédiée au temps réel embarqué :

- **La définition du motif Resource-Service** pour décrire l'interface de programmation structurelle des plates-formes logicielles d'exécution. Ce motif est une bonne pratique pour concevoir les métamodèles de plates-formes d'exécution,

- ▷ **La définition d'une extension au langage UML**, nommée Software Resource Modeling, pour la modélisation des plates-formes logicielle d'exécution multitâche. Cette extension applique le motif Resource-Service et contribue au profil UML MARTE[1] dédié à la modélisation et à l'analyse des systèmes temps-réel embarqués [11].
- ▷ **L'expérimentation d'un noyau de transformation** basé sur des modèles explicites de plates-formes et permettant de porter une application sur différentes plates-formes d'exécution.

Démarche

Cette étude débute par la caractérisation d'une ingénierie dirigée par les modèles générative d'application multitâche (chapitre 1). Elle caractérise dans un second temps la notion de plate-forme au sens de la littérature et positionne les contributions de cette thèse vis à vis des travaux existants (chapitre 2). Ensuite, la démarche de cette thèse s'articule en cinq chapitres.

Le chapitre 3 définit les concepts élémentaires et les liens entre ces concepts pour la modélisation structurelle des plates-formes logicielles d'exécution. Ces concepts constituent le motif Resource-Service, c'est-à-dire une bonne pratique de métamodèlisation destinée à la construction des métamodèles de plates-formes d'exécution.

Le chapitre 4 vise à confronter le motif Resource-Service aux cas de modélisation des plates-formes d'exécution. Cette confrontation doit permettre de définir des heuristiques de modélisation et de valider l'adéquation du motif pour la modélisation des plates-formes.

Le chapitre 5 outille le motif Resource-Service afin d'intégrer des modèles explicites de plates-formes d'exécution dans des infrastructures génératives. Il spécifie un ensemble de services dédiés à la manipulation des modèles de plates-formes.

L'objectif du chapitre 6 est d'appliquer le motif Resource-Service pour définir un langage de description de plates-formes logicielles d'exécution multitâche. Ce langage est nommé Software Resource Modeling (SRM). Il est définit et évalué dans ce chapitre.

Enfin, le chapitre 7 intègre les modèles explicites de plates-formes d'exécution dans une ingénierie générative dirigée par les modèles. Il définit une infrastructure de transformation basée sur des modèles de plates-formes explicites. Enfin, il évalue cette infrastructure sur un cas d'étude expérimental.

Les chapitres 4, 5 et 6 constituent la contribution générique de cette thèse. Les chapitres 6 et 7 appliquent ces résultats génériques pour les systèmes multitâches.

[1] MARTE : Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)

Première partie

Caractérisation de la problématique et état de l'art

Caractérisation de la problématique

1

Sommaire

Ce chapitre vise deux objectifs. Il s'intéresse tout d'abord à introduire le contexte de cette étude et le vocabulaire employé. Il vise ensuite à caractériser la problématique de cette thèse.

Pour cela, ce chapitre est divisé en deux sections. La première caractérise les applications considérées, c'est-à-dire les applications multitâches exécutées sur un exécutif temps réel embarqué. La seconde décrit l'ingénierie de développement mise en œuvre, c'est-à-dire l'ingénierie dirigée par les modèles dans laquelle les exécutifs temps réel embarqués doivent être pris en compte.

1.1 Caractérisation des applications mises en œuvre

L'objectif de cette section est de caractériser les applications considérées, c'est-à-dire d'identifier les supports d'exécution impliqués dans le développement des applications temps réel embarquées et de détailler leurs mises en œuvre.

1.1.1 Introduction à l'informatique embarquée temps réel

Définition d'un système embarqué temps réel

Un système temps réel est un système informatique de contrôle qui est connecté à un procédé physique dont il doit suivre et commander le comportement [12]. Pour cela, un système temps réel est principalement constitué de quatre composants : 1) une application logicielle contrôlant le procédé, 2) un support d'exécution matériel et/ou logiciel exécutant l'application, 3) des capteurs fournissant les données en entrée de l'application, et 4) des actionneurs exécutant les ordres produits par l'application. L'application, les plates-formes d'exécution, les capteurs et les actionneurs sont alors embarqués. Ils sont enfouis dans le procédé auquel ils sont connectés si bien que l'utilisateur en oublie leurs présences. Orthogonalement, ils sont temps réel, c'est-à-dire que leurs réactions sont assujetties à la dynamique des stimuli émis par le procédé.

Catégories des systèmes temps réel

L'évaluation d'un système embarqué temps réel ne repose pas uniquement sur l'enchaînement correct de ses actions et sur la justesse des résultats produits, mais également sur la satisfaction d'une multitude de contraintes induites par le procédé telles que des contraintes de sûreté de fonctionnement [13], des contraintes d'enfouissement [14] ou encore des contraintes temporelles [15]. Selon qu'il faille absolument respecter les contraintes ou qu'il soit possible de les violer occasionnellement, ces systèmes sont qualifiés de critiques, fermes ou non critiques. Ainsi, les systèmes critiques nécessitent le respect strict des contraintes. Les systèmes fermes admettent le non respect de certaines contraintes selon des critères définis par le contexte de l'application. Enfin les systèmes non critiques tolèrent une probabilité de contraintes non satisfaites.

Hypothèse 1.1 *Le champ de cette étude porte sur* **les applications temps réel embarquées non critiques**. *Il est admis que les applications ne possèdent pas de modes d'exécution, de contraintes de gestion des erreurs ou de caractéristiques de sûreté de fonctionnement.*

Programmation multitâche

Même si les systèmes considérés sont non critiques, ils n'en restent pas moins qu'ils sont embarqués et temps réel. Par conséquent, les différentes contraintes qu'ils doivent satisfaire induisent des ressources d'exécution en nombre limité, aux capacités limitées et au comportement temporel prédictif. Afin d'en optimiser l'utilisation, une philosophie de mise en œuvre très répandue est la concurrence, ou tout au moins le parallélisme. Elle consiste à exécuter en parallèle des fils d'exécutions séquentielles, c'est-à-dire des tâches. La programmation multitâche est une des réponses à la limitation en fréquence des unités de calculs. Elle permet d'outrepasser les attentes de communication, les temps de calculs, d'améliorer les temps

de réponse du système ou encore de simplifier la mise en œuvre des applications en utilisant des mécanismes de communication entre les entités concurrentes.

La mise en oeuvre d'un système multitâche consiste à programmer les éléments logiciels et à configurer les éléments matériels. Les choix des éléments logiciels et matériels, leurs organisations, leurs descriptions (algorithmes, structures de données) sont décrits dans une phase amont appelée phase de conception. Cette étape de conception produit alors un modèle, c'est-à-dire une abstraction du système, qui est réalisée (implantée) dans des étapes de mise en œuvre matérielles et/ou logicielles.

Mise en œuvre matérielle

La mise en œuvre matérielle des systèmes multitâches se concrétise principalement par quatre types d'architecture : (a) monoprocesseur, (b) multiprocesseur, (c) multicœur ou enfin (d) distribuée. Ainsi, lorsque l'installation est éclatée géographiquement, des architectures distribuées sont utilisées. De même, lorsque une puissance de calcul importante est nécessaire, des architectures multiprocesseurs ou multicœurs sont mises en place pour paralléliser les traitements.

Hypothèse 1.2 *Cette étude se restreint à une* **architecture monoprocesseur**.

La mise en œuvre d'une application multitâche sur une architecture monoprocesseur peut être matérielle et/ou logicielle. Elle est alors dite conjointe («co-design»). En effet, certains travaux tel que [16], visent à implanter certaines fonctionnalités dans des circuits matériels spécialisés comme les ASIC ou les FPGA et les autres fonctionnalités comme des entités logicielles s'exécutant sur le processeur lui-même. Les architectures résultantes sont alors mixtes et les étapes de mise en œuvre sont par conséquent conjointes.

Hypothèse 1.3 *Dans cette étude, il est admis que la plate-forme matérielle est abstraite par la plate-forme logicielle. Toutes les fonctionnalités du cahier des charges de l'application sont supposées être* **mises en œuvre de manière logicielle**.

Mise en œuvre logicielle

L'une des particularités de la mise en œuvre logicielle des systèmes multitâches est l'établissement d'un lien entre l'occurrence d'un événement et le déclenchement du traitement, c'est-à-dire de la tâche associée. Deux approches sont majoritairement utilisées [12] :

► **L'approche synchrone –** La technique de mise en œuvre synchrone consiste à prendre en compte les événements, émis par le procédé, à la fin des exécutions des tâches. Cela revient donc à construire directement le lien entre une tâche et son événement déclencheur sans se soucier d'éventuels conflits avec d'autres tâches en cours de fonctionnement. Pour cela, il est postulé, d'une part, que les durées des traitements et des communications sont nulles vis à vis de la dynamique du procédé contrôlé et, d'autre part, que la réaction du système vis à vis d'un stimulus est immédiate. Les exécutions des tâches et les instants d'observation des stimuli sont alors synchrones et non concurrents. La figure 1.1 page suivante , issue de [12], illustre la mise en œuvre synchrone de deux tâches nommées Directive et

Alarme qui sont exécutées suite aux occurrences des événements D et A. En pratique, cette hypothèse d'instantanéité est vérifiée lors de la programmation des systèmes. Les systèmes synchrones sont programmés sous la forme de systèmes échantillonnés ou la durée d'exécution des tâches est inférieure à la période d'échantillonnage [17].

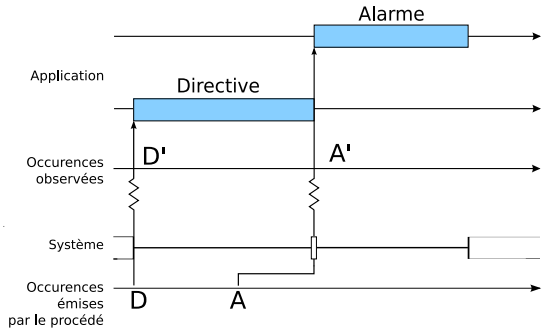

FIG. *1.1 – Mise en œuvre synchrone d'une application multitâche*

► **L'approche asynchrone –** La technique de mise en œuvre asynchrone consiste à observer continuellement l'occurrence d'événements, y compris pendant l'exécution de tâches. Les durées des traitements sont alors non nulles. Partant de cette hypothèse, soit

a) le temps régit les instants d'observation des stimuli et les instants d'observation des réactions du système et

b) les réactions à ces stimuli sont immédiates ce qui peut imposer l'arrêt, la suspension ou le report de la tâche en cours.

La première est une approche cadencée par le temps (Time-Triggered). La seconde est une approche cadencée par les événements (Event-Triggered). Dans les deux cas, la prise en compte des stimuli est asynchrone, parallèle, avec les exécutions des tâches puisque elle n'est pas synchrone avec la fin des exécutions. La figure 1.2 page suivante illustre cette mise en œuvre asynchrone pour les deux tâches Directive et Alarme.

Les capacités de préemption et d'ordonnancement de cette approche requièrent l'utilisation d'exécutifs logiciels dédiés qui sont en charge de gérer les exécutions concurrentes et la prise en compte des stimuli. L'exécutif OASIS [18] pour les approches cadencées par le temps et les exécutifs temps réel OSEK/VDX-OS [19], ARINC-653 [20], VXWORKS [21] pour les approches cadencées par les événements en sont des exemples. Ce sont typiquement des plates-formes logicielles embarquées d'exécution multitâche temps réel.

Hypothèse 1.4 *Une* **approche asynchrone cadencée par les événements** *est supposée satisfaire l'ensemble des exigences du cahier des charges.*

Dans cette étude, la mise en œuvre d'une application temps réel se concrétise donc par l'implantation d'une application multitâche sur une

plate-forme logicielle d'exécution multitâche, c'est-à-dire sur un exécutif temps réel.

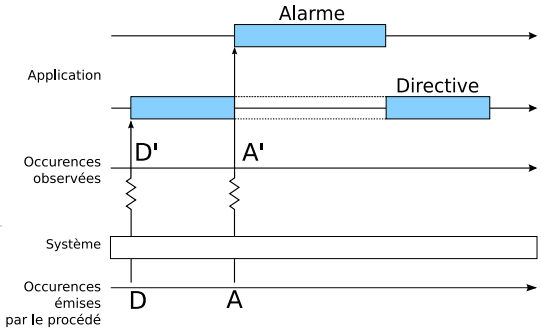

FIG. *1.2 – Mise en œuvre asynchrone d'une application cadencée par les événements*

1.1.2 Caractérisation et mise en œuvre des exécutifs temps réel

Un exécutif temps réel compose un système d'exploitation temps réel, c'est-à-dire un ensemble de programmes logiciels qui gèrent les quatre éléments fondamentaux d'un système informatique : le matériel, le logiciel, la mémoire et les données [22]. Ce système d'exploitation est alors dit temps réel puisqu'il implémente les mécanismes et les services nécessaires à la mise en œuvre des applications tout en garantissant des temps d'exécution finis, bornés et déterministes. Par exemple, il doit assurer que les temps de changement de contexte, les temps de latence à la préemption et en interruption sont constants quelque soit la charge du système.

Contrairement à un système d'exploitation classique, un exécutif temps réel favorise le dialogue ***machine-capteur*** plutôt que le dialogue ***homme-machine***. Pour cela, il est la plupart du temps constitué :

1) d'un noyau qui contient le strict minimum pour la mise en œuvre d'un système multitâche (principalement des fonctionnalités de gestion de tâches, de gestion des interactions et de gestion du temps processeur),
2) d'agences supplémentaires (gestion des interruptions, gestion des entrées/sorties, gestion de la mémoire, etc) déchargeant l'utilisateur d'une programmation souvent fastidieuse. Ces agences sont des librairies préprogrammées fournissant un ensemble de mécanismes et de services.

L'appartenance d'une agence au noyau est influencée par le type d'utilisation, les contraintes techniques, les temps de réponse ou encore les stratégies d'allocation des ressources du système. La délimitation entre noyau, exécutif et système d'exploitation n'est pas toujours facile à déterminer. Ainsi, dans une vision globale (exécutif-noyau) [23], il est possible d'extraire les agences suivantes :

Agences d'un exécutif multitâche temps réel

1) ***Les agences liées aux exécutions concurrentes*** qui regroupent les entités dédiées aux exécutions multitâches,

2) ***Les agences d' interactions*** dédiées aux interactions entre les contextes concurrents d'exécution. Ces interactions sont soit des communications de données ou soit des synchronisations d'exécution,
3) ***Les agences dédiées à l'ordonnancement*** qui orchestrent les exécutions concurrentes et gèrent l'attribution des ressources matérielles d'exécution,
4) ***Les agences des entrées/sorties*** constituées des entités réalisant la gestion des périphériques matériels d'exécution sous-jacents,
5) ***Les agences des horloges*** dédiées à la représentation et la manipulation du temps et des échelles de temps,
6) ***Les agences de gestion des erreurs*** qui proposent des mécanismes pour la gestion des modes de fonctionnement et des exceptions logicielles et
7) ***Les agences d'initialisation*** qui gèrent l'initialisation, le démarrage et le redémarrage de l'exécutif.

Toutes ces agences sont illustrées dans la figure 1.3 .

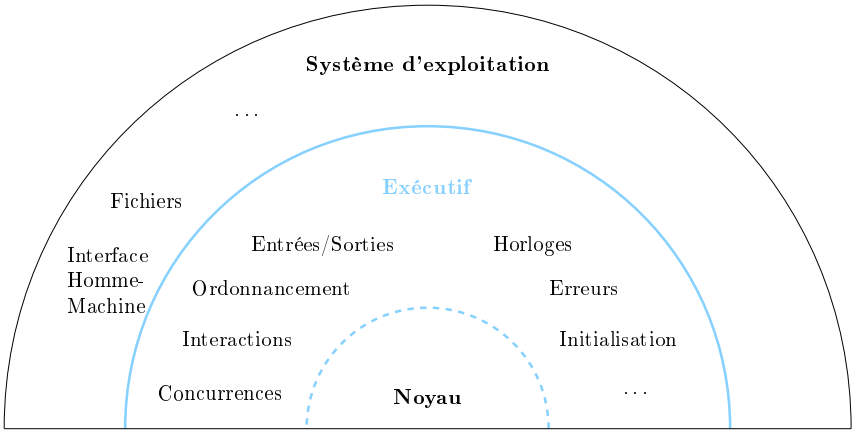

FIG. *1.3 – Architecture générale d'un système d'exploitation temps réel*

Hypothèse 1.5 **Les agences de gestion des exécutions concurrentes, de gestion des interactions, de gestion de l'ordonnancement et de gestion des entrées/sorties** *sont supposées suffisantes pour la mise en place des applications visées par cette étude.*

Dans cette étude les plates-formes considérées sont donc composées de ces quatre agences. Le processus de développement doit donc proposer des artéfacts de modélisation pour intégrer ces agences dans une ingénierie générative dirigée par les modèles.

1.2 Caractérisation de l'ingénierie mise en œuvre

Cette section introduit les concepts de l'Ingénierie Dirigée par les Modèles (IDM) puis identifie la problématique de cette étude vis à vis de la modélisation et de l'intégration d'une plate-forme d'exécution dans un processus de développement dirigé par les modèles.

1.2.1 Introduction à l'Ingénierie Dirigée par les Modèles

L'IDM est une ingénierie générative

L'Ingénierie Dirigée par les Modèles (IDM) vise à proposer un cadre méthodologique et technologique pour faciliter la prise en compte progressive des différents formalismes nécessaires à la production et à la maintenance d'un logiciel.

Pourquoi l'IDM ?

L'IDM n'est pas une rupture technologique vis à vis des solutions de développement. Elle ne vise pas, par exemple, à remplacer les solutions existantes telles que les langages de programmation, les outils de développement et les méthodologies de conception propriétaires. Au contraire, elle vise à les intégrer tout au long du cycle de développement autour d'un socle conceptuel et technologique unifié (composé de modèles et de passerelles entre ces modèles). La plus value espérée est ainsi l'unification des processus de production et de maintenance du logiciel [24].

*Principe et vocabulaire de l'*IDM

Pour cela, le principe même de l'IDM est de capitaliser les savoirs faire non plus au niveau des codes sources mais dans des ***modèles***. Il n'existe pas à ce jour de consensus sur la définition d'un modèle. Des initiatives académiques [6, 25, 26, 27] et industrielles telle que la norme sur les architectures dirigées par les modèles (MDA : Model-Driven Architecture) [28] ont permis de clarifier les concepts de cette discipline informatique émergente. Ainsi, dans la suite, les définitions et les notations établies par Favre [6] sont utilisées.

Définition 1.1 **Un système** *est l'entité à modéliser.*

Définition 1.2 **Un modèle** *est une abstraction d'un système construite dans une intention particulière. Un modèle représente un système. Cette relation de représentation est notée μ*

Pour que les modèles soient utilisables par des outils informatiques, des langages de modélisation doivent être définis. Ces langages permettent alors de décrire les modèles des systèmes. De manière naturelle dans l'IDM, il existe des modèles de ces langages, c'est-à-dire des ***métamodèles***.

Définition 1.3 **Un métamodèle** *est un modèle d'un langage de modélisation. C'est une abstraction d'un langage permettant de décrire des modèles.*

La caractéristique «méta»

En linguistique, le préfixe «méta» est employé dès qu'une opération est appliquée deux fois. Par exemple, une discussion qui définit comment les futurs discussions doivent être menées est une métadiscussion (les interlocuteurs définissent les règles et les concepts nécessaires pour discuter entre eux). Dans l'IDM, un modèle qui vise à définir les règles et les concepts utilisés pour décrire d'autres modèles est un ***métamodèle***. De même, un modèle qui définit les règles et les concepts pour décrire des métamodèles est un ***métamétamodèle***.

Définition 1.4 **Un métamétamodèle** *est un modèle d'un langage de modélisation permettant de décrire des métamodèles.*

La relation entre un élément d'un modèle et un ou plusieurs éléments de son métamodèle est appelée la relation de ***conformité***. Cette relation est notée χ .

Pour atteindre cette caractéristique «méta», il ne faut pas raisonner sur le contenu du modèle, c'est-à-dire sur ce qu'il faut modéliser dans le système. Il faut plutôt raisonner sur les concepts (et les relations entre ces concepts) nécessaires à la description des modèles d'un système [29]. Par exemple, dans la figure 1.4 , l'objectif est la modélisation de figures géométriques imbriquées. Une de ces figures est présentée en partie droite. L'espace technique est donc celui des formes qui peuvent être spécialisées et composées. C'est le métamétamodèle. Dans cet espace, un métamodèle permet de décrire des rectangles, dont certains sont des carrés et d'autres des ronds. Un carré peut posséder un rectangle (par héritage un carré) ou un rond. Ce métamodèle peut être utilisé pour décrire de nombreuses formes et en particulier celles décrites en partie droite. Une modèle de cette figure est par exemple constituée de deux carrés. Un carré est imbriqué dans le premier carré puis un rond est imbriqué dans le second carré. Dans ce modèle, les caractéristiques de couleur ne sont pas décrites puisque le métamodèle ne les capture pas.

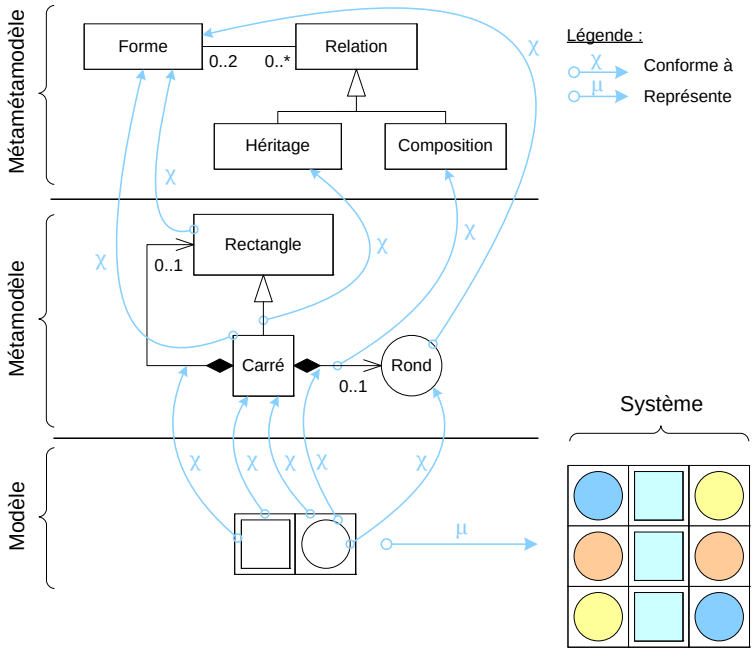

FIG. *1.4 – Illustration des concepts de l'IDM*

La transformation de modèle

Un des intérêts de l'architecture en couche modèle, métamodèle et métamétamodèle, est la capitalisation des transformations entre modèles. Ainsi, en décrivant les transformations à partir des métamodèles sources et cibles et non pas à partir des modèles eux mêmes, celles-ci fonctionnent quels que soit les modèles (conforment à ces métamodèles). Une même transformation peut alors traiter différents modèles sources et produire différents modèles cibles.

Définition 1.5 **Une transformation de modèle** *est une fonction,* $\tau : S \rightarrow T$ *, qui à partir d'un*

ensemble de modèles sources dans S crée un ensemble de modèles cibles dans T. Les ensembles S et T sont des ensembles de modèles conforment à deux ensembles de métamodèles. Si ces deux ensembles de métamodèles sont identiques alors la fonction est endogène, sinon elle est exogène.

Historique des langages de transformations de modèle

La transformation de modèle n'est pas une discipline propre à l'IDM. Ainsi, trois générations (chronologiques) de langages de transformations sont identifiées dans [30] :

1) ***Les langages de transformation de structures séquentielles d'enregistrement***, dans lesquels un script spécifie comment un fichier d'entrée (un modèle source) est réécrit en un fichier de sortie (un modèle cible). Des exemples de ces langages sont les scripts Unix, AWK ou PERL,
2) ***Les langages de transformation d'arbres*** dans lesquels les fragments de l'arbre de sortie (modèle cible) sont générés au cours du parcours de l'arbre d'entrée (modèle source). Ils se basent généralement sur des documents au format XML et l'utilisation de XSLT ou XQUERY,
3) ***Les langages de transformation de graphes*** pour lesquels un modèle en entrée (graphe orienté étiqueté) est transformé en un modèle en sortie. Dans ce cas, la description d'une transformation de modèle sur un ensemble de modèles est décrite par un modèle en soi. Des langages de transformation de modèles tels que ATL (Atlas Transformation Langage) [24] ou le standard du consortium OMG, QVT (Query-View-Transformation) [31], sont des exemples de langages de transformation de graphes. Ils sont associés à des moteurs de transformation tels que, respectivement, la machine virtuelle ATL et l'initiative SMART-QVT™ [32].

Cette étude ne s'intéresse pas à la proposition d'un langage de transformation de graphes et n'aborde pas non plus les problématiques de conception des moteurs de transformation [24].

Hypothèse 1.6 **Les langages de transformations de graphes** *sont utilisés dans le cadre de cette étude.*

Ainsi, dans cette étude, les langages de transformation fournissent des métamodèles de transformation w_τ à partir desquelles des modèles de transformation M_τ sont décrits. Ces métamodèles sont composés de règles de transformation manipulant explicitement les éléments des métamodèles. Ces règles de transformation sont alors spécifiques aux métamodèles sources et cibles. L'exécution des règles de transformation est à la charge d'un moteur de transformation. La description des règles peut être déclarative (l'ordre d'exécution des règles n'est pas défini par l'utilisateur) ou impérative (l'ordre d'exécution des règles est défini par l'utilisateur). La figure 1.5 page suivante illustre la mise en œuvre des transformations de modèles dans cette étude.

Transformation de modèle et génération de code

En principe, les transformations de modèles sont utilisées pour passer de modèles en modèles, modèles qui sont sérialisés dans des technologies spécifiques à l'IDM telle que le XMI par exemple. Lorsqu'il est nécessaire de transformer un modèle en une représentation textuelle (du code), plusieurs familles de langages existent :

1) ***Les langages de transformations de modèles*** générant des chaines de caractères,
2) ***Les langages dédiés à la génération de code*** (également appelé ***langage de template***) tels que ACCELEO [33] et XPAND [34],
3) ***Les langages d'ordre supérieur*** permettant à la fois de générer les générateurs de code et les parseurs. TCS [35] et SINTASK [36] en sont des exemples.

Dans cette étude, toutes ces techniques peuvent être utilisées indifféremment.

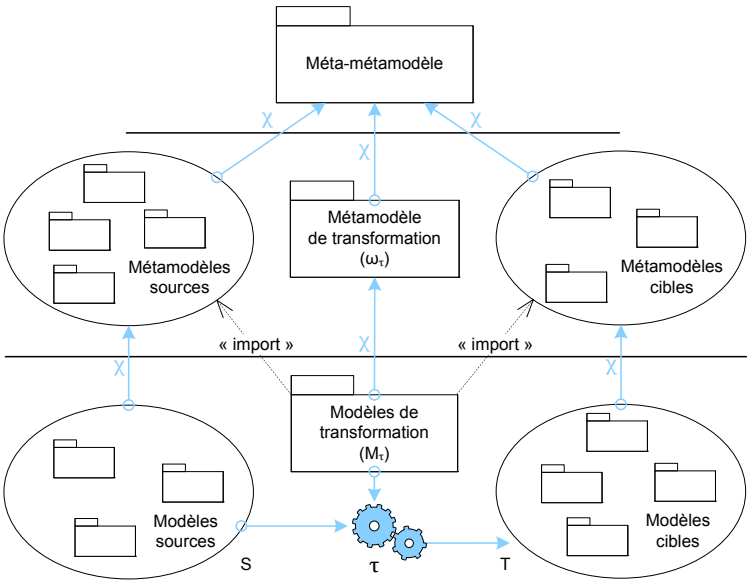

FIG. *1.5 – Mise en œuvre des transformations dans l'IDM*

1.2.2 Mise en œuvre des Ingénieries Dirigées par les Modèles

La mise en œuvre des ingénieries dirigées par les modèles se décompose en deux points principaux : a) la conception de métamodèles, et b) l'intégration de ces métamodèles dans un processus génératif.

Identification des approches de conception des métamodèles

Un métamodèle est une description de la syntaxe abstraite

Un métamodèle est une description de la syntaxe abstraite d'un langage. La spécification de la syntaxe abstraite regroupe l'ensemble des concepts du langage et des liens entre ces concepts. La conception d'un métamodèle se concrétise par deux approches distinctes. La première est la création d'un métamodèle spécifique pour les besoins de développement, c'est-à-dire la définition d'un langage spécifique au domaine (Domain Specific Language (DSL)). La seconde est l'extension d'un métamodèle existant.

Langage Spécifique au Domaine

La création d'un DSL consiste à définir l'ensemble des métaconcepts

et leurs relations pour pouvoir décrire des métamodèles représentatifs du domaine étudié. Cette approche nécessite :

a) ***une expertise en métamodélisation***, c'est-à-dire une expertise dans la définition des métaéléments et des métaliens entre ces éléments,

b) ***une expertise du métier*** pour lequel est créé ce langage,

c) ***la définition d'une syntaxe concrète***, c'est-à-dire la définition d'une représentation textuelle ou graphique manipulée par l'utilisateur pour décrire les modèles et

d) ***la conception d'un ensemble d'outils*** pour manipuler ce langage (par exemple un éditeur, un analyseur syntaxique et un analyseur sémantique).

Extension de métamodèle générique

Dans certains cas, il est plus efficace de réutiliser et d'adapter un métamodèle existant dont les éléments sont génériques à plusieurs contextes de développement plutôt que de développer un nouveau métamodèle à part entière et un nouvel outillage pour manipuler ce métamodèle. Cette approche par extension nécessite :

a) ***une connaissance du métamodèle à étendre***,

b) ***une expertise du métier*** pour lequel est créé cette extension et

c) ***la définition d'une syntaxe concrète*** appropriée (textuelle ou graphique).

Cette approche a pour principal avantage de capitaliser, et donc de réutiliser, les outils, la structure, la sémantique et même parfois la notation graphique du métamodèle étendu. Le langage Unified Modeling Language (UML), standardisé par l'OMG dans [37], est un exemple de langage généraliste que l'utilisateur peut étendre pour des besoins particuliers.

Hypothèse 1.7 **Le langage UML** *est supposé satisfaire l'ensemble des besoins de modélisation.*

En pratique, l'extension UML se concrétise par deux mécanismes majeurs que sont l'extension dite lourde et l'extension dite légère. La technique d'extension lourde manipule le métamodèle étendu en lecture mais aussi en écriture. Typiquement, cette extension permet d'importer des éléments du métamodèle afin d'ajouter, de supprimer ou de modifier leurs caractéristiques et leurs sémantiques. Pour cela, cette technique utilise des opérateurs tels que la fusion, la redéfinition ou la spécialisation par exemple [38]. Ces opérateurs limitent les avantages de réutilisation cités dans le paragraphe précédent. L'extension légère, quant à elle, manipule le métamodèle en lecture seulement, c'est-à-dire que les métaéléments sont importés mais leurs structures, leurs sémantiques et leurs utilisations ne peuvent pas être modifiées. Cette technique est utilisée pour affiner l'utilisation des métaconcepts dont la généricité n'est pas adéquate dans un contexte donné et dont la sémantique est source d'ambigüité (point de variation sémantique). Elle est normalisée par l'OMG au travers de la notion de profil et plus particulièrement de Profil UML (voir chapitre 18 de [37]).

Hypothèse 1.8 **L'extension légère par profil** *est supposée suffisante pour satisfaire les besoins d'extensions d'*UML *dans cette étude.*

Dans cette étude, la problématique de modélisation des plates-formes d'exécution doit donc être traitée par l'utilisation de profil UML.

Processus de développement dirigé par les modèles

L'intégration des plates-formes d'exécution dans un processus de développement vise à générer l'application implantée sur cette plate-forme, c'est-à-dire l'application exécutable sur cette plate-forme. Ce processus s'appuie sur un ensemble de modèles et de métamodèles ainsi que sur une succession de transformations de modèles. L'une des promesses de l'IDM est ainsi de fournir un cadre technique pour intégrer progressivement les concepts métiers des plates-formes et ainsi pouvoir générer l'application exécutable en bout de chaîne. L'approche MDA définit justement un cadre pour ce type de processus.

L'approche MDA

L'initiative MDA, vise à mettre en œuvre un modèle de fonctionnalités (également appelé modèle métier) sur différentes solutions technologiques, c'est-à-dire sur des plates-formes d'exécution différentes. A partir de descriptions indépendantes des plates-formes (Platform Independent Model PIM), elle vise à générer des applicatifs dédiés (Platform Specific Model PSM) à une plate-forme technologique donnée, sous entendu à une plate-forme dont la technologie permet d'exécuter le système.

L'indépendance n'est pas l'ignorance

Dans cette approche, l'indépendance ne signifie pas l'ignorance des plates-formes cibles mais au contraire la connaissance, c'est-à-dire l'ouverture sur toutes les plates-formes d'exécution susceptibles d'être ciblées. En pratique, cette ouverture se concrétisée par une montée en abstraction. Un PIM à un niveau d'abstraction est raffiné en plusieurs PSM décrits à des niveaux d'abstraction sous-jacents. Par exemple, la concurrence peut être décrite à haut niveau d'abstraction par des concepts de processus [39] ou d'objets temps réel [40] puis raffinée en ensemble de tâches sur un exécutif donné. A un haut niveau l'application est indépendante de l'exécutif mais spécifique au domaine du multitâche.

En somme, un processus de développement (MDA) est en fait une succession de transformations, de portages, d'une plate-forme à une autre. La description d'un tel développement est illustré en figure 1.6 page suivante . La description d'une application (PIM) est transformée en une implantation (PSM) sur une ou plusieurs plates-formes données. Les transformations permettant de passer d'un PIM à un PSM sont alors guidées pas les modèles des plates-formes visées. Ces modèles sont décrits soit implicitement dans la transformation elle même ou soit explicitement dans des modèles à part entière [41].

Hypothèse 1.9 *Cette étude s'appuie sur l'approche de développement préconisée par la spécification* **MDA**.

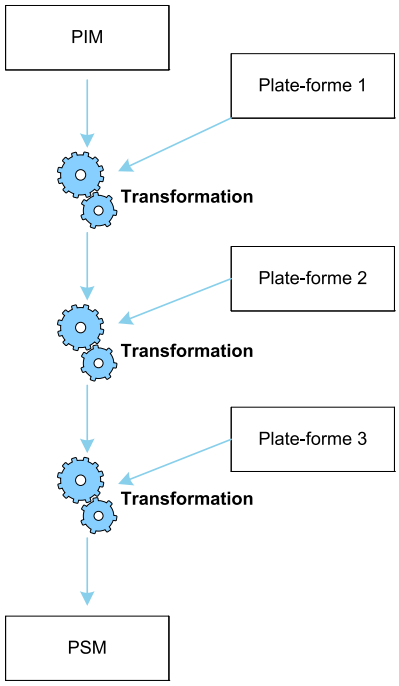

FIG. *1.6 – Synoptique de l'ingénierie générative mise en œuvre dans cette étude*

CONCLUSION

Dans ce chapitre, le champ d'investigation de cette étude a été restreint. Ces hypothèses de travail et cette problématique sont résumées dans la figure 1.7 page suivante . Dans cette figure, les branches de l'arbre détaillent les différentes possibilités auxquelles cette étude est confrontée. Une flèche indique un choix effectué et un rond une alternative non prise en compte. Cette étude cible donc le développement des applications logicielles temps réel embarquées monoprocesseurs s'exécutant sur des exécutifs temps réel multitâches. Ces exécutifs sont les plates-formes d'exécution à modéliser et à intégrer dans l'ingénierie dirigée par les modèles de cette étude. Cette étude doit donc apporter des réponses aux trois questions suivantes :

1. ***Qu'est ce qu'un modèle de plates-formes logicielles d'exécution multitâche ?***
2. ***Qu'est ce qu'un métamodèle de plates-formes logicielles d'exécution multitâche ?***
3. ***Comment intégrer des modèles et des métamodèles de plates-formes dans une ingénierie générative dirigée par les modèles de type* MDA *?***

Le chapitre suivant, chapitre 2 page 23, identifie les besoins de cette problématique et positionne les contributions de cette étude vis à vis des travaux existants.

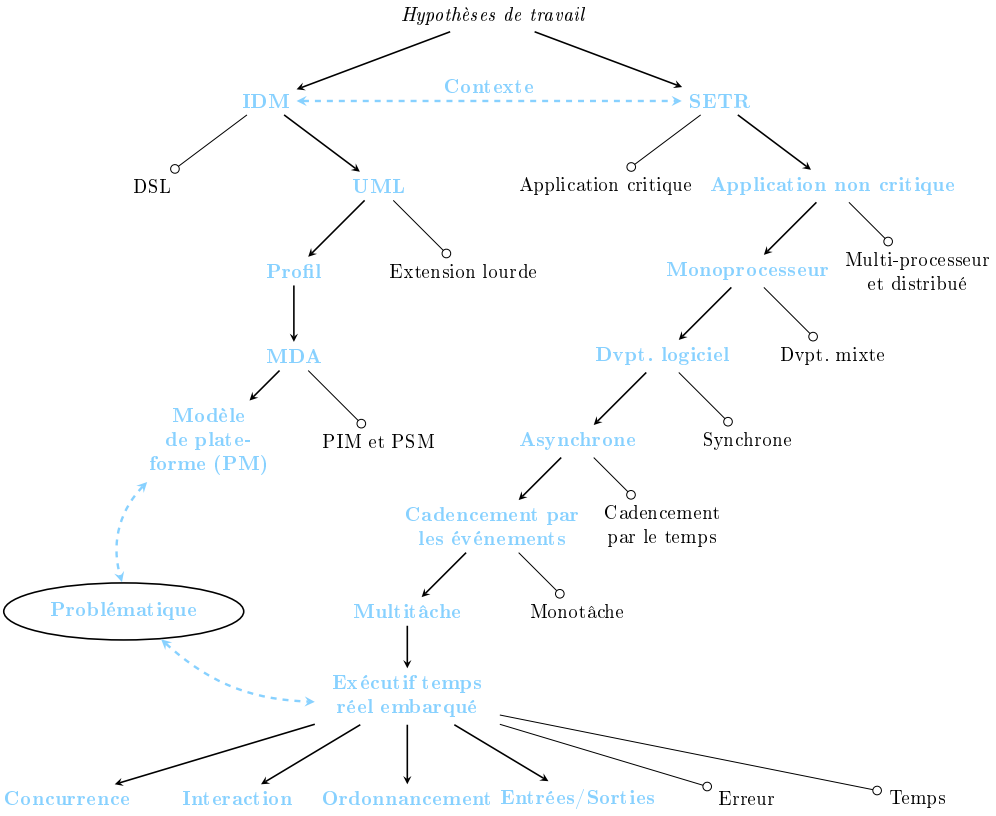

FIG. *1.7 – Synthèse des hypothèses de travail de cette étude*

État de l'art 2

Sommaire

Ce chapitre vise à positionner les contributions de cette étude vis à vis des travaux existants. Plus particulièrement, il vise trois objectifs : a) définir le concept de plate-forme d'exécution au sens de la littérature, b) définir les besoins de modélisation de ces plates-formes d'exécution, et c) positionner cette thèse vis à vis des travaux existants.

Pour cela, ce chapitre est divisé en deux sections. La première définit ce qu'est une plate-forme d'exécution au sens de l'IDM. Cette définition permet de déduire les besoins de modélisation des plates-formes dans une ingénierie générative dirigée par les modèles. Ces besoins sont utilisés comme critères de comparaison dans la seconde et la troisième sections. La seconde section étudie les approches modélisant les plates-formes logicielles d'exécution multitâche. Elle les décrit succinctement puis les confronte aux critères de comparaison. Enfin, elle positionne les contributions de cette thèse.

2.1 Définition des critères de comparaison

La caractérisation des critères passe par a) la définition du concept de plate-forme et b) la caractérisation des besoins.

2.1.1 Définition du concept de plate-forme dans l'IDM

Historiquement, le concept de plate-forme a été utilisé pour identifier la structure matérielle permettant l'exécution du logiciel. L'apparition de couches logicielles a étendu l'usage de ce concept pour décrire de la même façon des supports logiciels. Ainsi, les machines virtuelles, les intergiciels et les exécutifs sont des exemples de plates-formes d'exécution logicielles.

Une application n'est exécutable que dans le contexte d'une plate-forme d'exécution

L'exécution consiste en la réalisation d'une application (d'un système) [42]. La plate-forme traduit, calcule, les traitements spécifiés dans l'application en un ensemble de comportements physiques. Une plate-forme d'exécution propose ainsi un ensemble de concepts, de méthodes et de règles d'utilisation spécifiant quand et comment sont réalisés les traitements de l'application [13]. Une application n'est alors dite exécutable que dans le contexte d'une plate-forme d'exécution.

*La plate-forme dans l'*IDM *n'est pas forcément dédiée à l'exécution, c'est avant tout un serveur*

Dans l'IDM, le concept de plate-forme ne se limite pas aux entités d'exécution. Ainsi, Atkinson et Kühne [43] définissent les plates-formes comme des infrastructures, des systèmes, permettant la satisfaction d'un ensemble de besoins émis par d'autres systèmes appelés ***applications***. L'application est considérée comme un client et la plate-forme comme un serveur. L'exécution est un des services de ce serveur mais ce n'est pas le seul. L'analyse, la représentation et le stockage sont d'autres exemples de ces services. Ainsi, un outil d'analyse d'ordonnancement, un logiciel d'édition et une base de données sont considérés comme des plates-formes au sens de l'IDM.

La plate-forme est une abstraction

Sangiovanni-Vincentelli et Martin [8] soulignent qu'une plate-forme offre des services dont l'implantation n'est pas visible pour le client. Une application utilise les services de la plate-forme sans se soucier de l'implantation de ces services. La plate-forme est alors considérée comme une couche d'abstraction dans le processus de développement. En analogie aux solutions programmées, cette couche facilite un certain nombre de raffinements vers une sous-couche d'abstraction, c'est-à-dire une plate-forme sous-jacente. Selic [44] souligne, à ce titre, que bien que l'abstraction soit une caractéristique importante du concept de plate-forme, ce n'est pas une caractéristique intrinsèque. Une plate-forme est avant tout caractérisée par sa capacité à satisfaire un besoin. L'abstraction est dans ce cas une manière pratique de représenter les possibilités qu'une plate-forme offre pour satisfaire ce besoin.

La plate-forme est un rôle

En pratique, la distinction entre plate-forme et application dépend du point de vue où l'on se place. Toute plate-forme est une application pour une plate-forme de plus bas niveau d'abstraction [42]. Ainsi, un fournisseur de plate-forme matérielle considère le système d'exploitation comme une application puisque les besoins d'exécution de ce dernier sont satis-

faits par sa plate-forme (matérielle). En revanche un développeur logiciel perçoit le système d'exploitation comme un serveur d'exécution pour son application. La caractérisation d'un système comme étant une plate-forme n'est donc pas liée à la nature du système, mais bien plus à son rôle joué dans un contexte donné.

La plate-forme est abstraite ou concrète

Orthogonalement à cette notion de rôle, Almeida [45] distingue deux types de plates-formes : les plates-formes abstraites et les plates-formes concrètes. Cette distinction fait référence au concept d'éléments abstraits des langages objets. Dans son approche, une plate-forme abstraite est une plate-forme fictive, idéale face aux besoins à un instant donné du développement de l'application, c'est-à-dire une spécification d'un ensemble de technologies qui ne sont pas forcément implantée réellement. Dans un cycle de développement dirigé par les modèles, l'utilisation de plate-forme abstraite est récurrente. Par exemple dans [46], pour cibler différents exécutifs temps réel depuis une description multitâche haut niveau, un métamodèle intermédiaire nommée Concept of Real Time Application CORTA est utilisé. Il n'existe pas de concrétisation de cette plate-forme permettant directement d'exécuter les applications multitâches spécifiées (par exemple une machine virtuelle). CORTA est donc une plate-forme abstraite au sens d'Almeida. En revanche, les exécutifs visés depuis CORTA sont des plates-formes concrètes puisqu'ils permettent d'exécuter les applications multitâches spécifiées. Les implantations programmées de ces plates-formes permettent d'obtenir un comportement physique des applications spécifiées.

La plate-forme est implicite et/ou explicite

Indépendamment qu'elle soit abstraite ou concrète, il est intéressant de noter qu'une plate-forme peut être décrite dans l'IDM en tant que modèle (conforme à un métamodèle) et/ou comme un métamodèle (conforme à un métamétamodèle). La plate-forme est alors dite, respectivement, explicite (modèle) et/ou implicite (métamodèle). Par exemple, en UML, la sémantique de traitement des événements est basée sur l'hypothèse dite de ***run to completion*** selon laquelle un événement peut être dépilé et traité à condition que le traitement de l'événement précédent soit terminé. UML impose alors un ensemble de règles de calculs implicites permettant d'exécuter l'application (la réception des événements par exemple). Ces règles spécifient l'exécution du système. Elles sont implicites puisque le modeleur décrit son application en connaissant et en respectant implicitement ces règles. Il structure son modèle en fonction de ces règles. Il ne prévoit, par exemple, aucun mécanisme de synchronisation entre les traitements pour s'assurer qu'ils sont effectués dans l'ordre d'arriver des événements. Implicitement, UML satisfait ce besoin. Si ce modèle spécifique à la plate-forme d'exécution UML doit être utilisé pour générer du code exécutable, la sémantique de l'exécution ***run to completion*** peut être explicitée soit dans le générateur lui même ou dans un modèle paramétrant le générateur.

Définition du concept de plate-forme

En somme, il existe dans la littérature plusieurs familles de plates-formes et plusieurs façons de représenter une même famille de plates-formes. La définition de l'OMG proposée dans la norme Model Driven Architecture (Architecture Dirigée par les Modèles, MDA) [28] synthé-

tise ces différentes définitions. Puisque dans le MDA la notion de plate-forme n'est pas limitée à l'exécution, elle est réifiée dans la proposition 2.1.

Proposition 2.1 *Une plate-forme est un ensemble de sous-systèmes et de technologies qui fournissent un ensemble cohérent de fonctionnalités au travers d'interfaces et la spécification de patrons d'utilisation que chaque sous-système qui dépend de la plate-forme peut utiliser sans être concerné par les détails et sur la façon dont les fonctionnalités sont implantées [28].*

Une plate-forme est avant tout un rôle joué par un système à une étape du cycle de développement. Ce système peut être abstrait (fictive) où concret (opérationnel pour réaliser un comportement physique). Il est implicite lorsqu'il est décrit et utilisé comme un métamodèle et explicite lorsqu'il est décrit et utilisé au niveau modèle.

2.1.2 Caractérisation des besoins de modélisation

*L'*API *caractérise la plate-forme*

Les concepts, les méthodes et les règles d'utilisation fournis par la plate-forme sont généralement accessibles à l'application au travers d'une interface de programmation, appelée aussi l'interface de programmation de l'application ou API (Application Programming Interface). Outre la signature des concepts et des méthodes (sans se soucier de leurs implantations), l'API définit aussi le sens et le comportement de chacun de ces concepts et de ces méthodes (souvent de manière textuelle dans un manuel d'utilisation). Les concepts sont alors des mécanismes et les méthodes des services. Les mécanismes sont caractérisés par un ensemble de propriétés et les services par un ensemble de paramètres. Dans le contexte d'une ingénierie capable de générer des applications exécutables, les modèles de plates-formes sont donc constitués, tout au moins, d'une description des interfaces de programmation.

Un modèle de la plate-forme est un modèle de son API

Atkinson et Kühne [43] et Marvie *et al.* [42] ont identifié les besoins de modélisation de ces API. Dans ces approches, il existe une volonté de caractériser la signature structurelle et le comportement observable d'un point de vue utilisateur. Ils aboutissent donc à une caractérisation en quatre points : 1) la syntaxe concrète, 2) la taxonomie des mécanismes et des services, 3) le cycle de vie des mécanismes et des services ainsi que 4) les contraintes d'utilisation des mécanismes et des services inhérentes à la plate-forme.

► **La syntaxe concrète –** La syntaxe concrète constitue la représentation du concept utilisée par l'utilisateur pour spécifier son application. Cette syntaxe peut être textuelle ou graphique. Un exemple typique est la génération de code. En connaissant la suite des lettres associée à un concept, un générateur peut produire l'appel à un service de la plate-forme. La caractérisation de cette syntaxe concrète permet, par exemple, de paramétrer des générateurs de code génériques pour produire des applications spécifiques à une ou plusieurs plates-formes. La génération est alors dirigée par la syntaxe concrète décrite dans les modèles de plates-formes.

► **La taxonomie –** La taxonomie est la classification des données. Dans le contexte des plates-formes d'exécution, la taxonomie est la plus part du temps fournie à l'utilisateur de manière informelle au travers des noms des concepts et des notices d'utilisation de la plate-forme. Elle se concrétise alors par une classification selon les rôles joués par les mécanismes et les services, c'est-à-dire par leurs effets sur l'application. Dans le chapitre 1 page 7, cette classification a été décrite en quatre familles. Ainsi dans cette étude, la taxonomie doit permettre de décrire a) la concurrence, b) les intéractions, c) l'ordonnancement et d) les entrées/sorties.

► **Le cycle de vie –** Le cycle de vie des mécanismes et des services de la plate-forme représente les comportements des mécanismes et des services observables depuis l'interface de programmation.

► **Les contraintes d'utilisation –** Un dernier artéfact fourni par l'interface de programmation est l'ensemble des contraintes d'utilisation de la plate-forme. Le non respect de ces règles peut entraîner une mise en œuvre erronée et non fonctionnelle de l'application. Par exemple, la fonctionnalité d'une application appelant des services bloquants dans une routine d'interruption peut ne pas être garantie par certaines plates-formes d'exécution.

Ces quatre points constituent les critères de comparaison de cette étude.

2.2 État de l'art sur la modélisation des plates-formes logicielles d'exécution

Cette section s'intéresse à la modélisation des plates-formes logicielles d'exécution. Tout d'abord, elle introduit les principales approches existantes de modélisation. Ensuite, elle synthétise les apports et les lacunes de ces approches pour positionner les contributions de cette thèse.

2.2.1 Description des études existantes

Des études se sont intéressées à fournir des concepts destinés à la modélisation des plates-formes. Ce sont soit des langages spécifiques, c'est-à-dire des langages spécifiques au domaine du temps réel multitâche (Domain Specific Language, DSL), soit des extensions au métamodèle UML, c'est-à-dire des profils UML. Bien que cette étude vise l'extension du métamodèle UML, il est intéressant d'étudier de la même façon les approches DSL pour identifier les apports à réutiliser ou les lacunes à combler.

AADL

Description de l'approche AADL

Le langage Architecture Analysis and Design Language (AADL) [47],

normalisé par le consortium SAE, est un DSL dédié à la description d'architectures. Il permet la modélisation des plates-formes d'exécution logicielles grâce à trois concepts : Process, Thread et ThreadGroup. Le premier modélise un espace mémoire fournit par la plate-forme, le second un fil d'exécution et le troisième un groupement de fils d'exécution. Les communications entre ces espaces mémoires et ces fils d'exécution sont modélisés par des ports (Port) et des connecteurs (Connection). Les cycles de vie de ces concepts sont détaillés dans la norme elle-même.

Avantages de l'approche AADL

Le principal avantage de ce DSL est d'imposer une plate-forme implicite, c'est-à-dire une plate-forme décrite au niveau métamodèle. Les applications AADL sont alors décrites sur cette plate-forme implicite. Ce DSL cadre donc la description des applications en imposant une sémantique d'exécution particulière.

Limitations de l'approche AADL

Néanmoins, la proposition d'une nouvelle plate-forme ne permet pas la description d'autres plates-formes. Elle impose la description d'une seule plate-forme, celle spécifiée par la norme. Ce n'est donc pas un métamodèle de plate-forme.

TUT-Profile

Description de l'approche TUT-Profile

TUT-Profile [10] est un profil UML (dans sa version 2.0) pour la conception des systèmes embarqués temps réel. Ce profil a été étendu dans [48] pour la modélisation des plates-formes logicielles d'exécution. Ainsi, il définit quatre concepts SwPlatform, SwPlatformLibrary, SwPlatformComponent et SwPlatformProcess pour décrire les plates-formes logicielles. Le premier décrit la plate-forme dans sa globalité. Le second permet de décrire un package regroupant l'ensemble des constituants de cette plate-forme. Le troisième représente chacun des constituants de cette plate-forme sous la forme de classes UML. Enfin le dernier, SwPlatformProcess, représente un élément (typé) dont le type est un composant de la plate-forme (SwPlatformComponent). Pour cela les auteurs spécialisent la notion de propriété UML et préconisent l'utilisation des diagrammes structurés où les classes sont composées (une classe est composée de propriétés). Dans TUT-Profile, il n'y pas de concepts explicites dédiés à la spécification des cycles de vie des éléments de la plate-forme. Il s'appuie pour cela sur les diagrammes d'activités et les diagrammes d'états transitions. De même, il utilise des annotations UML pour représenter les contraintes d'utilisation.

Avantage et limitations de l'approche TUT-Profile

L'avantage de l'approche TUT-Profile est l'extension du métamodèle UML dans sa version 2.0 et la réification du concept de package par le stéréotype SwPlatform. Cela permet d'identifier explicitement les packages jouant le rôle de plate-forme à une étape de développement. Néanmoins, les mécanismes et les services de la plate-forme restent totalement implicites puisque il n'y a aucun concept dédié à l'identification des mécanismes et des services métiers (la concurrence, l'interaction et l'ordonnancement par exemple).

SPT

Description de l'approche SPT

Le profil UML SPT [9], normalisé par l'OMG, est un profil UML (version 1.4) dédié à la modélisation de l'ordonnancement, de la performance et du temps. Il propose des artéfacts pour la modélisation des plates-formes d'exécution par le biais du package Generic Resource Modeling (GRM). GRM propose principalement deux notions : les Resource et les Service. Il n'y a pas d'autres artéfacts dédiés à la description des types de ressources et des types de services. Par contre, il est possible de décrire les rôles que les ressources jouent une fois utilisées dans l'application. Les rôles définis sont ceux de a) Processor Resource, b) Communication Resource, c) Device, d) Active Resource, e) Passive Resource, f) Protected Resource et g) UnProtected Resource. Ces concepts permettent de décrire des ressources utilisées pour des exécutions concurrentes (Processor, Active et Passive), pour des communications (Communication), pour des synchronisations (Protected et UnProtected) ainsi que pour la gestion des entrées/-sorties (Device). Tout comme l'approche TUT-Profil, SPT s'appuie sur UML pour la description des cycles de vie et des contraintes d'utilisation.

Avantages et limitations de l'approche SPT

La notion de ressource est intéressante pour décrire les mécanismes des plates-formes puisqu'elle conceptualise des entités aux capacités d'exécution et aux capacités d'instanciation limitées, ce qui correspond au rôle de serveur d'une plate-forme (voir section 2.1.1 page 25). Par contre, l'identification des rôles joués par les ressources plutôt que les ressources elles-mêmes est un handicape. Le modèle résultant n'est pas un modèle de plate-forme mais un modèle de l'application implantée sur cette plate-forme. De plus, la encore la plate-forme reste en grande partie implicite. Des mécanismes tels que les boites aux lettres ou les sémaphores ne sont pas identifiés explicitement.

Metropolis

Plusieurs travaux se sont inspirés de l'approche Platform Based Design [8] dédiée à la définition d'une approche de développement dirigée par les modèles de plates-formes. Le plus significatif est la méthodologie (et l'outil associé) Metropolis [49]. Dans cette approche, un métamodèle nommé Metropolis Meta Model [50, 51] est défini. Il permet de décrire, entre autre, des modèles de plates-formes sous la forme d'un pseudo code Java.

Description de l'approche Metropolis

Le métamodèle Metropolis définit les concepts de Process, de Media, d'Interface, de Quantity Manager et de Netlist pour décrire la structure et le comportement des plates-formes. Un Process permet de décrire des entités dont l'exécution est concurrente. Les Media sont utilisés pour les interactions, c'est-à-dire les communications de données et les synchronisations entre les Process. Les Quantity manager, spécifiés par l'utilisateur, contrôlent l'accès aux Media partagés. Les Process et les Media possèdent des paramètres et des méthodes. Ils sont connectés par l'intermédiaire de Port. Les connexions entre ces Port sont stockées dans des Netlist. La

Netlist regroupe les instances des Process et des Media utilisées dans l'application. Les langages formels Linear Temporal Logic (LTL) et Logic of Constraints (LOC) sont utilisés, respectivement, pour la vérification de propriétés dans le temps et pour la description des performances attendues du système. LTL et LOC peuvent donc être tous les deux utilisés pour la description et pour la spécification des contraintes d'utilisation de la plate-forme.

Avantages de l'approche Metropolis

L'approche Metropolis a pour avantage, d'une part, de décrire n'importe quelle plate-forme qu'elles soient abstraites ou concrètes et, d'autre part, d'être opérationnelle [52]. L'outil Metropolis permet en effet de vérifier des contraintes fonctionnelles ou architecturales, de simuler et de synthétiser des fonctionnalités et enfin d'explorer les différentes configurations possibles d'allocation entre un modèle fonctionnel et un modèle de plate-forme. L'utilisation d'un contrôleur de quantité (QuantityManager) et d'un ensemble de contraintes est un apport majeur de cet outil pour l'intégration de modèle de plate-forme dans le développement de l'application. Toutes ces fonctionnalités sont liées entre elles par l'intermédiaire d'un langage pivot unique : le Metropolis MetaModel.

Limitations de l'approche Metropolis

Néanmoins, d'après les auteurs de l'outil eux mêmes (voir paragraphe I.C de [52]), la généricité du métamodèle Metropolis ne facilite pas la description des plates-formes. Les concepts de Process et de Media sont utilisés aussi bien pour la description de l'application et pour la description de la plate-forme. Ce sont donc des concepts à un haut niveau d'abstraction qui n'identifient pas les différentes familles de mécanismes et de services, ce qui limite l'intégration des modèles résultants dans des ingénieries génératives paramétrables et réutilisables. Enfin, même si le métamodèle Metropolis permet de décrire des plates-formes, il n'existe pas dans l'approche la notion de plate-forme implicite, c'est-à-dire l'utilisation d'un métamodèle pour représenter la plate-forme, ce qui limite l'utilisation de cette approche dans une ingénierie générative dirigée par les modèles.

Platform Modeling Langage

Dans sa thèse, Szemethy [53] définit un métamodèle, nommé Platform Modeling Langage ou PML, dédié à la description de modèles de plates-formes. Les plates-formes ne sont pas forcément dédiées à l'exécution d'application. Il décrit par exemple des plates-formes dédiées à la description et l'analyse des applications tel que l'outil UPPAAL[1] par exemple. Ces plates-formes sont soit implicites, soit explicites. Dans le cas des plates-formes implicites, une transformation de modèle[2] spécifique génère un modèle de la plate-forme conforme au métamodèle PML.

Description de l'approche PML

Dans le métamodèle PML, un modèle de plate-forme (PlatformModel), contient a) des composants (ComponentModels) et b) des règles de transformation entre les composants des plates-formes sources et cibles (Pattern, Match). Ce métamodèle se veut donc minimaliste si bien qu'il

[1] http ://www.uppaal.com/

[2] Transformation écrite dans le langage GReaT http ://www.isis.vanderbilt.edu/tools/GReAT.

étend le métamodèle UML (version 1.4) par héritage (extension dite lourde). Le métamodèle PML-UML résultant permet de modéliser la structures des mécanismes et la signature des services, leurs cycles de vie et leurs contraintes d'utilisation (utilisation des diagrammes d'activité, des diagrammes d'états-transitions, et des contraintes OCL). Dans cette approche, un modèle de plate-forme est typiquement représenté par un diagramme de classe UML. Le même métamodèle est donc utilisé pour décrire l'application, les plates-formes et les transformations entre ces plates-formes. Il est intégré dans l'atelier de modélisation GME [3].

Avantages de l'approche PML

L'avantage de l'approche PML est l'outillage des transformations associées au métamodèle PML. Cet outillage, intégré à l'outil GME, est capable de générer une partie des règles de transformation permettant de passer d'une plate-forme source vers une plate-forme cible. La génération n'est pas totale, 60% de la transformation est générique sur les chiffres publiés dans [53], mais elle permet d'assister l'intégration des plates-formes dans un cycle de développement. Ces plates-formes sont alors implicites ou explicites.

Limitations de l'approche PML

Même si l'idée de générer la transformation qui va effectivement manipuler les modèles de plates-formes est intéressante, la description de certaines règles de transformation dans les modèles de plates-formes fausse l'intégration des plates-formes dans une ingénierie générative. Le modèle de plate-forme est en effet un amalgame de mécanismes, de services et de règles de transformations. Enfin, l'extension dite lourde du métamodèle UML est un handicap car elle impose la description d'un outillage spécifique pour la représentation des modèles de plates-formes. Une extension par profil aurait pu tout aussi bien être choisie.

2.2.2 Synthèse et positionnement

Le tableau 2.2.2 page ci-contre synthétise et compare les différentes approches de modélisation des plates-formes logicielles d'exécution multitâches vis à vis des besoins de modélisation de cette étude. Dans ce tableau une croix (✗) signifie qu'il n'est pas proposé d'artéfacts spécifiques pour la modélisation. Une coche (✓) signifie que des artéfacts sont proposés. La dernière colonne distingue les approches permettant de décrire des modèles de plates-formes (distingué par une coche (✓)) et non des applications implantées sur des plates-formes implicites[4] (distingué par une croix (✓)).

[3] http ://www.isis.vanderbilt.edu/projects/gme/

[4] Plates-formes décrites au niveau méta.

TAB. 2.1 – *Comparaison des études existantes sur la modélisation des plates-formes logicielles d'exécution multitâche*

	Approches	Syntaxe concrète	Taxonomie: Concurrence	Taxonomie: Interaction	Taxonomie: Ordonnancement	Taxonomie: Entrée/Sortie	Taxonomie: Cycle de vie	Taxonomie: Contrainte d'utilisation	Taxonomie: Métamodèle de plate-forme
DSL	**AADL**	✗	✓	✗[1]	✗	✗	✗	✗	✗[2]
DSL	**PML**[5]	✗	✗[4]	✗[4]	✗[4]	✗[4]	✓	✗	✓[3]
DSL	**Metropolis**	✗	✓	✓	✓	✗	✓	✓	✓[3]
UML	**TUT-Profil**	✗	✗[4]	✗[4]	✗[4]	✗[4]	✓[6]	✓[7]	✓[2]
UML	**SPT**	✗	✓	✗	✓	✗	✓[6]	✓[7]	✗[2]

[1] Concepts génériques de ports et de connecteurs
[2] Le langage proposé est une nouvelle plate-forme.
[3] Le langage proposé permet de décrire des plates-formes existantes ou fictives.
[4] Concepts génériques de composants
[5] Extension lourde du métamodèle UML
[6] UML fournit les moyens pour décrire les cycles de vie des concepts
[7] Utilisation du langage de contrainte OCL

Toutes les approches existantes : AADL, PML, METROPOLIS, TUT-PROFIL et SPT proposent des artéfacts incomplets ou top abstraits pour modéliser les plates-formes logicielles d'exécution multitâche. En fait, ces approches modélisent les plates-formes à un très haut niveau d'abstraction. Ce niveau d'abstraction élevé permet de décrire aussi bien les plates-formes logicielles que les plates-formes matérielles. Cependant, il ne facilite pas la compréhension et l'informatisation des modèles résultants, c'est-à-dire qu'il ne permet pas la mise en place d'outils automatiques d'analyse ou de génération. Ces analyses et ces générations sont en fait possibles, mais elles nécessitent des sources d'information supplémentaires, sources qui caractérisent de la même façon la plate-forme et qui devrait donc apparaître logiquement dans un modèle de plate-forme. Par exemple, le profil UML SPT et le langage spécifique METROPOLIS modélisent les mécanismes, respectivement, par des ressources et par des composants. Les services sont quant à eux modélisés comme des opérations UML et des opérations Java. Il est ainsi impossible pour un outil informatique de faire la différence entre deux ressources ou entre deux services sans informations supplémentaires. Même si l'analyse syntaxique et sémantique du nom d'un mécanisme pourrait permettre cette distinction, elle n'est pas garantie ni exempte d'une intervention de l'utilisateur.

Positionnement

Par rapport aux travaux existants, les besoins de modélisation de la taxonomie des mécanismes et des services doivent être satisfaits. Cette thèse doit proposer des concepts métiers concrets tels que la notion de tâche, de sémaphore, de boîte aux lettres. Elle ne doit pas définir une nouvelle API comme le font les approches AADL ou SPT, mais elle doit proposer des artéfacts pour décrire ces API. Dans un premier temps, cette étude ne traite pas le problème de la syntaxe concrète ni celui des cycles de vie puisque ces besoins nécessitent, auparavant, la satisfaction des besoins taxinomiques. De même, elle ne s'intéresse pas à la modélisation des contraintes d'utilisation. Des langages dédiés tels OCL, LTL et LOC peuvent être utilisés conjointement. Il faudra alors outiller les modeleurs UML pour évaluer ces règles. Cette étude n'a pas pour objectif d'interfacer de tels outils dans un premier temps.

Conclusion

Dans ce chapitre, le concept de plate-forme et de modèle de plate-forme ont été définis au sens de l'IDM. Le modèle d'une plate-forme est le modèle de son API. Ensuite des besoins ont été définis et les études existantes ont été confrontées. Leurs synthèses et leurs comparaisons positionnent le travail de cette thèse. La figure 2.1 positionne cette thèse vis à vis des besoins.

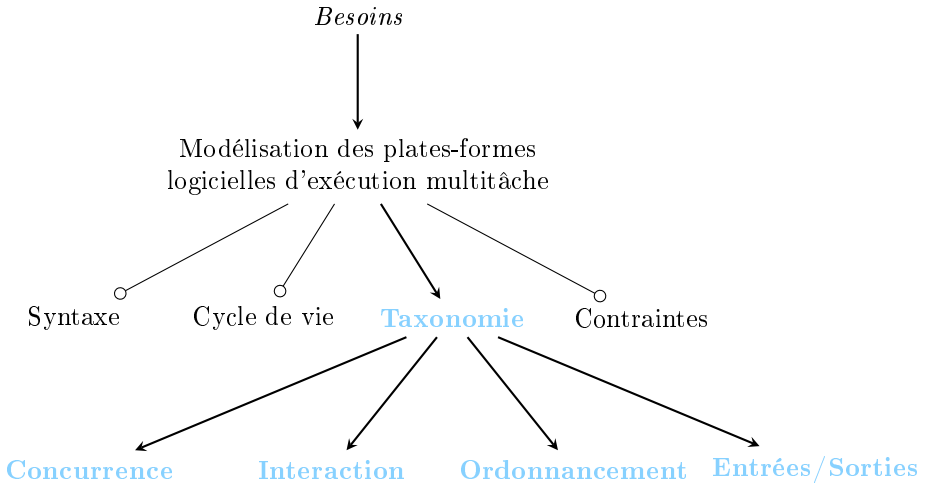

Fig. 2.1 – *Synthèse des besoins pris en compte dans cette étude*

La démarche de ce travail doit aboutir à la modélisation explicite des plates-formes logicielles d'exécution multitâche et à l'intégration des modèles résultants dans un développement dirigé par les modèles. Elle doit permettre de décrire l'ontologie des agences multitâches visées (concurrence, interaction, ordonnancement et entrée/sortie).

Pour satisfaire ces besoins, il faut tout d'abord étudier de manière générique les briques élémentaires d'un métamodèle de plate-forme puis appliquer ces briques sur un métamodèle de plate-forme multitâche. Ce sont les sujets de la deuxième et de la troisième partie.

Deuxième partie

Contributions génériques à la modélisation des plates-formes logicielles d'exécution

Définition et implantation du motif Resource-Service

3

Sommaire

Ce chapitre vise à définir les briques élémentaires, c'est-à-dire les concepts primitifs et les liens entre ces concepts, constituant le cœur des métamodèles dédiés à la modélisation des plates-formes logicielles d'exécution dans l'IDM.

Pour cela, cette section commence par identifier les contours de modélisation des plates-formes. Ensuite, elle définit un motif de modélisation nommé Resource-Service, c'est-à-dire un ensemble de métaconcepts et de métaliens primitifs nécessaires à la description des plates-formes logicielles d'exécution. Dans une dernière section, l'implantation de ce motif dans un profil UML est discuté.

3.1 Définition des contours de modélisation

3.1.1 Illustration des modélisations à mettre en œuvre

Dans le chapitre précédent 2.1.2 page 27, un modèle de plate-forme est définit comme un modèle de son API. Tout comme le modèle applicatif, ce modèle doit être décrit en UML. Il peut être décrit comme un Model UML (plate-forme explicite) ou comme un Profil UML (plate-forme implicite) (voir sous-section 3.3.2 page 50). Afin de concrétiser ces notions de plates-formes explicites et implicites, la figure 3.1 page suivante représente, de manière simplifiée, une application robotisée. Dans cette application, un contrôleur met à jour la consigne de position par l'opération updateControl et calcul la consigne de trajectoire par l'opération compute. Pour effectuer leurs calculs, ces opérations utilisent la variable position. Dans cet exemple, un Robot ne possède qu'un contrôleur et qu'une position. Par conséquent, la position est partagée par les deux opérations updateControl et compute. Cette application est implantée sur un ensemble de plates-formes logicielles d'exécution : OSLibrary et RtPlatform.

Illustration des modèles de plates-formes

La première représente une implantation d'un exécutif temps réel qui fournit les concepts de partition mémoire (Process), de sémaphore (Semaphore) et de tâches (Thread). L'ordonnancement de ces tâches est effectué par un ordonnanceur (s) défini et instancié par la plate-forme elle-même. Dans cette plate-forme, il n'existe pas le concept de tâche périodique. Une couche logicielle nommée RtPlatform a été développée pour y remédier.

Le profil RtPlatform constitue la seconde plate-forme d'exécution de cette application. Elle fournit le concept de service temps réel (RtService). Ce service propose un contexte d'exécution périodique. Pour des raisons propres à la méthodologie de développement, cette plate-forme est décrite au niveau «méta» comme un profil UML. Les méthodes updateControl et compute sont décrites comme des services temps réel. Ces services sont alloués explicitement sur l'ordonnanceur s de la plate-forme explicite OSLibrary. Ils s'exécutent dans le contexte d'un Robot qui réifie le concept de processus de la plate-forme OSLibrary. Les accès simultanés à la position sont gérés par un sémaphore explicite. Le diagramme de séquence illustre la prise du sémaphore par la tâche updateControl. Dans ce diagramme, la ligne de vie de l'objet Controller est dupliquée pour faciliter la lisibilité du diagramme. En réalité, il n'existe qu'un seul contrôleur dans l'application finale.

Dans cet exemple, il apparait que les relations entre l'application et les plates-formes sont de deux types : les relations implicites et les relations explicites.

3.1.2 Études des relations entre applications et plates-formes

Relations implicites

L'application du stéréotype rtService sur l'opération compute spécifie l'exécution de cette opération dans le contexte d'un service temps réel

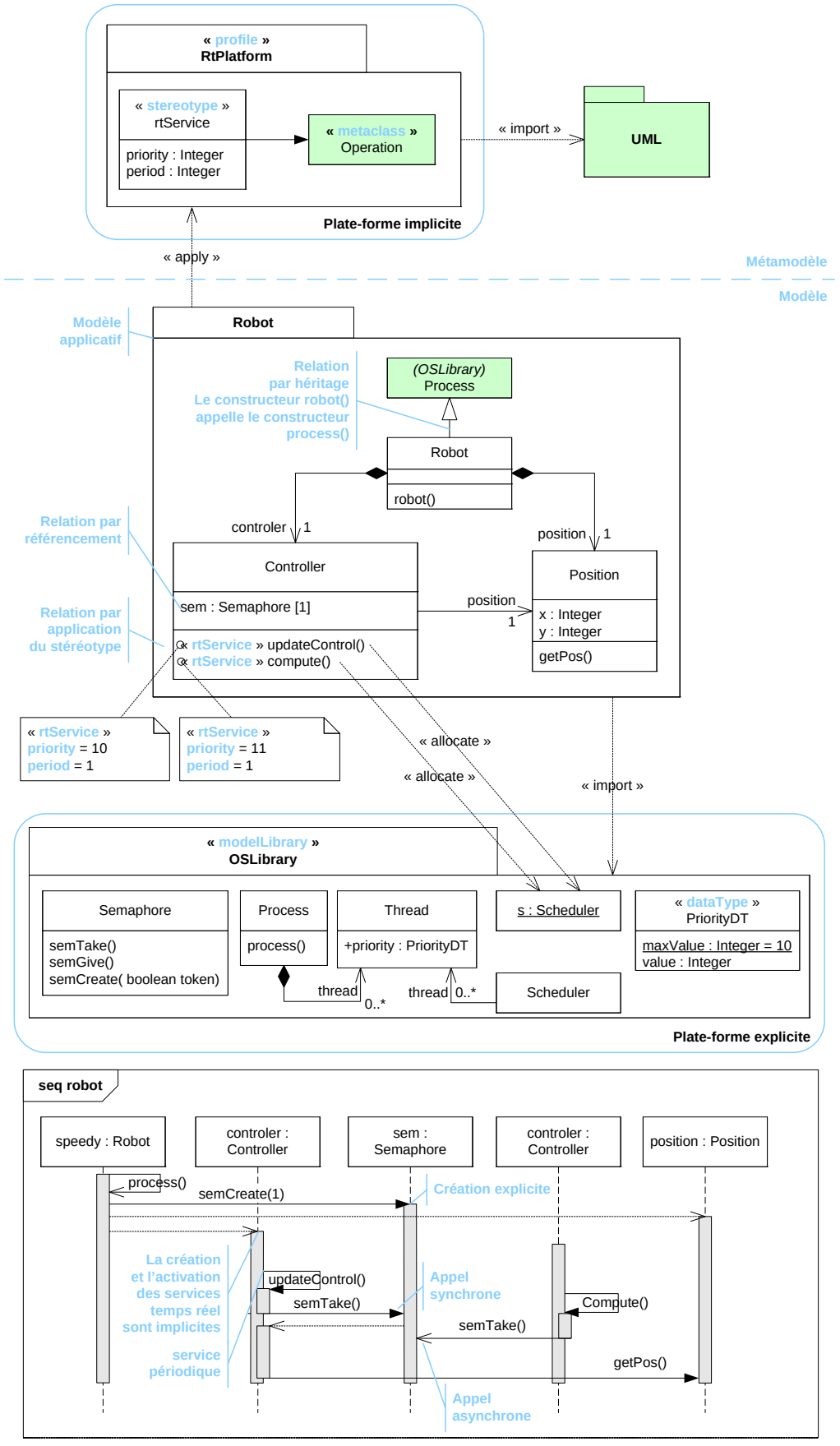

FIG. 3.1 – *Illustration des modèles de plates-formes dans* UML

de la plate-forme RtPlatform (Ce service est par défaut périodique). L'annotation d'un élément applicatif par un stéréotype spécialise donc le modèle applicatif pour une plate-forme donnée. L'application d'un stéréotype lie implicitement le contexte d'exécution du concept applicatif avec le contexte d'exécution du concept de la plate-forme. Toutefois, l'application d'un stéréotype n'est pas la seule relation implicite dans cet exemple. L'appel de l'opération semTake avec attente ou non, modélisé en UML dans le diagramme de séquence par un appel synchrone ou asynchrone, spécialise eux aussi l'exécution de l'application. L'ensemble des plates-formes implicites impliquées dans la description de l'application est alors l'ensemble {RtPlatform, UML}. L'application du stéréotype participe en fait à une relation implicite plus globale qui est la relation de conformité (χ). Cette relation implante implicitement l'exécution de l'application dans les contextes d'exécution d'un ensemble de plates-formes décrites au niveau méta.

Relations explicites

En ce qui concerne les relations explicites, celles-ci sont plus facilement détectables puisque explicites. Dans la figure 3.1 page précédente , trois types de relations différentes sont utilisées : l'héritage, le référencement et les dépendances. Premièrement, l'héritage spécialise par enrichissement [54] l'application en réifiant le concept de processus de la plate-forme OSLibrary. Le comportement de la classe Robot hérite du comportement de la classe Process (appel du constructeur parent spécifié dans le diagramme de séquence).

Deuxièmement, le référencement d'un sémaphore par le contrôleur spécialise l'application pour la plate-forme OSLibrary. Ce référencement se traduit par le typage dans une description structurelle (typage de la variable sem par la classe Semaphore) ou par un appel de service dans une description comportementale (appel du service semTake).

Troisièmement, des dépendances sont décrites explicitement entre les opérations de l'application et l'ordonnanceur de la plate-forme. Cette relation de dépendance est une allocation dans l'exemple. Des relations de Substitution , de Realisation ou enfin d'Usage sont aussi utilisables, chacune dans un contexte défini par le métamodèle.

Il existe donc de nombreuses manières de spécialiser une application pour une plate-forme donnée. Afin de définir les briques élémentaires de modélisation des plates-formes, il est nécessaire de définir les contours de modélisation des plates-formes.

3.1.3 Synthèse des contours de modélisation

Un modèle applicatif ne peut pas modifier un modèle de plate-forme

Par définition une application ne peut modifier l'implantation d'une plate-forme. Le modèle applicatif ne peut donc pas modifier le modèle de la plate-forme. Ce dernier est en «lecture seule» lorsqu'il est utilisé pour décrire une application. Ainsi, les relations sont toujours orientées de l'application (la source), vers la plate-forme (la cible). C'est nativement le cas dans toutes les relations de l'exemple précédent. Les relations ne sont donc pas à prendre en compte lors de la description des plates-formes. Les

relations appartiennent soit au modèle applicatif lui même (l'héritage par exemple) ou soit à un modèle orthogonal dédié (un modèle d'allocation par exemple).

Diversité des approches de modélisations

Par contre, la diversité des relations participe à la diversité des styles de modélisation. Indépendamment des approches d'implantation, il existe par exemple des approches de modélisation objet, des approches de modélisation fonctionnelle ou bien encore des approches de modélisation à base de composant. La figure 3.1 page 40 , représente la plate-forme par des classes, cependant, une modélisation par composants peut être envisagée. Ainsi, outre la description de modèle et de profil UML, le cœur d'un métamodèle de plate-forme doit être ouvert à toutes les approches de modélisation disponibles dans UML pour décrire des API. Le cœur d'un tel métamodèle est défini dans cette étude par un motif.

Dans les travaux de Gamma *et al.* [55], un motif se différencie d'un patron (*Design pattern*) par le fait que le patron englobe le problème de conception à solutionner et que le motif est une solution proposée à ce problème. Un patron est constitué du problème, du motif proposé et d'un ou plusieurs exemples d'implantation de ce motif. Dans cette étude, le problème est la modélisation des plates-formes, le motif est nommé Resource-Service et des exemples d'implantations sont illustrés dans la suite de cette thèse.

3.2 Définition du motif Resource-Service

La modélisation des API indépendamment des approches de modélisation, passe par l'identification de besoins primitifs de modélisation structurelle et comportementale, puis la satisfaction de ces besoins dans un profil UML.

Une pratique de conception d'un profil UML est une approche en deux étapes [56, 57]. Une première étape consiste à spécifier tous les concepts utiles pour modéliser le domaine d'intérêt, à savoir les concepts métiers et leurs relations (par exemple les concepts de tâche et de sémaphore). Le modèle résultant est alors un modèle de domaine, c'est-à-dire la spécification d'un métamodèle pour le domaine considéré. Ensuite, la seconde étape vise à implanter cette spécification sous la forme d'un DSL ou d'un profil UML. Les avantages de cette approche sont la séparation des préoccupations (concepts métiers versus concepts linguistiques) et la factorisation du modèle de domaine pour plusieurs implantations. La figure 3.2 page suivante illustre l'approche de conception suivie dans la suite de cette section. Dans un premier temps, les besoins sont définis et le modèle de domaine est construit. Ensuite, une implantation du modèle de domaine dans un profil UML est proposée.

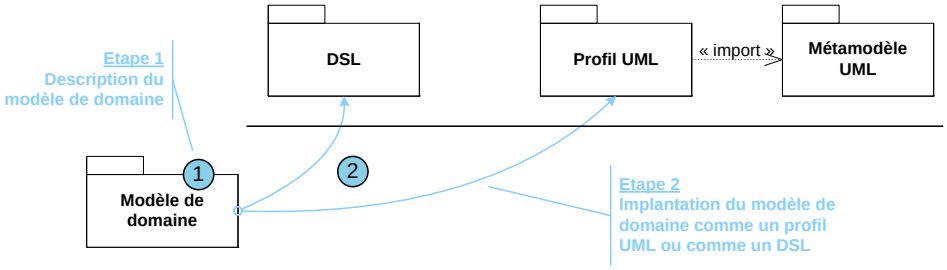

FIG. 3.2 – *Approche de conception des profils UML*

3.2.1 Modélisation structurelle des APIs

Une API logicielle fournit principalement à l'utilisateur un ensemble : a) de types, b) de types de donnée, c) d'instances et d) d'opérations.

► **Les types –** Comme le rappel Steel et Jézéquel [58], les types sont des ensembles de valeurs sur lesquelles des opérations peuvent être définies. Ils fournissent un contexte implicite pour les opérations et contraignent l'utilisateur pour la description de l'application. Outre leur nom, ils sont décrits par des propriétés. En pratique, ils représentent les mécanismes fournis par la plate-forme comme le sémaphore et la tâche (Thread) de l'exemple 3.1 page 40 .

► **Les types de donnée –** Les types de donnée sont des types particuliers dont les instances ne sont identifiées que par leurs valeurs. Les types de donnée spécifiques aux APIs des systèmes embarqués sont par exemple les entiers signés, non signés sur huit, seize ou trente deux bits. Dans l'exemple 3.1 page 40 , la priorité d'un Thread est un type de donnée particulier dont les valeurs sont comprises entre zéro et dix.

► **Les instances –** Les instances sont des éléments typés, prédéfinis et initialisés par la plate-forme elle-même. Elles sont paramétrables par l'utilisateur et, parfois, directement manipulables par l'utilisateur. Dans la figure 3.1 page 40 , l'ordonnanceur s est un exemple de ces instances initialisées par la plate-forme. L'utilisateur peut le paramétrer en lui affectant par exemple une nouvelle politique d'ordonnancement. Il peut aussi le manipuler en suspendant, par exemple, sa capacité de préemption sur les tâches du système.

► **Les opérations –** Les opérations sont des ensembles d'actions que la plate-forme fournie pour manipuler les types. En pratique, ce sont les services permettant de manipuler les mécanismes des plates-formes. Ces opérations sont appelées et paramétrées par l'utilisateur. La prise (semTake) et la vente (semGive) d'un sémaphore sont par exemple effectuées par des services de l'API dans la figure 3.1 page 40 .

3.2.2 Définition des briques structurelles

Le métamodèle UML fournit des artéfacts pour la modélisation des types, des types de donnée, des instances et des opérations respectivement par les concepts de Type, DataType, InstanceSpecification et Operation. Le concept UML Type est un élément abstrait, racine, du métamodèle qui est ensuite réifié pour les approches objets (concepts de Classifier et de Class) et les approches composants (StructuredClassifier et Component) par exemple. Cette réification est intéressante pour le métamodèle de plate-forme recherché car elle permet de modéliser des approches technologiques et méthodologiques hétérogènes.

Cependant, comme la montré l'état de l'art du chapitre 2, UML doit être spécialisé pour décrire des concepts dont les instances sont en nombre limité et dont les capacités d'exécution sont limitées. La caractérisation de ces capacités d'exécution s'illustre par la description des performances et des qualités de services fournies. Ces caractéristiques qualifient les performances de la plate-forme considérée. Les temps d'exécution au pire cas sont des exemples de ces caractéristiques. Certaines études [59, 60] proposent par exemple une quantification des pires temps d'exécution pour les services de différents exécutifs temps réel. Ces quantifications doivent donc pouvoir être modélisées et liées explicitement aux artéfacts de modélisation des APIs (les types, les types de donnée et les services par exemple). Certains concepts d'UML nécessitent donc d'être réifiés par des concepts proches des préoccupations métiers des plates-formes d'exécution, c'est-à-dire par les concepts de Ressource et de Service proposés dans [9] et dans [44].

► **Ressource –** La notion de ressource (Resource en anglais) représente les types de la plate-forme, c'est-à-dire les mécanismes fournis par la plate-forme qu'il faut savoir gérer puisque leurs capacités d'exécution et leurs instanciations sont limitées. Ces types sont alors caractérisés par des propriétés (ResourceProperty). Elles sont instanciées sous la forme d'instance de ressource (ResourceInstance).

► **Service –** Les traitements offerts par la plate-forme sont des services de ressources (Service), c'est-à-dire un ensemble d'opérations qu'un client (l'application) requière d'un serveur (la plate-forme) pour pouvoir s'exécuter. Ces services sont alors principalement composés de paramètres, c'est-à-dire d'éléments typés, renseignés lors des appels aux services.

Tous ces artéfacts de modélisation sont des éléments annotés, c'est-à-dire qu'ils peuvent être qualifiés par des propriétés décrivant la qualité de service garantie (le pire temps d'exécution par exemple). Ces propriétés reflètent alors les caractéristiques d'exécution et de performance offertes à l'utilisateur par la plate-forme.

La figure 3.3 page ci-contre illustre le modèle de domaine de ces concepts.

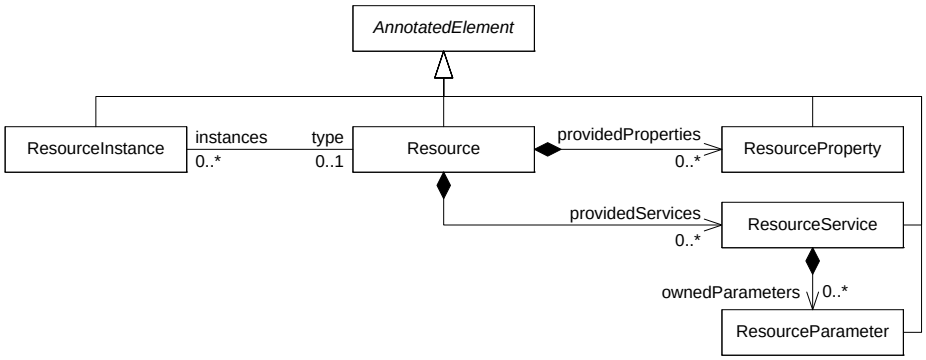

Fig. 3.3 – *Ébauche du modèle de domaine du patron* Resource-Service

3.2.3 Agencement des briques élémentaires de modélisation

Deux agencements sont possibles pour conceptualiser le domaine des plates-formes logicielles d'exécution : a) un agencement hiérarchique et b) un agencement par référencement.

Agencement hiérarchique

Traditionnellement en métamodèlisation, les concepts d'un domaine sont organisés sous la forme d'un arbre hiérarchique. Pour chaque concept du domaine, un noeud de l'arbre est créé. Ce noeud est un métatype. Il est lié aux autres concepts par des relations d'héritages, d'associations ou de dépendances. Un fragment du modèle de domaine résultant est présenté dans la figure 3.4 . Ce modèle est constitué de ressources. Ces ressources possèdent des propriétés et des services. Certaines ressources sont des tâches Task. Une tâche possède une priorité et des services d'activation et de suspension d'exécution. Certaines de ces tâches sont périodiques. Elles sont alors caractérisées par une période.

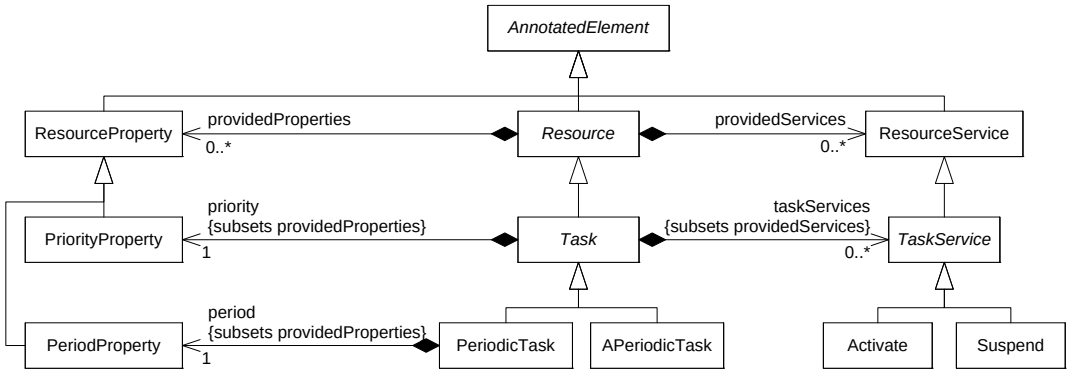

Fig. 3.4 – *Agencement hiérarchique du modèle de domaine*

Nombre de métatypes important et redondance d'information

Comme le montre la figure 3.4 , cet agencement impose, d'une part, un nombre important de métatypes et, d'autre part, l'utilisation d'héritage

multiple. L'utilisation d'un métatype pour chaque mécanisme, chaque propriété, chaque service et chaque paramètre composant le modèle de domaine des plates-formes est un frein cognitif à l'adoption et à l'utilisation d'un tel métamodèle. C'est d'autant plus un frein cognitif que, dans la majorité des cas, ces métatypes ne possèdent aucune métacaractéristique particulière. Par conséquent, il y a une redondance d'information entre le rôle attribué à l'extrémité de l'association et le métatype typant ce rôle. Par exemple, la notion de période est identifiée à la fois par le métatype PeriodProperty et par la propriété period de la composition (composition entre PeriodicTask et PeriodProperty). Il y a donc redondance d'information.

Des outils pourraient être envisagés pour gérer, filtrer et guider l'utilisation de tous ces métatypes. Néanmoins l'objectif d'un motif pour la modélisation des plates-formes d'exécution n'est pas la spécification d'une méthodologie ou d'un outillage générique ; mais la capitalisation d'un ensemble de concepts génériques nécessaires à l'établissement de méthodologies spécifiques. Il est donc nécessaire, par construction, de minimiser et de concentrer le nombre de concept à manipuler. Dans le cas de cette étude, cette minimisation est obtenue grâce à un agencement du motif par référencement.

Agencement par référencement

L'agencement par référencement est une forme plus évoluée de l'agencement hiérarchique. Tout comme l'approche hiérarchique, cette approche consiste à concevoir le modèle de domaine autour de concepts structurés sous la forme d'une typologie hiérarchique. Ces concepts sont alors caractérisés par un ensemble de métapropriétés et de métaopérations. Les concepts ne nécessitant pas de caractéristiques particulières sont modélisés en tant que métarôles et non plus en tant que métatypes. Comme le montre la figure 3.5 , certains concepts sont en fait décrits comme des propriétés référencées par des associations et non comme des classes.

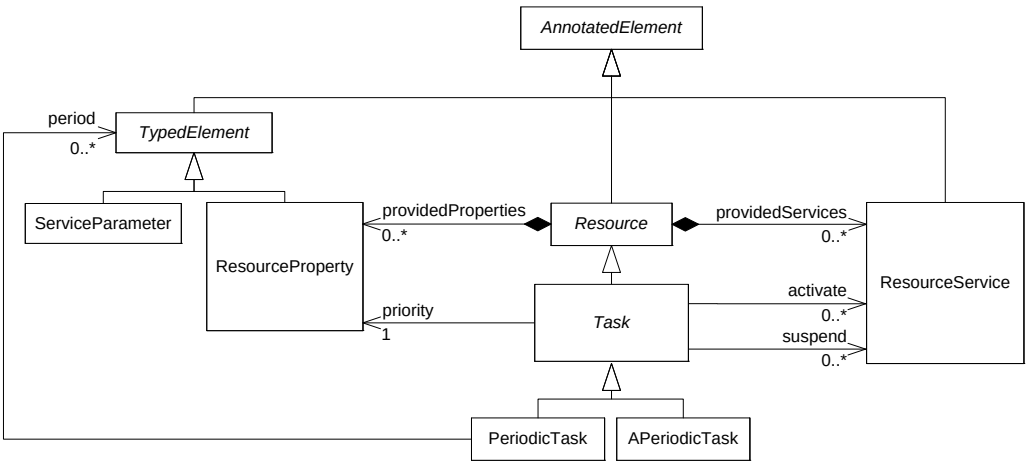

Fig. 3.5 – *Agencement par référencement du modèle de domaine*

Dans la figure 3.5 page ci-contre , les ressources possèdent toujours des propriétés et des services. Ces propriétés et ces services jouent désormais des rôles. Par exemple, une tâche (Task) possède des attributs. Un de ses attributs joue le rôle de priorité. De même, certains de ses services sont utilisés pour activer ou suspendre l'exécution : activate et suspend.

Réduction du nombre de métatypes et limitation de la redondance d'information

Cette approche de conception du métamodèle permet de limiter le nombre de métatypes manipulés et, par agencement, de guider l'utilisation de ces types. En effet, dans le cas ou le métatype ne nécessite pas de propriétés ou d'opérations particulières, elle évite la description d'un métatype et d'une association. Elle limite donc la prolifération des métatypes et la redondance d'information .

Souplesse et liberté de modélisation

De plus, cette approche n'impose pas que l'élément référé soit l'élément possédé. Il est en effet possible de référencer des attributs ou des services n'appartenant pas à la ressource décrite. Cette variation peut être discriminante dans certains contextes de modélisation. Elle répond cependant à un réel besoin dans le cas des plates-formes d'exécution. En effet, comme le montre la figure 3.6 , certaines méthodologies de modélisation peuvent imposer l'utilisation d'interfaces pour modéliser les services et attributs fournis par les ressources. Ainsi même si ce sont les interfaces qui possèdent les attributs et les services à identifier, ce sont les classes qu'il faut identifier comme des ressources. Il est donc nécessaire d'ouvrir le modèle domaine pour pouvoir identifier des caractéristiques qui ne sont pas forcément possédées dans le contexte identifié, c'est-à-dire dans la ressource elle-même.

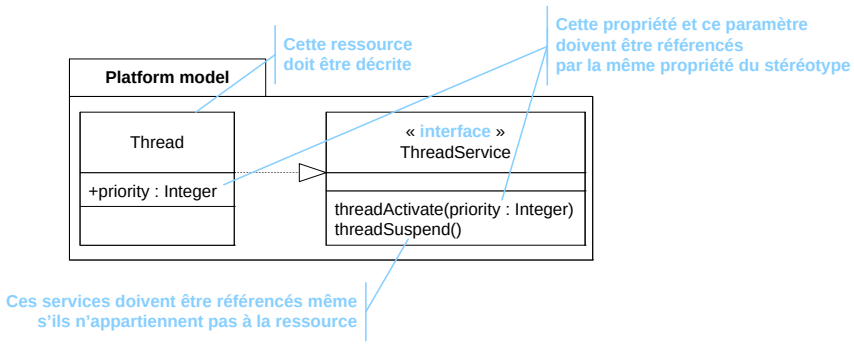

FIG. 3.6 – *Utilisation d'interfaces pour la description d'une plate-forme*

Flexibilité de modélisation

Enfin, cette approche par référencement permet de décrire des éléments de métatypes différents qui exercent des rôles équivalents. Par exemple, le rôle de priorité peut à la fois être joué par un attribut de la ressource et par le paramètre d'un service. Dans certains systèmes d'exploitation par exemple, les services de création de tâche sont paramétrés par la priorité de la tâche. Dans d'autres systèmes, la priorité de la tâche est déclarée statiquement comme un attribut de la tâche. C'est par exemple le cas dans la figure 3.6 ou la priorité d'un Thread est décrite à l'initialisation par l'attribut priority puis est modifiée par le paramètre priority lors de l'activation threadActivate. L'approche par référencement permet ainsi

de capitaliser les rôles joués et donc de minimiser la redondance d'information.

Toutefois, il est vrai que cette approche peut complexifier la compréhension des référencements. Un outil peut alors réduire cette complexité en filtrant, par un système de vue, les métatypes (Property ou Parameter) que l'utilisateur souhaite observer à un instant donné. L'approche par référencement est donc l'approche la plus appropriée pour décrire un métamodèle de plate-forme dans cette étude.

3.2.4 Synthèse du motif

Le modèle de domaine du motif de modélisation dédié à la description des plates-formes logicielles d'exécution est présenté dans la figure 3.7 .

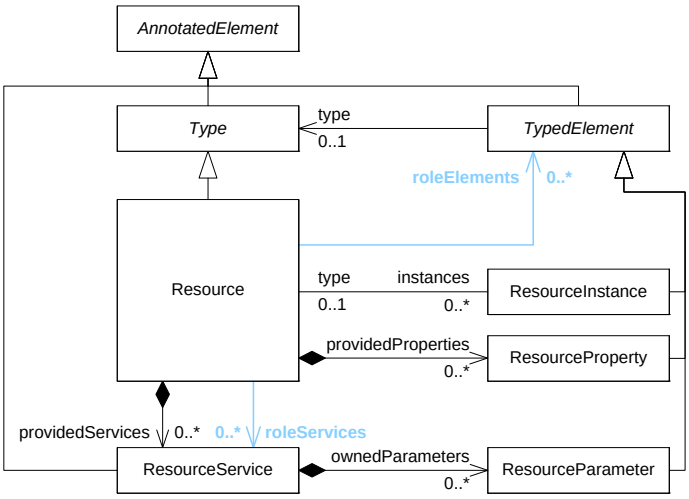

Fig. 3.7 – *Synthèse du modèle de domaine du motif* Resource-Service

Pour qu'un tel motif soit utilisable, il faut désormais l'implanter. Dans cette étude, cette implantation est réalisée sous la forme d'un profil UML.

3.3 Implantation du motif Resource-Service dans UML

Implanter le motif dans UML se concrétise par la description d'un profil UML. Après avoir détaillé les constituants d'un profil UML, il est nécessaire d'étudier dans un deuxième temps le niveau de modélisation des plates-formes (modèle ou métamodèle) puis, dans un troisième temps, de décrire les extensions UML effectives.

3.3.1 Rappel sur les profils UML

Un Profil UML est une extension du métamodèle UML. C'est un package UML qui importe, en lecture seule, le métamodèle UML et plus particu-

lièrement les éléments du métamodèle dont la sémantique et l'utilisation doivent être affinées. Pour cela, les éléments profilés du métamodèle sont étendus par des Stereotype. Ces extensions (Extension), représentées par une flèche noire pleine, spécifient le ou les contextes dans lesquelles le stéréotype peut être appliqué (utilisé).

Présentation du concept de stéréotype

A l'origine, les stéréotypes ont été introduits par Wirfs-Brock *et al.* [61] pour pouvoir classer les éléments selon leurs responsabilités dans le système. L'OMG a étendu l'utilisation des stéréotypes comme un moyen d'étendre des métaclasses UML (un stéréotype hérite de la métaclasse InfrastructureLibrary::Constructs::Class). Ainsi, un stéréotype possède des propriétés (également appelées des Tag (étiquette)), des opérations, des associations et des généralisations. Hormis les généralisations qui ne peuvent être décrites qu'entre stéréotypes, les associations peuvent être établies aussi bien entre les stéréotypes eux mêmes ou entre les stéréotypes et les métaclasses. Dans ce dernier cas (stéréotype - métaclasse), l'association est impérativement navigable dans un seul sens, du stéréotype vers la métaclasse, puisque dans le cas contraire un attribut serait ajouté à la métaclasse ce qui est antinomique avec un import en lecture seule du métamodèle UML.

La figure 3.8 illustre la définition d'un profil UML pour la description d'une tâche périodique possédant une période et une priorité entière héritée d'un stéréotype Task. Cette tâche possède une opération UML jouant le rôle de point d'entrée, c'est-à-dire une routine (une suite séquentielle d'instructions) à exécuter dans le contexte de cette tâche. Ces éléments constituent un profil nommé MultitaskingProfile. Ce profil est **appliqué** sur un modèle nommé Application. La taxonomie de la classe Controller est par exemple affinée en appliquant le stéréotype PeriodicTask, en affectant une valeur dix à la priorité (Priority), cinq à la période (Period) et en précisant le point d'entrée (l'opération) de la tâche (updateControl). Les champs priority, period et entryPoint sont en fait des propriétés du stéréotype renseignées par une ou plusieurs valeurs (Tagged valued au sens UML). Elles sont illustrées en couleur puisque ce sont des métapropriétés et non des propriétés de l'élément stéréotypé.

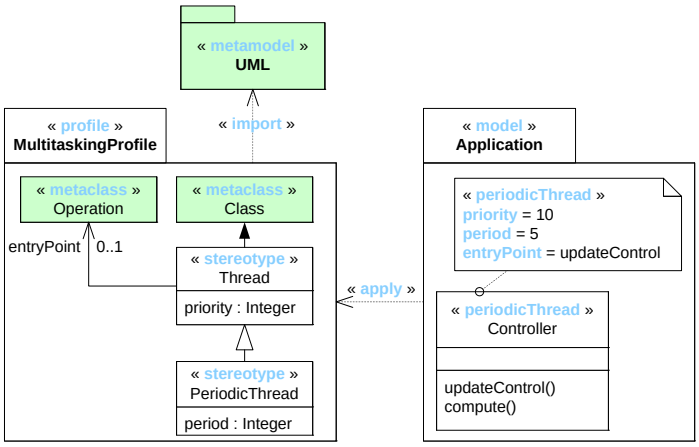

FIG. 3.8 – *Extrait d'un profil* UML *pour la description d'application multitâche*

3.3.2 Choix du niveau de modélisation

UML permet de décrire les plates-formes dans deux niveaux de modélisation distincts que sont a) les profils (au niveau méta) et b) les modèles ou plus particulièrement les librairies de modèles (au niveau modèle).

Modélisation de la plate-forme par un profil

Un profil UML est théoriquement capable de satisfaire les besoins de modélisation structuraux et comportementaux des plates-formes puisqu'un stéréotype est une classe en UML (une classe possède des propriétés et des comportements). La description de la plate-forme comme un ensemble de stéréotype UML est intéressante puisque les éléments stéréotypés sont caractérisés par leurs rôles joués dans le système. Il est ainsi possible de spécialiser un modèle applicatif pour une plate-forme donnée en stéréotypant les éléments de ce modèle applicatif. Par exemple, dans la figure 3.1 page 40 , l'application du stéréotype rtService sur une opération précise que cette opération joue le rôle de service temps réel (elle s'exécute dans le contexte du service temps réel). Les éléments applicatifs sont en fait stéréotypés par les éléments de la plate-forme pour décrire les rôles qu'ils jouent dans la plate-forme. Les méthodologies UML dédiée à l'analyse des systèmes utilisent abondamment cette approche de modélisation [62] [63]. Malheureusement dans le contexte de cette étude, une plate-forme ne peut pas être directement modélisée comme un profil UML puisque la norme interdit l'application de stéréotype sur un stéréotype. Il est par conséquent impossible de décrire un profil UML décrivant une plate-forme par un autre profil UML décrivant un métamodèle de plate-forme.

Pour envisager une modélisation de la plate-forme par un profil UML, un processus en deux étapes (minimum) est nécessaire. Les éléments sont tout d'abord décrits comme un modèle UML puis, par une succession de transformations, ce modèle est promut en un profil UML [57]. Ces transformations sont complexes car il ne suffit pas de promouvoir une classe en un stéréotype et une opération de cette classe en une opération de ce stéréotype. Il faut par ailleurs chercher quelles sont les métaclasses d'UML à étendre, restructurer les éléments du profil et s'assurer de la cohérence du profil vis à vis du métamodèle UML. Que la description de la plate-forme est implicite, elle nécessite la description d'une librairie de modèle UML explicite.

Modélisation de la plate-forme par une librairie de modèles

En UML, la librairie de modèle (ModelLibrary) est un package particulier qui se rapporte aux librairies des langages de programmation. Elle est utilisée pour décrire des éléments qui sont réutilisés par différents packages. C'est le cas des éléments d'une plate-forme d'exécution qui sont utilisés par plusieurs packages applicatifs.

La modélisation d'une plate-forme par une librairie de modèle est une modélisation «classique» UML avec par exemple des diagrammes de classes, des diagrammes de structures composites et des diagrammes de comportements. Les mécanismes sont décrits comme des classes et les services comme des opérations. L'application importe la librairie, instancie les mécanismes et appelle les services (voir la figure 3.1 page 40). Les

extensions du métamodèle UML concernent donc tout au moins les métaclasses Class et Opération. Elles sont détaillées dans la section suivante.

3.3.3 Extension du métamodèle UML

L'implantation du motif RESOURCE-SERVICE) en un profil UML suit un ensemble de règles explicitées ci-dessous :

1. Un élément AnnotatedElement est un UML::kernel::Classes::Element. Un Element peut être commenté, annoté par des stéréotypes et contraint,
2. Une ResourceProperty est une propriété UML (UML::Kernel::Classes::Property),
3. Un ServiceParameter est un paramètre UML (UML::Kernel::Classes::Parameter),
4. Un Service est une opération UML (UML::Kernel::Classes::Operation) puisque un paramètre est possédé par une opération,
5. Une Ressource fait l'objet d'un stéréotype à part entier car il apporte de nouvelle association vers les classes TypedElement et Operation d'UML. Ce stéréotype étend le concept de classe (UML::Kernel::Classes::Class) puisque qu'il est le premier élément hiérarchique dans le métamodèle UML à posséder des propriétés et des services. Le stéréotype Resource ne possède par d'icône particulière puisqu'il est destiné à être réifié pour un domaine donné. Son extension avec la métaclasse Class n'est pas exigée (isRequired=false) ce qui permet à l'utilisateur de décrire des classes dans le modèle de plate-forme qui ne sont pas pour lui des ressources. Les machines à états sont des exemples classiques de ces classes qui ne sont pas forcément considérées comme des ressources d'exécution en tant que telle (la métaclasse ProtocoleStateMachine hérite de la métaclasse Class dans UML),
6. Les rôles providedProperties, providedServices et providedParameters sont implantés par le métamodèle UML lui même, par les rôles ownedAttribute ownedOperation et ownedParameter,
7. Les associations entre Resource et TypedElement ne sont navigables que dans un seul sens, c'est-à-dire que le rôle associé à cette association est possédé par Resource, par conséquent ce rôle n'est pas transformé en un stéréotype mais en une propriété du stéréotype Resource. Cette propriété est typée par l'élément UML::Kernel::Classes::TypedElement,
8. Les associations entre Resource et Service ne sont navigables que dans un seul sens, c'est-à-dire que le rôle associé à cette association est possédé par Resource, par conséquent ce rôle n'est pas transformé en un stéréotype mais en une propriété du stéréotype Resource. Cette propriété est alors typée par l'élément UML Operation et
9. Les instances des ressources ResourceInstance sont des spécifications d'instances d'UML (UML::Kernel::Classes::InstanceSpecification). Une règle OCL est nécessaire pour limiter le nombre de classes par instance. Une instance est associée à au plus une classe contrairement

au métamodèle UML ou une instance peut être associée à plusieurs classes.

La figure 3.9 illustre l'implantation du motif Resource-Service dans un profil UML.

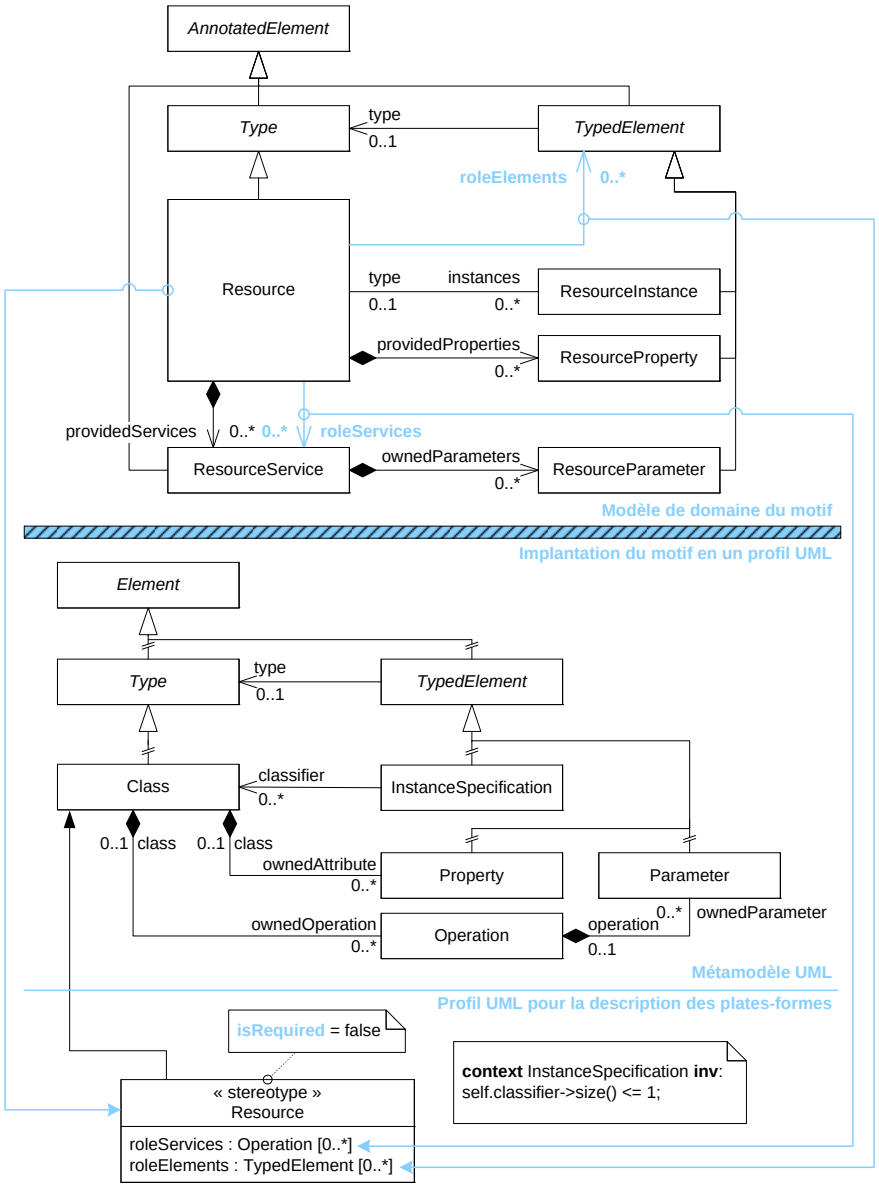

Fig. 3.9 – *Implantation du motif* Resource-Service *dans un profil UML*

3.4 Discussion

Le motif Resource-Service *est-il une plate-forme ?*

Le motif Resource-Service est destiné à aider la création de métamodèles dédiés à la modélisation des plates-formes d'exécution. Le principal risque de description d'un tel motif c'est qu'il ne permette pas de décrire

un métamodèle de plate-forme mais qu'il soit intrinsèquement une plate-forme à part entière, c'est-à-dire une plate-forme abstraite décrite au niveau méta. D'autre travaux liés à UML tels que [10], [16] ont par exemple proposé des plates-formes abstraites au niveau méta. La figure 3.10 reproduit leurs approches, nommée ici Littérature. Elle compare en partie gauche un motif de la littérature et en partie droite une implantation du motif RESOURCE-SERVICE (Resource-Service).

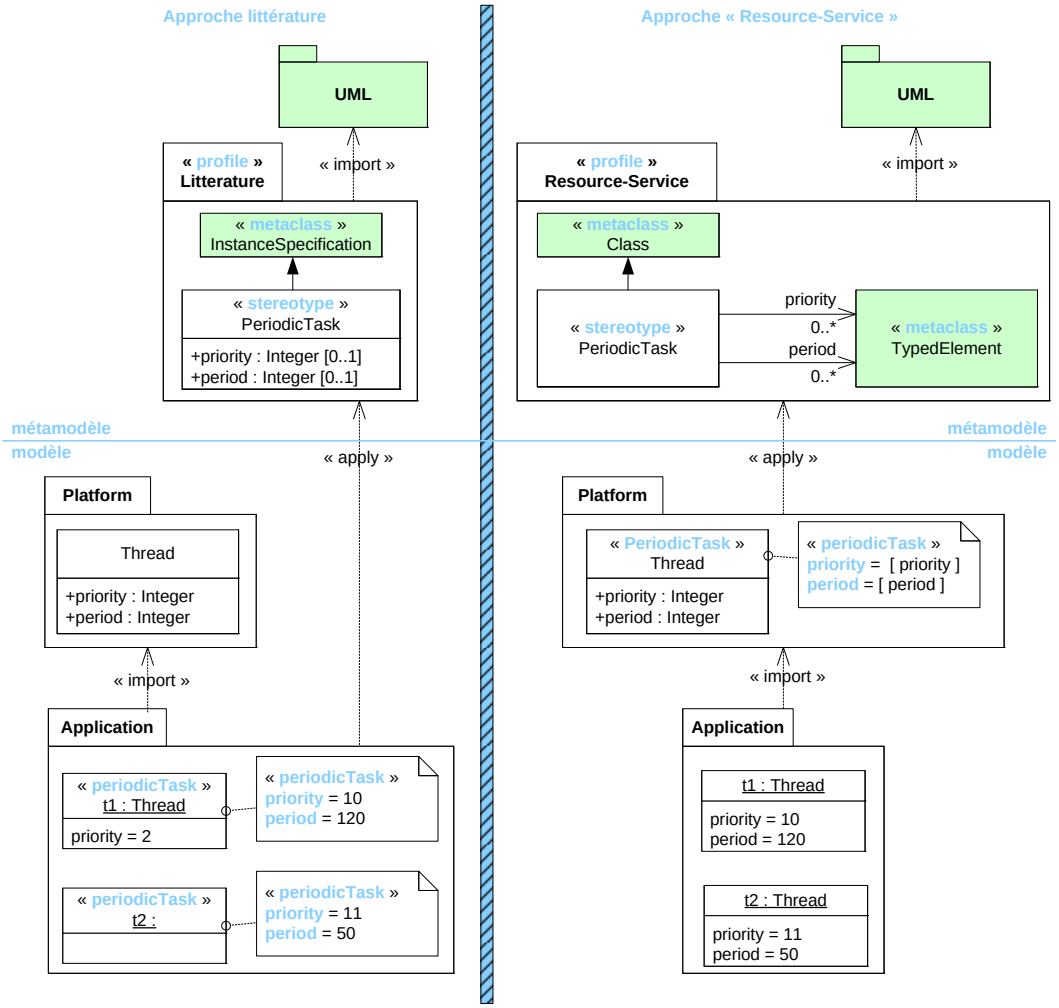

FIG. *3.10 – Positionnement du motif* RESOURCE-SERVICE *vis à vis des approches de la littérature*

Les motifs de la littérature ne sont pas des métamodèles de plates-formes

Il apparait dans l'approche dite de la littérature que le métamodèle n'est pas un langage pour décrire des plates-formes d'exécution. En effet dans la partie gauche de la figure 3.10 , les propriétés des ressources sont d'ores et déjà typées par des types primitifs. Dans le profil résultant le stéréotype étend la métaclasse InstanceSpecification et est utilisé pour décrire les caractéristiques de l'application et non les caractéristiques de la plate-forme. Par exemple, la priorité de l'instance t1 est précisée par les propriétés du stéréotype et non par les propriétés de son type Thread.

Il est important de noter que les valeurs des métaattributs n'appar-

Caractéristique modèle versus caractéristique méta

tiennent pas «physiquement» aux entités du modèle. L'entité nommé t2 dans la partie gauche de la figure 3.10 page précédente n'a aucune «connaissance» de sa priorité puisqu'elle n'est pas définie dans le modèle. Aucun service de sa classe ne peut accéder à la priorité décrite dans le stéréotype puisque cette information n'appartient pas au modèle mais au métamodèle. Même si ces informations sont décrites visuellement dans le modèle sous la forme d'une note textuelle, celles-ci restent des métapropriétés, c'est-à-dire que ce ne sont pas des propriétés de l'élément modélisé. Ce sont des propriétés utilisées pour décrire les métacaractéristiques de l'entité modélisée (ici la notion d'instance UML). Pour connaître cette information, il faut être capable de faire de l'introspection sur les entités, c'est-à-dire d'être capable de manipuler des caractéristiques du langage dans le contexte d'une entité d'un modèle. Si certains langages comme le JAVA le permettent, il n'y a rien de natif dans l'IDM et plus particulièrement dans UML. La règle OCL 3.1 montre l'utilisation des opérations JAVA pour y parvenir, opérations qui sont propres à l'outil UML2 d'ECLIPSE et non à OCL ou à UML.

```
context UML!Element def : getTagValue(
          stereo : UML!Stereotype,
          tag : UML!Property) :
  UML!Element =
  self.getValue(self.getAppliedStereotype(stereo.name),tag.name);
```

Code 3.1 – Expression OCL de la littérature pour l'obtention des métavaleurs d'un stéréotype

Incohérence des modélisations

Outre ce problème d'information méta et non modèle, ce style de modélisation peut conduire très rapidement à des incohérences comme l'illustre la figure 3.10 page précédente . Par exemple, une valeur est affectée à la propriété priority de la classe Thread. Cependant, cette valeur est différente de celle spécifiée dans la métapropriété priority du stéréotype RtService. Il n'existe aucun lien entre ces deux informations même si fondamentalement elle exprime la même sémantique, c'est-à-dire la priorité de la tâche.

Détournement des stéréotypes

Enfin, dans le cas de gauche, l'application d'un stéréotype sur un élément UML n'est pas une classification (une conformité) comme le préconisait à l'origine Wirfs-Brock *et al.* [61]. Ce n'est pas un rôle joué par un élément du modèle mais une instanciation d'un mécanisme décrit au niveau méta. Dans le cas du service, c'est un appel sur un service décrit au niveau méta. Par exemple, dans la figure 3.10 page précédente , le type de l'élément t2, c'est-à-dire de la spécification d'une instance t2 au sens d'UML, n'est pas défini explicitement dans le modèle. Ce n'est pas une faute de modélisation puisque le métamodèle UML le permet. En fait, l'application du stéréotype a implicitement «typé» cette instance t2 comme étant une PeriodicTask de la plate-forme Litterature. t2 est une tâche au sens du profil. L'utilisation de stéréotype pour «typer» les instances décrites dans le modèle n'est pas une approche native d'UML. Par conséquent, les outils manipulant ces modèles de plates-formes devront être adaptés pour pouvoir déterminer automatiquement quel est le type de cette instance. Il ne suffira pas d'interroger la propriété classifier d'une InstanceSpecification mais il faudra aussi se soucier des stéréotypes appliqués sur cette instance

en espérant que tous les stéréotypes de l'application soient utilisés comme des métatypes et non comme des annotations guidant les transformations outillées. Les figures 3.2 et 3.3 illustrent la description de la même règle OCL pour chacune des approches de la figure 3.10 page 53 . Appelées dans le contexte d'une instance, ces deux fonctions retournent la liste des classes de cette instance. Dans le cas de la littérature, il faut agréger (primitive include) les classes spécifiées par la métapropriété classifier et les stéréotypes appliqués. Dans l'approche du motif, seul la métapropriété suffit.

```
context UML::InstanceSpecification def:getClassifiers():
  Sequence(UML::Classifier) =
  self.classifier->including(self.getAppliedStereotypes()).flatten();
```

Code 3.2 – Expression OCL de la littérature pour l'obtention des classes d'une instance

```
context UML::InstanceSpecification def:getClassifiers() :
  Sequence(UML::Classifier) =
  self.classifier;
```

Code 3.3 – Expression OCL de cette étude pour l'obtention des classes d'une instance

Plate-forme implicite versus métamodèle de plate-forme

En somme, le profil Litterature est une nouvelle API permettant de décrire des applications, c'est-à-dire une plate-forme implicite. Ce n'est pas une abstraction d'un langage pour décrire des plates-formes, c'est-à-dire un métamodèle, mais la description d'une nouvelle plate-forme décrite au niveau méta. Le motif Resource-Service n'est justement par cette nouvelle API. Ce n'est pas une plate-forme implicite mais un motif pour décrire des plates-formes. Le profil construit autour du motif Resource-Service décrit bien les ressources de la plate-forme et non l'utilisation de ces ressources pour décrire l'application. Le lecteur pourra noter que le profil de droite (Resource-Service) peut être utilisé pour décrire le profil de gauche (Litterature) dans la figure 3.10 page 53 . L'inverse est par contre impossible.

Conclusion

L'objectif principal de ce chapitre était de décrire les briques élémentaires d'un métamodèle de plate-forme et leurs agencements, c'est-à-dire le cœur d'un métamodèle de plate-forme logicielle d'exécution.

Pour cela, un motif nommé Resource-Service a été défini. Il peut être utilisé pour concevoir un métamodèle générique de plate-forme logicielle ou pour la conception d'un métamodèle de plates-formes logicielles spécifiques à des domaines donnés. Il doit être utilisé, appliqué et réifié lors de la conception d'un métamodèle de plate-forme. Dans le contexte de cette étude, un tel métamodèle est un profil UML. Des règles d'implantation de ce motif dans un profil UML ont donc été définies. Cette implantation UML (profil+métamodèle) répond aux besoins de modélisation de la taxonomie identifiés dans la section 2.1.2 page 27. La figure 3.11 page suivante illustre la satisfaction de ces besoins.

La proposition de ce motif est une contribution originale par rapport aux travaux de la littérature. Il a donc fait l'objet de deux publications. Une première au sein de la conférence ECRTS (Euromicro Conférence on Real-Time Systems) dans la section des travaux en cours [64] et une seconde dans la revue nationale Objet RTSI [65].

Avant d'être appliqué sur un métamodèle de plate-forme d'exécution réel (chapitre 6 page 91), ce motif doit être confronté au différent cas de modélisations. Il en résultera des heuristiques de modélisations utilisables dans les méthodologies de modélisation appliquant le motif Resource-Service.

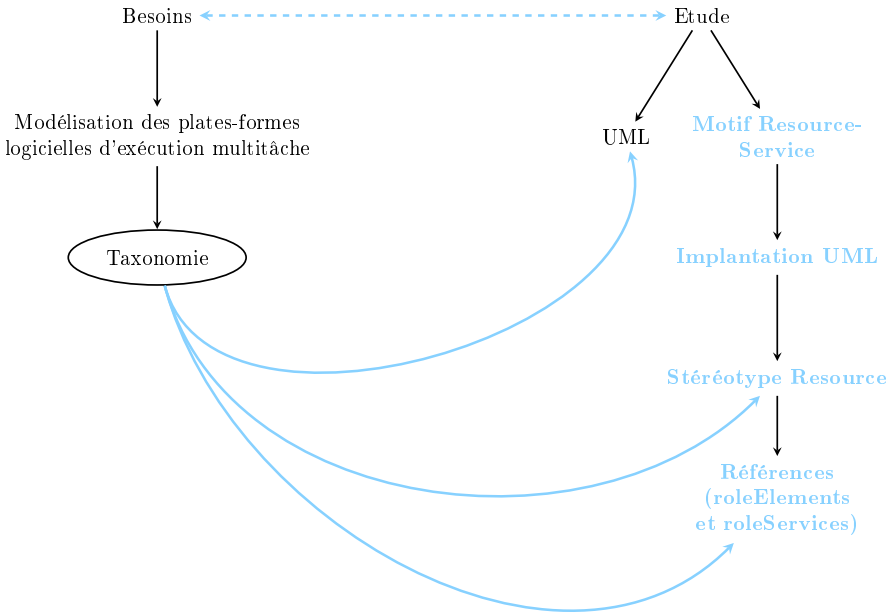

Fig. 3.11 – *Positionnement du motif* Resource-Service *vis à vis des besoins de modélisation*

4 Spécification d'un cadre méthodologique dédié au motif Resource-Service

Sommaire

Ce chapitre vise à étudier l'impact du motif Resource-Service sur la modélisation des plates-formes. Plus particulièrement, il doit définir les heuristiques de modélisation constituant le cœur d'une méthodologie dédiée la modélisation des plates-formes. Ce cœur doit ensuite être étendu pour des méthodes métiers, c'est-à-dire des méthodes de modélisation adéquates pour un contexte de développement donné.

Pour cela, le travail est divisé en deux étapes. Premièrement, les cas de modélisation auquel sont confrontés les métamodèles de plates-formes sont identifiés. Deuxièmement, les heuristiques de modélisation sont déduites pour chacun de ces cas. Enfin, dans une troisième et dernière section, ces heuristiques et dépendances sont discutées.

4.1 Identification des cas de modélisation

Un métamodèle de plate-forme appliquant le motif Resource-Service est utilisé pour modéliser des plates-formes logicielles d'exécution. Ce métamodèle représente une ontologie des plates-formes logicielle d'exécution permettant de décrire des modèles de plates-formes différents [66]. Cette utilisation pose le problème de l'exhaustivité du métamodèle. L'ontologie qu'il représente rassemble les concepts communs du domaine (des plates-formes). Cependant, chaque plate-forme possède ses propres caractéristiques et ses propres spécificités. Ainsi, un ou plusieurs éléments du modèle de la plate-forme peuvent être décrits par un ou plusieurs éléments du métamodèle. Réciproquement, un ou plusieurs éléments du métamodèle peuvent être utilisés pour décrire un ou plusieurs éléments de la plate-forme. La figure 4.1 illustre chacun de ces cas. Ils sont détaillés ci-dessous :

- 1 élément de la plate-forme est décrit par 1 élément du métamodèle ($1 \mapsto 1$),
- 1 élément de la plate-forme est décrit par M éléments du métamodèle ($1 \mapsto M$) avec M un entier tel que $M \geq 2$,
- N éléments de la plate-forme sont décrits par 1 élément du métamodèle ($N \mapsto 1$) avec N un entier tel que $N \geq 2$,
- N éléments de la plate-forme sont décrits par M éléments du métamodèle ($N \mapsto M$) avec N et M deux entiers tel que $N \geq 2$ et $M \geq 2$.
- Aucun élément de la plate-forme est décrit par M éléments du métamodèle ($0 \mapsto M$) avec M un entier tel que $M \geq 1$,
- N élément de la plate-forme est décrit par aucun (0) élément du métamodèle ($N \mapsto 0$) avec N un entier tel que $N \geq 1$,

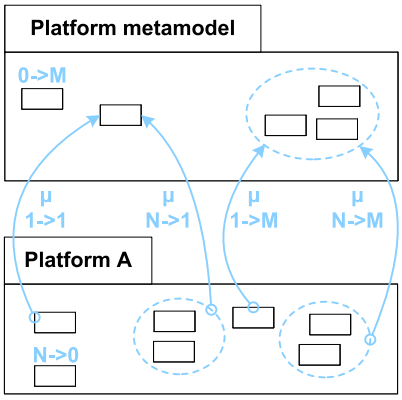

Fig. 4.1 – *Illustration des cas de modélisation auxquels sont confrontés les métamodèles de plates-formes*

Pour chacun de ces cas, des heuristiques de modélisation dédiées au motif Resource-Service peuvent être définies.

4.2 Définition des heuristiques de modélisation

Dans cette étude, la réponse du motif Resource-Service aux différents cas de modélisation doit être étudiée pour a) les ressources, b) les éléments typés restreints aux propriétés ainsi qu'aux paramètres et c) les services.

4.2.1 Définition des heuristiques de modélisation pour les ressources

Correspondance $1 \mapsto 1$

La modélisation d'une ressource de la plate-forme par un stéréotype du métamodèle se concrétise par l'application d'un stéréotype, c'est-à-dire par la relation de conformité, χ, entre la ressource et le stéréotype. Cette relation de conformité indique que la sémantique du stéréotype dénote la sémantique de la ressource. Par exemple, dans la figure 4.2 , la sémantique du concept (PeriodicThread) est décrite par la sémantique dénotant le concept de tâche périodique (PeriodicTask) du métamodèle de plate-forme. La sémantique de PeriodicThread est conforme à la sémantique de PeriodicTask :

$$\text{PeriodicThread } \chi \text{ PeriodicTask} \tag{4.1}$$

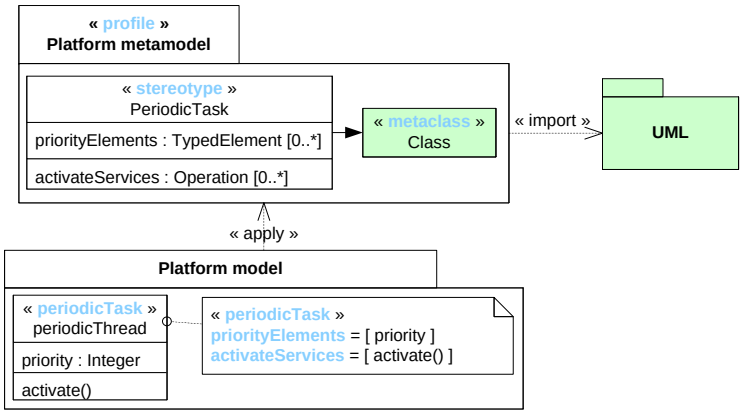

Fig. 4.2 – *Illustration de la correspondance* $1 \mapsto 1$ *pour les ressources*

Correspondance $1 \mapsto M$

La modélisation d'une ressource de la plate-forme par plusieurs éléments du métamodèle se concrétise par l'application de plusieurs stéréotypes sur cette ressource, c'est-à-dire par une conformité multiple[1]. Dans la

[1] Bien que la conformité multiple soit théoriquement possible dans les approches DSL, dans la plupart des implantations actuelles (l'infrastructure EMF d'Eclipse par exemple) cette relation est implantée par la relation d'instanciation Java. Le métamodèle est implanté comme un ensemble de classe Java. Les éléments du modèle sont des instances de ces classes. Or en Java, une instance ne peut avoir qu'une seule classe et par conséquent la relation de conformité est simple.

norme UML, l'application de plusieurs stéréotypes est un point de variation sémantique. Pour la modélisation des plates-formes, un mécanisme pour figer ce point de variation est l'opérateur de composition qui est notée $\otimes$ dans cette étude. Une application multiple est la composition de chacune des applications unitaires. En d'autres termes, la sémantique de la ressource modélisée est la composition des sémantiques dénotant chacun des stéréotypes appliqués. Par exemple dans la figure 4.3 , une ressource PeriodicProcess stéréotypée doublement par PerioricTask et MemoryPartition est une composition du concept de tâche périodique et du concept de partition mémoire au sens du métamodèle tel que :

$$\text{PeriodicProcess} \; \chi \; (\text{PeriodicTask} \otimes \text{MemoryPartition}) \tag{4.2}$$

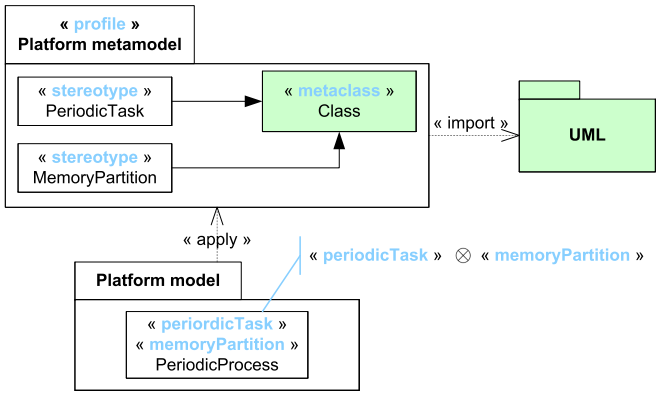

FIG. 4.3 – *Illustration de la correspondance* $1 \mapsto M$ *pour les ressources*

Correspondance $N \mapsto 1$

La modélisation de plusieurs éléments de la plate-forme par un seul stéréotype est un cas trivial puisqu'il illustre l'application du même stéréotype sur différents éléments du modèle.

Correspondance $N \mapsto M$

Cette correspondance est une composition des cas $N \mapsto 1$ et $1 \mapsto M$. Elle se traduit par l'application de plusieurs stéréotypes sur plusieurs éléments.

Correspondance $0 \mapsto M$

Cette correspondance correspond au cas ou il est intéressant de mettre en œuvre plusieurs ressources dont la sémantique globale correspond à la sémantique d'un ou plusieurs concepts du métamodèle. Par exemple, cela permet de décrire des motifs de conception multitâche [67].

En UML, il existe deux approches principales pour décrire des motifs de conception. La première consiste à décrire des collaborations et la seconde à décrire des types abstraits qui ne possèdent pas d'implantation spécifiques dans la plate-forme mais qui agrège les éléments à mettre en œuvre pour concevoir cette fonctionnalité.

Les collaborations (se référer au chapitre neuf de [37]) sont les supports de modélisation des motifs de conception dans UML. Une collaboration conceptualise un ensemble de classe collaborant entres elles pour satisfaire un objectif précis. Elle décrit les liens requis entre les instances de ces classes. Ces instances jouent alors des rôles dans le contexte de la collaboration. Ces rôles sont représentés par des propriétés, appelées des parties Part. Ces parties spécifient les caractéristiques structurelles des classes dans le contexte de cette collaboration (par des valeurs par défaut). Elles sont liées par des connecteurs qui définissent les interactions requises entre les instances collaborantes. Les parties et les connecteurs sont respectivement typés par des classes et des associations du modèle d'un modèle de classe associé (le diagramme Composite en UML). Les parties et les extrémités des connecteurs possèdent des multiplicités. La figure 4.4 illustre la description structurelle d'un motif de conception d'une tâche périodique sur une plate-forme ne fournissant que les concepts de tâches, d'événements et d'alarmes. Dans ce motif une alarme signale périodiquement un événement à une tâche. La période de la tâche est alors le temps entre les réceptions de deux occurrences de cet événement. La description de cette collaboration par le métamodèle de plate-forme se concrétise par l'application du stéréotype PeriodicTask. La sémantique du stéréotype appliqué dénote la sémantique de la collaboration tel que :

$$(\text{Thread} \otimes \text{Event} \otimes \text{Alarm}) \; \chi \; \text{PeriodicTask} \tag{4.3}$$

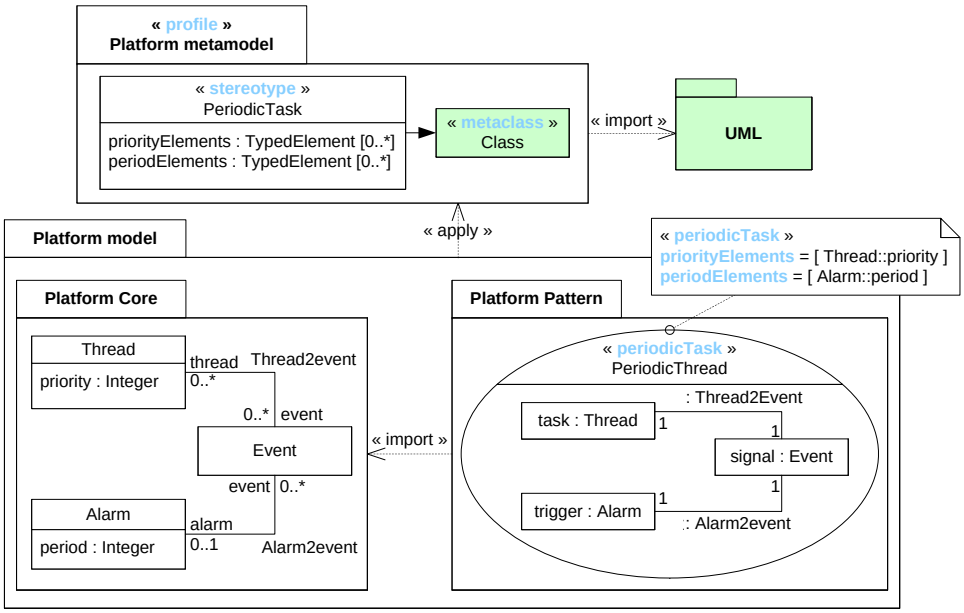

FIG. 4.4 – *Utilisation des collaborations pour illustrer la correspondance* $0 \mapsto M$

Bien que les collaborations soient adéquates pour des modélisations structurelles, elles ne peuvent pas posséder d'opération ce qui limite leurs utilisations dans le contexte du motif RESOURCE-SERVICE (le motif référence des opérations). Une alternative est l'utilisation de classes structurées. Ces classes possèdent des parties (Part) et décrivent des opérations Operation. Par exemple dans la figure 4.5 , le motif d'initialisation d'une tâche périodique est modélisé. Ce motif d'initialisation est décrit par une activité constituée des actions de création de l'événement, d'activation de la tâche et enfin d'activation de l'alarme. Tout comme les classes utilisées pour décrire l'implantation, cette activité est abstraite puisqu'elle ne possède pas d'implantation native dans la plate-forme.

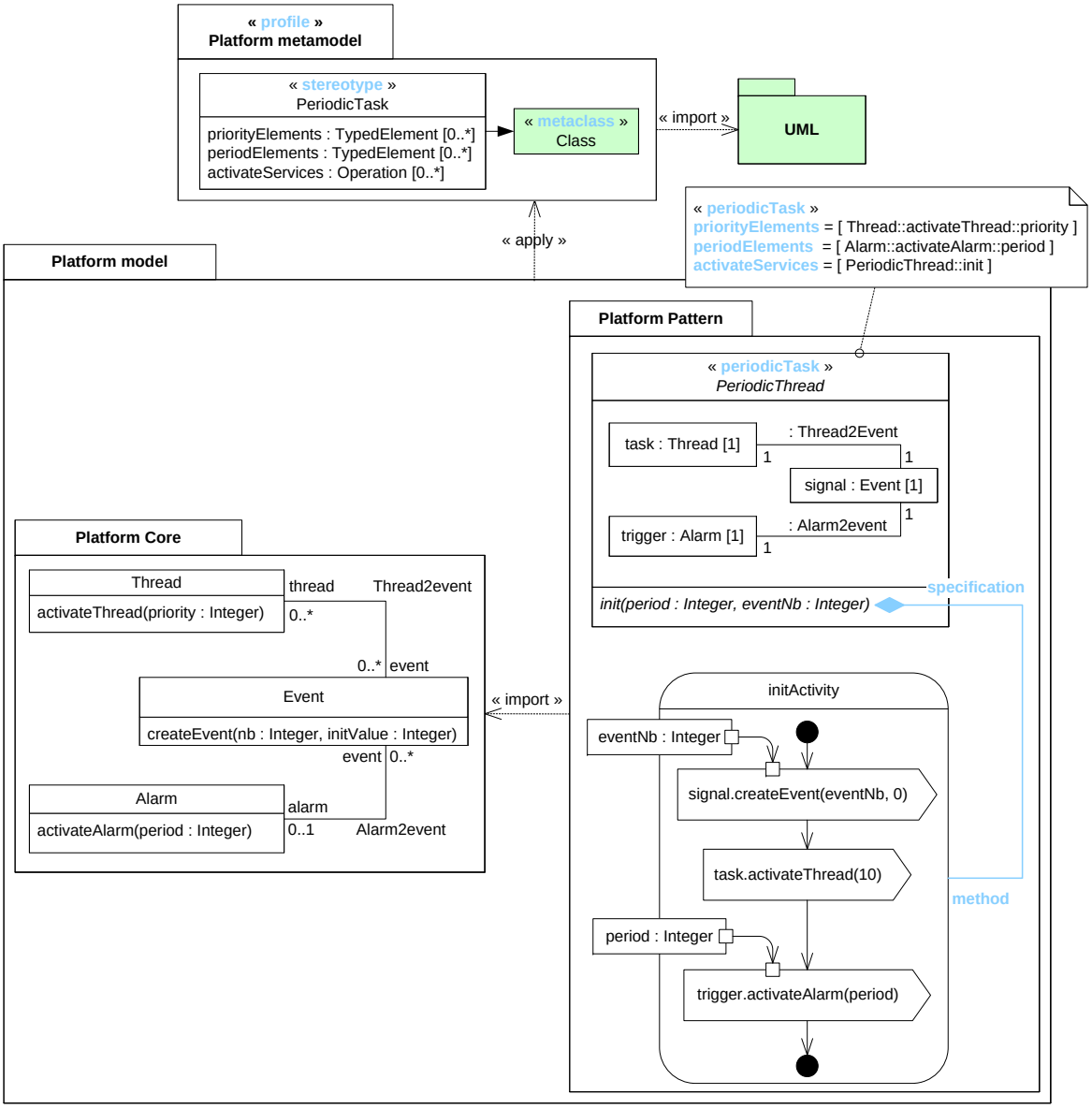

FIG. 4.5 – *Utilisation des classes composées pour illustrer la correspondance* $0 \mapsto M$

Correspondance $N \mapsto 0$

Outre la non modélisation de ces ressources par le métamodèle (faiblesse du métamodèle), la modélisation de ressources qui ne correspondent à aucun élément du métamodèle de plate-forme est possible si et seulement si le comportement de cette ressource est simulé par la composition des comportements des autres ressources de la plate-forme qui sont explicitement décrites par le métamodèle. L'utilisation de classes structurées peut permettre de satisfaire ce cas. Ces classes possèdent alors des parties qui référencent des ressources décrites par le métamodèle. Ces parties sont privées car elles abstraient une implantation possible du mécanisme. La figure 4.6 illustre la description structurelle d'une tâche périodique dont le comportement est déduit de la collaboration de chacune des ressources référencées.

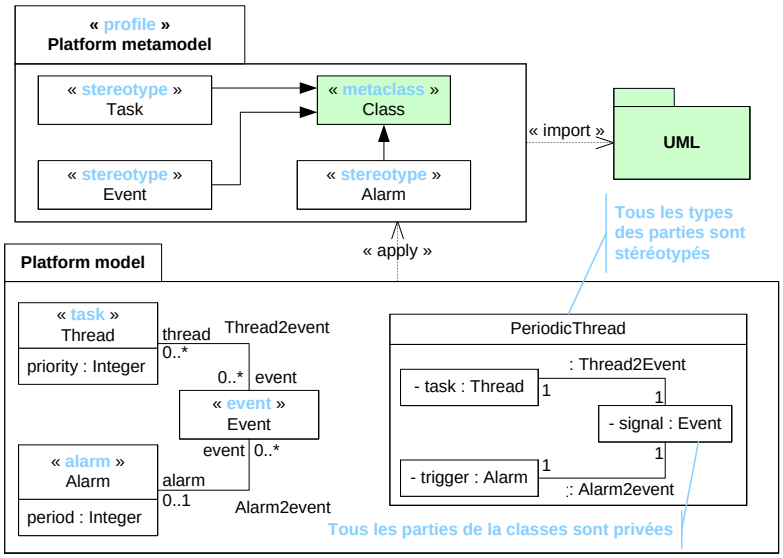

FIG. 4.6 – *Illustration de la correspondance* $N \mapsto 0$ *pour les ressources*

4.2.2 Définition des heuristiques de modélisation pour les éléments typés

Correspondance $1 \mapsto 1$

La description d'une propriété ou d'un paramètre de la plate-forme par un élément du métamodèle est intrinsèque au motif RESOURCE-SERVICE. Elle se traduit par le référencement de cet élément typé par une propriété d'un stéréotype. La sémantique de ce référencement est l'égalité mathématique (=). Une valeur (une instance) de l'élément typé est égale à une valeur de la propriété du stéréotype. Par exemple, dans la figure 4.2 page 60 , la propriété priority est référencée par la propriété priorityElements, c'est-à-dire que mathématiquement la valeur référencée est égale à la valeur affectée à l'attribut tel que :

$$| \, priorityElements \, | \; = \; | \, priority \, | \tag{4.4}$$

Correspondance $1 \; \mapsto \; M$

La représentation d'une propriété ou d'un paramètre par plusieurs propriétés d'un stéréotype est intrinsèque au motif. Dans la figure 4.7 , la ressource PeriodicThread est caractérisée par une échéance Deadline. Dans cette plate-forme (fictive), la politique d'ordonnancement est une politique hors ligne à échéance. La priorité d'un PeriodicThread est donc définie par son échéance. L'attribut Deadline est alors référencé aussi bien comme une échéance que comme une priorité (PriorityElements et DeadlineElements). La sémantique de ce référencement est là aussi l'égalité mathématique :

$$\begin{aligned} | \, priorityElements \, | \; &= \; | \, deadline \, | \\ | \, deadlineElements \, | \; &= \; | \, deadline \, | \end{aligned} \tag{4.5}$$

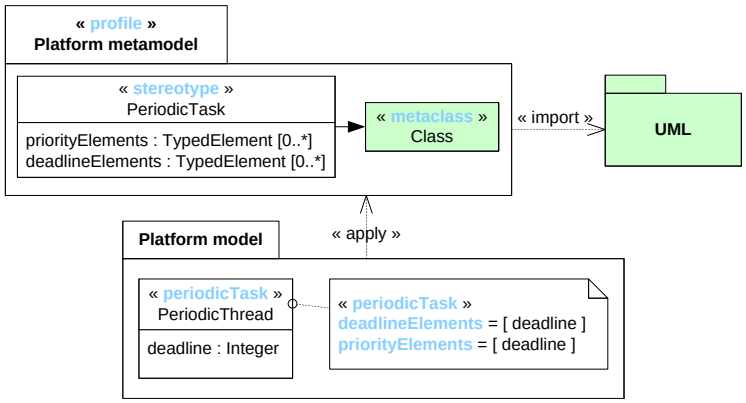

FIG. *4.7 – Illustration de la correspondance* $1 \; \mapsto \; M$ *pour les éléments typés*

Correspondance $N \; \mapsto \; 1$

Le motif RESSOURCE-SERVICE répond nativement à ce cas de figure puisqu'il autorise qu'une propriété du stéréotype référence plusieurs éléments du modèle. La figure 4.8 page suivante illustre par exemple le cas ou à la fois une propriété et un paramètre d'un service sont utilisés, dans des contextes différents, pour spécifier l'échéance d'une tâche périodique.

$$\begin{aligned} | \, deadlineElements \, | \; &= \; | \, deadline \, | \\ | \, deadlineElements \, | \; &= \; | \, deadlineParam \, | \end{aligned} \tag{4.6}$$

Correspondance $N \; \mapsto \; M$

Cette correspondance est une composition des cas $N \; \mapsto \; 1$ et $1 \; \mapsto \; N$. Elle se traduit par un référencement multiple et par des références multiples.

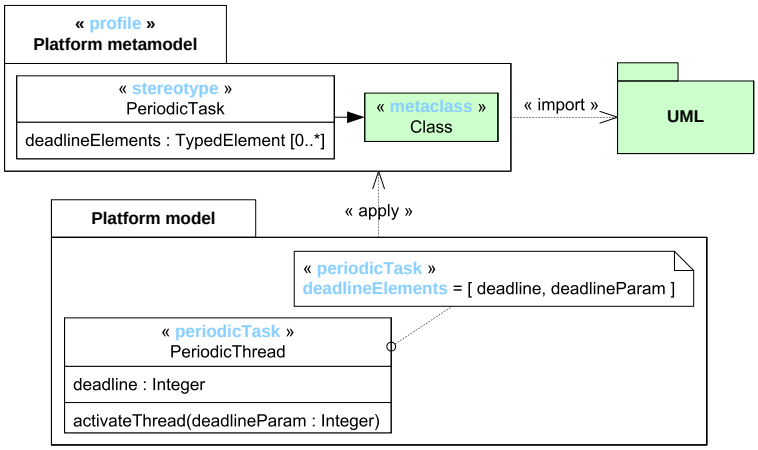

FIG. 4.8 – *Illustration de la correspondance* $N \mapsto 1$ *pour les éléments typés*

Correspondance $0 \mapsto M$

Ce cas de figure apparaît lorsqu'un ou plusieurs éléments de la plate-forme sont impliqués dans une expression mathématique dont le résultat doit être référencé par le métamodèle. Par exemple dans la figure 4.9 page ci-contre , la priorité d'un PeriodicThread est issue de formules mathématiques. Le choix de la formule dépend de la politique de l'ordonnanceur Scheduler. Lorsque la politique est monotone sur la période (Rate monotonic (RM)), la valeur de la priorité est l'inverse de la période, sinon la priorité est l'inverse de l'échéance (politique monotone sur l'échéance, Deadline Monotonic (DM)). La propriété du stéréotype doit alors référencer le résultat de la formule mathématique, c'est-à-dire le résultat d'une expression. En UML, une expression est un élément typé, par conséquent, une propriété d'un stéréotype peut référencer cette expression. Cette expression peut être, soit composée d'opérandes et d'opérateurs (Expression), ou soit opaque (OpaqueExpression). Dans ce dernier cas les langages JAVA ou OCL peuvent être par exemple utilisés. La figure 4.9 page suivante illustre l'utilisation d'une OpaqueExpression spécifiée en OCL pour décrire l'expression relative à la priorité.

$$| \, priorityElements \, | \; = \; | \, priorityExpression \, | \tag{4.7}$$

Correspondance $N \mapsto 0$

Tout comme le cas précédent des Expression ou des OpaqueExpression peuvent être utilisées. Dès lors, les modélisations de propriétés et de paramètres qui ne correspondent à aucun élément du métamodèle de plate-forme sont possibles si et seulement si l'expression, dénotant cette propriété ou ce paramètre, référence des éléments identifiés par le métamodèle. Par exemple dans la figure 4.10 page ci-contre , le métamodèle ne propose aucun élément pour identifier la période d'une tâche. Dans la plate-forme modélisée, la période est toujours définie comme l'instant sui-

vant l'échéance de la tâche. Cette échéance est identifiée par le métamodèle, par conséquent il est possible de décrire la sémantique de la période par une expression référençant l'échéance de la tâche. Cette expression peut ainsi permettre de générer une valeur pour la période même si cette propriété n'est pas explicitement identifiée par le métamodèle.

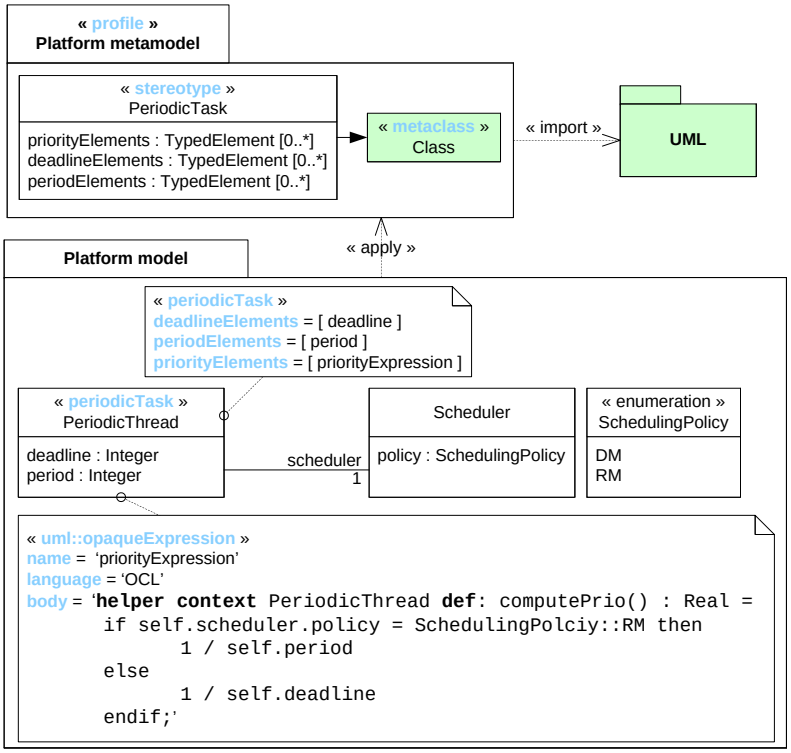

FIG. 4.9 – *Illustration de la correspondance* $0 \mapsto M$ *pour les éléments typés*

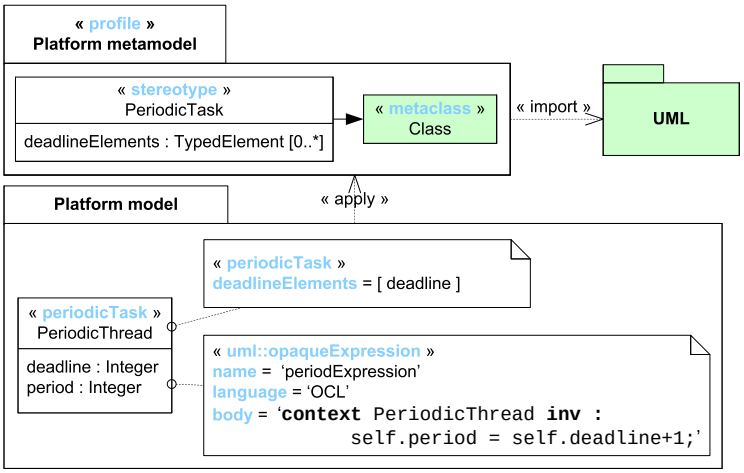

FIG. 4.10 – *Illustration de la correspondance* $N \mapsto 0$ *pour les éléments typés*

4.2.3 Définition des heuristiques de modélisation pour les services

Correspondance $1 \mapsto 1$

La modélisation d'un service de la plate-forme se concrétise par le référencement de ce service par une propriété d'un stéréotype. Dans ce cas, la sémantique dénotant la propriété du stéréotype décrit la sémantique du service référencé. Par exemple, dans la figure 4.2 page 60 , le service threadActivate est référencé par la propriété activateServices. Au sens du métamodèle, il permet donc d'activer une tâche périodique.

Correspondance $1 \mapsto M$

Un service peut être référencé nativement par plusieurs propriétés d'un stéréotype. Dans la figure 4.11 , le service threadSpawn permet soit d'activer une tâche déjà créée ou bien de créer une tâche qui n'est pas active. Du point de vue du métamodèle, le même service est utilisé pour créer et pour activer une tâche périodique (PeriodicThread). De même, au sens du métamodèle, la création et l'activation d'une tâche périodique nécessite un double appel au service threadSpawn.

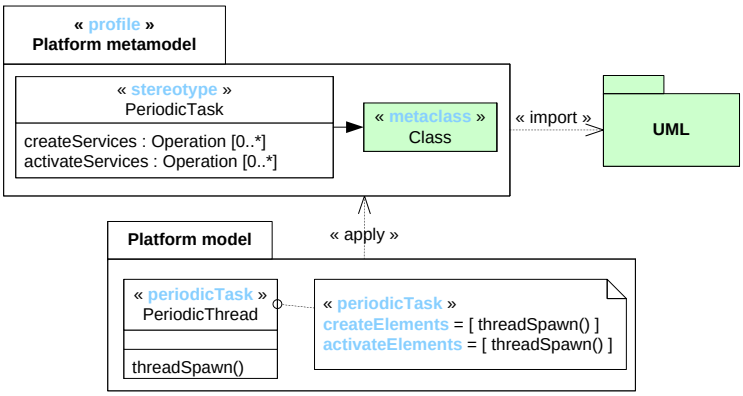

FIG. *4.11 – Illustration de la correspondance* $1 \mapsto M$ *pour les services*

Correspondance $N \mapsto 1$

La description de plusieurs services par une même propriété d'un stéréotype est intrinsèque au motif RESOURCE-SERVICE. La figure 4.12 page suivante illustre le cas du polymorphisme où deux services avec des signatures différentes permettent d'activer une tâche. Leurs sémantiques sont dénotées par un seul stéréotype.

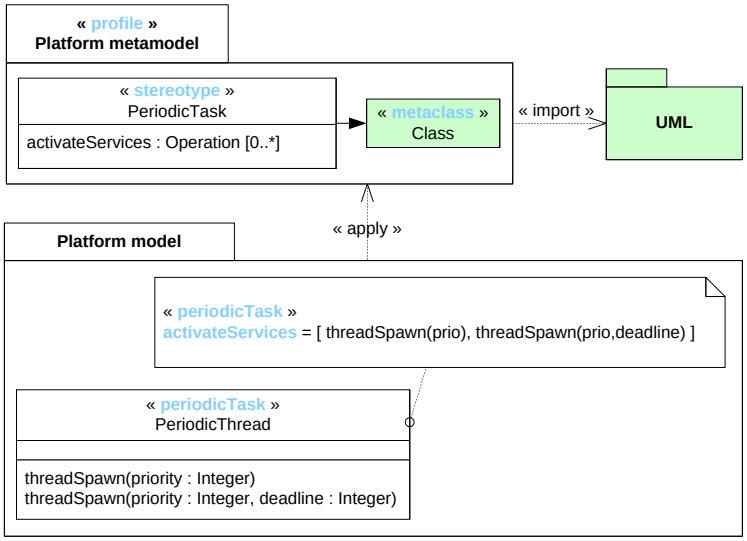

FIG. *4.12 – Illustration de la correspondance* $N \mapsto 1$ *pour les services*

Correspondance $N \mapsto M$

Cette correspondance est une composition des cas $N \mapsto 1$ et $1 \mapsto N$. Elle se traduit par des référencements multiples et par des références multiples.

Correspondance $0 \mapsto M$

Les activités UML permettent de décrire des flots d'action, de flots de contrôle et des flots de donnée. Par exemple, dans la figure 4.13 page suivante , le service chain permet d'activer un PeriodicThread précisé en paramètre puis de suspendre la tâche appelante. Cette sémantique n'est pas nativement supportée par la plate-forme. Une activité est alors utilisée pour décrire une implantation de cette sémantique. Cette activité est abstraite puisque son implantation n'existe pas réellement dans la plate-forme. Malheureusement, l'implantation du motif RESOURCE-SERVICE ne permet pas de référencer directement des activités UML mais essentiellement des opérations. L'utilisateur doit donc modéliser une opération virtuelle dans la ressource décrite puis référencer le corps de cette opération par l'activité citée précédemment. La figure 4.13 page suivante illustre une telle modélisation.

Correspondance $N \mapsto 0$

La solution à ce cas de figure rejoint le cas précédent en ce sens que des Activiy UML peuvent être utilisées pour définir cette sémantique. Elle est alors décrite par appelle aux services identifiés par le métamodèle. L'identification de services ne correspondant à aucun élément du métamodèle de plate-forme est alors possible si et seulement elle est décrite par des appels successifs à des services référencés. La figure 4.14 page suivante

illustre l'exemple du service chainTask présent dans la plate-forme mais absent du métamodèle.

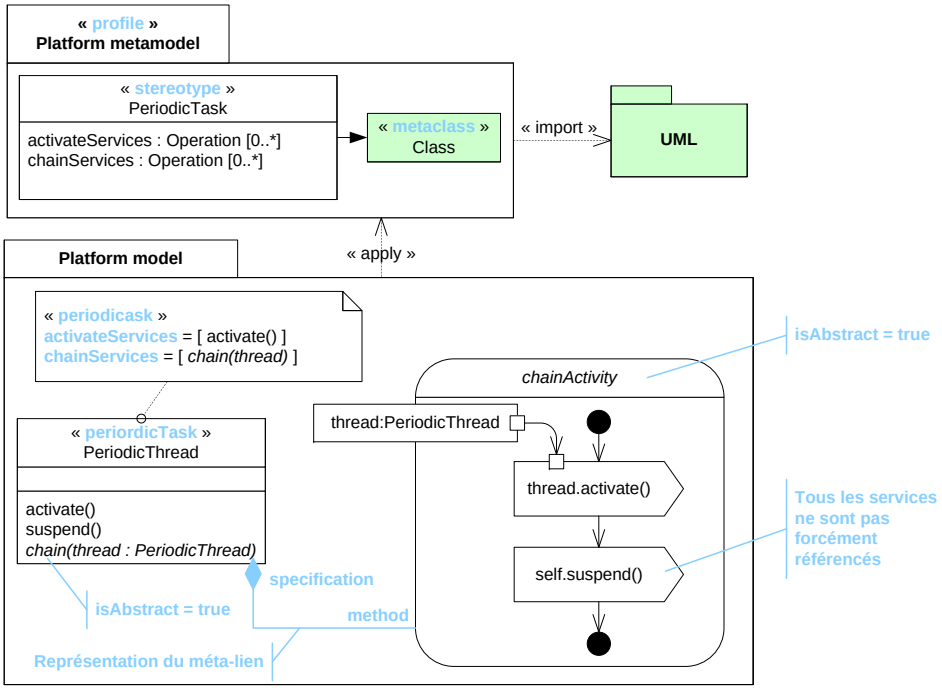

FIG. 4.13 – *Illustration de la correspondance* 0 $\mapsto$ *M pour les services*

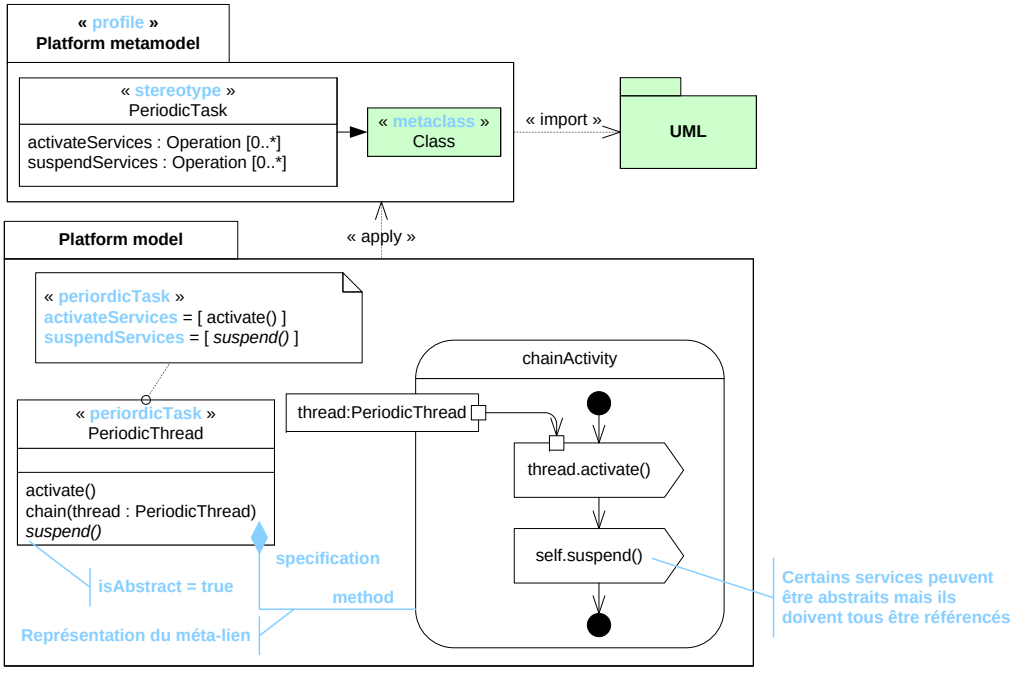

FIG. 4.14 – *Illustration de la correspondance N* $\mapsto$ 0 *pour les services*

4.2.4 Synthèse des heuristiques de modélisation

Le tableau 4.2.4 synthétise les heuristiques de modélisation identifiées.

TAB. *4.1 – Synthèse des heuristiques de modélisation pour chaque correspondance Plate – forme ↦ pivot*

Correspondance		Resource	Éléments typés (Propriétés et paramètres)	Services
$1 \rightarrowtail 1$	$1 \mapsto 1$	Application d'un stéréotype	Une référence	Une référence
$N^1 \rightarrowtail M^2$	$1 \mapsto M$	Stéréotypes multiples	Référencements multiples[3]	Référencements multiples[3]
	$N \mapsto 1$	Application multiples[4]	Références multiples[5]	Références multiples[5]
	$N \mapsto M$	Stéréotypes et applications multiples	Référencements et références multiples	Référencements et références multiples
$0 \rightarrowtail M^6$	$0 \mapsto 1$	Collaboration où classe stéréotypées	Expression	Activity
	$0 \mapsto M$	Collaboration ou classe stéréotypées plusieurs fois	Expression référencée plusieurs fois	Activity référencées plusieurs fois
$N^7 \rightarrowtail 0$	$N \mapsto 0$	Classes structurées référant des ressources stéréotypées. Ces références sont privées.	Expression référant des éléments typés référencés	Activity référant des services référencés.

[1] N un entier tel que $N \geq 2$,
[2] M un entier tel que $M \geq 2$,
[3] Un élément du modèle est référencé par plusieurs propriétés du stéréotype,
[4] Plusieurs stéréotypes sont appliqués sur le même élément,
[5] Une propriété du stéréotype référence plusieurs éléments du modèle,
[6] M un entier tel que $M \geq 1$,
[7] N un entier tel que $N \geq 1$,

4.3 DISCUSSIONS

Adéquation du motif ?

Le grand degré de liberté de modélisation fourni par le motif est une force face à l'hétérogénéité des plates-formes à modéliser. Cette liberté de modélisation permet une grande flexibilité d'utilisation, flexibilité qui doit

être limitée par des contraintes (OCL) dans des méthodologies outillées. Même si le couple (UML, Resource-Service) a permis d'identifier des solutions de modélisation pour chacun des cas de modélisation, les cas où la plate-forme ne propose pas nativement de concept ($0 \mapsto M$) et ou le métamodèle ne permet pas de modéliser certains concepts de la plate-forme ($N \mapsto 0$) sont discutables.

Le cas $0 \mapsto M$ est-il légitime ?

Le cas $0 \mapsto M$ est surprenant en première lecture car il illustre la contradiction de vouloir modéliser quelque chose qui n'existe pas dans la plate-forme. Pourtant, il est légitime lorsque les modèles de plates-formes sont utilisés dans des méthodologies outillées. En effet, ces modèles de plates-formes sont supposés être utilisés en entrée de transformations de modèles qui produisent un modèle spécifique à une plate-forme cible à partir d'un modèle spécifique à une plate-forme source. Dans les cas où la plate-forme source propose un ensemble de concepts que la plate-forme cible ne propose pas, il est nécessaire de décrire une implantation du mécanisme manquant sur la cible. C'est donc le cas ou il faut modéliser un mécanisme inexistant nativement (0) par un ou plusieurs éléments du métamodèle (M). Il est légitime de décrire cette implantation par le métamodèle de plate-forme puisque les transformations sont écrites vis à vis de ce métamodèle. Il est vrai que dans un tel cas de figure, le modèle de la plate-forme est pollué par des détails utiles dans un contexte de transformation donné. Pour pouvoir réutiliser et capitaliser les modèles de plates-formes, il est donc préconisé de séparer la modélisation de la plate-forme et la modélisation des motifs d'implantation. Comme le montre la figure 4.15 , les méthodologies de modélisation des plates-formes peuvent imposer la description de deux modèles. Le premier regroupe les concepts intrinsèques à la plate-forme (Platform Core) et le second les motifs de conceptions associés à cette plate-forme (Platform Pattern).

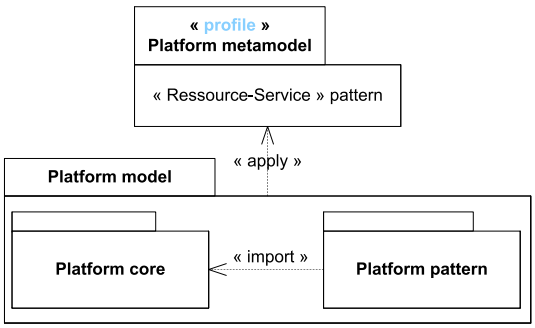

Fig. 4.15 – *Structuration préconisée des modèles de plate-forme*

Le cas $N \mapsto 0$ est-il légitime ?

Le cas ($N \mapsto 0$) est beaucoup plus litigieux. Une première approche, simple, est de considérer que c'est une faiblesse du métamodèle. Ce cas illustre en effet le fait que le métamodèle ne possède aucun artéfact pour décrire certains éléments de la plate-forme. L'extension du métamodèle permet alors de ce ramener au cas $1 \mapsto 1$, $N \mapsto 1$, $1 \mapsto M$ ou $N \mapsto M$. Dans une deuxième approche, ce cas implique la description des éléments de la plate-forme par des éléments primitifs de la plate-forme qui sont décrits par le métamodèle. Cette description n'est peut être pas repré-

sentative de l'implantation. Elle peut être utilisée dans des phases d'optimisation des transformations et des générations. Cette approche reste néanmoins très exploratoire. Elle fait appel à des techniques de l'intelligence artificielle et plus particulièrement du domaine de la configuration [68]. Ce cas de figure soulève le problème de la définition d'une ontologie minimaliste ou d'une ontologie complète d'un domaine. Dans le cas des plates-formes d'exécution, l'hétérogénéité réfute l'hypothèse d'une ontologie complète. Elle favorise la description d'une ontologie (minimaliste ou non) et d'un ensemble d'artéfacts de modélisation pour palier aux différents cas de modélisation.

Conclusion

Dans ce chapitre, l'objectif était d'étudier les heuristiques de modélisation associées au motif Resource-Service. Il apparaît que l'implantation UML de ce motif répond à tous les cas de figure rencontrés durant la modélisation des plates-formes. Il a été montré par des expérimentations que le couple (UML, motif) permet de répondre à chacun des besoins de modélisation. Ces expérimentations identifient un ensemble d'heuristiques de modélisation des plates-formes. Ces heuristiques constitueront le cœur de méthodologies de modélisation des plates-formes. La méthodologie Accord|UML [40] pourrait par exemple être enrichie par ces heuristiques pour intégrer explicitement les plates-formes d'exécution dans le cycle de développement.

Ainsi, après avoir défini et validé expérimentalement le motif Resource-service, il faut désormais s'assurer que ces méthodologies peuvent être outillées, c'est-à-dire que des architectures et des services de transformations peuvent être décrits. Le chapitre 5 page 75 vise à spécifier les services primitifs dédiés au motif Resource-Service.

5 Spécification d'un cadre technologique dédié au motif Resource-Service

Sommaire

Ce chapitre vise à décrire un cadre technologique pour le motif Resource-Service. Il vise à outiller le motif Resource-Service afin d'intégrer des modèles explicites de plates-formes d'exécution dans des infrastructures génératives, c'est-à-dire dans des transformations de modèles outillées.

Pour cela, le chapitre est divisé en deux étapes. Premièrement, l'intégration des modèles de plates-formes dans des processus de transformation est étudiée. Ces modèles de plates-formes peuvent être explicites ou implicites. Deuxièmement, ce chapitre spécifie les services utiles lors de la conception des transformations basées sur des métamodèles de plates-formes explicites. Ces spécifications sont spécifiques au motif Resource-Service. Elles sont formalisées sous la forme d'ensembles mathématiques et d'algorithmes. Elles pourront ainsi être utilisées pour différents outils dédiés à l'intégration explicite des plates-formes d'exécution.

5.1 Définition des architectures de transformations

Un des objectifs d'une ingénierie générative est d'automatiser la production des applications. Cette automatisation est constituée : a) d'un espace de problème, c'est-à-dire une application à implanter, b) d'un espace de solution, c'est-à-dire un ensemble des plates-formes cibles sur lesquelles l'application doit être implantée et, c) d'une base de connaissance de configuration, c'est-à-dire une base de connaissance pour décider et effectuer l'implantation [69]. Dans le cadre de cette étude, l'espace de problème est un ensemble de modèles applicatifs spécifiques à une plate-forme. L'espace de solution est l'ensemble des mécanismes et des services cibles avec lesquelles l'application doit désormais être implantée dans la plate-forme cible. L'espace de configuration est un ensemble des modèles de transformations. Ces transformations sont exogènes ou endogènes. Elles sont décrites manuellement ou semi-automatiquement.

5.1.1 Définition des architectures exogènes

Dans l'infrastructure de cette étude basée sur des métamodèles de plates-formes, les modèles de plates-formes, explicites ou implicites, paramètrent l'exécution des transformations. Ils apparaissent en tant que paramètres des modèles de transformation (M_τ). Comme le montre la figure 5.1 page suivante , le modèle M_τ importe les métamodèles utilisés pour décrire ces plates-formes. La transformation τ, issue de (M_τ), transforme une application spécifique à une ou plusieurs plates-formes sources en une application implantée sur une ou plusieurs plates-formes cibles.

Transformation d'ordre supérieur

Afin de mettre en évidence le schéma de transformation ***classique*** dans lequel les métamodèles de plates-formes ne sont pas explicites, il suffit de considérer que la transformation τ est obtenue par une transformation d'ordre supérieur , τ_{HOT} (HOT : High Order Transformation). Comme le montre la figure 5.2 page suivante , le modèle de cette transformation, $M_{\tau_{HOT}}$, importe : a) les métamodèles applicatifs, b) les métamodèles des plates-formes sources, c) les métamodèles des plates-formes cibles et d) le métamodèle du langage de transformation. A partir de ces métamodèles, la transformation τ_{HOT} génère le modèle de transformation M_τ.

Intégration des modèles de plates-formes implicites

Dans le cas ou les modèles de plates-formes sont implicites, c'est-à-dire promu au niveau méta, les transformations ne sont plus paramétrées par des modèles mais par des profils. Comme ces profils ne peuvent être décrits directement par des métamodèles de plate-forme, les infrastructures de transformation nécessitent d'expliciter la transformation de promotion τ_p. Celle-ci est utilisée pour promouvoir une description explicite en une description implicite. Elle est illustrée dans la figure 5.3 page 79 . Même si ce n'est pas représenté dans la figure, là aussi une transformation d'ordre supérieur peut être utilisée.

Un cas particulier de ces architectures est celui des architectures endogènes dans lesquelles les métamodèles sources et cibles sont identiques.

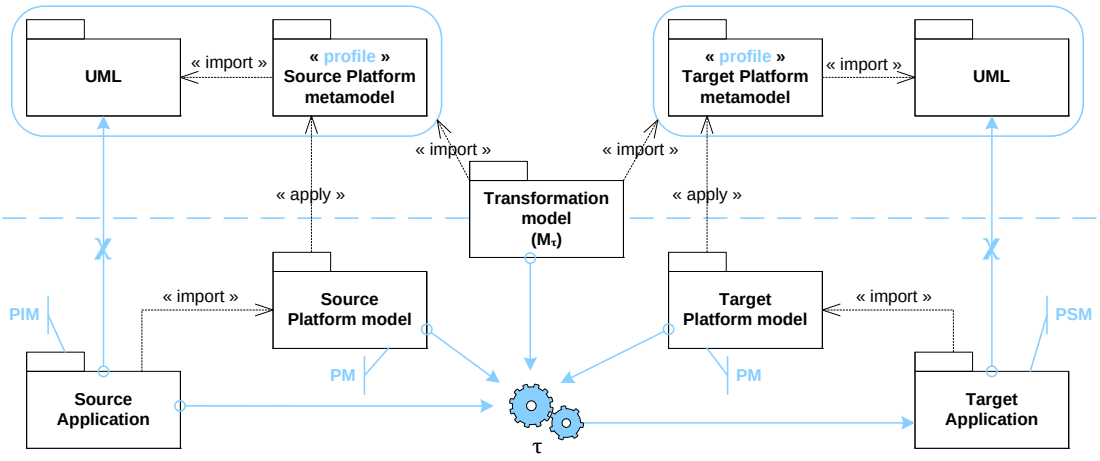

FIG. 5.1 – *Infrastructure de transformations exogènes basées sur des modèles de plates-formes explicites*

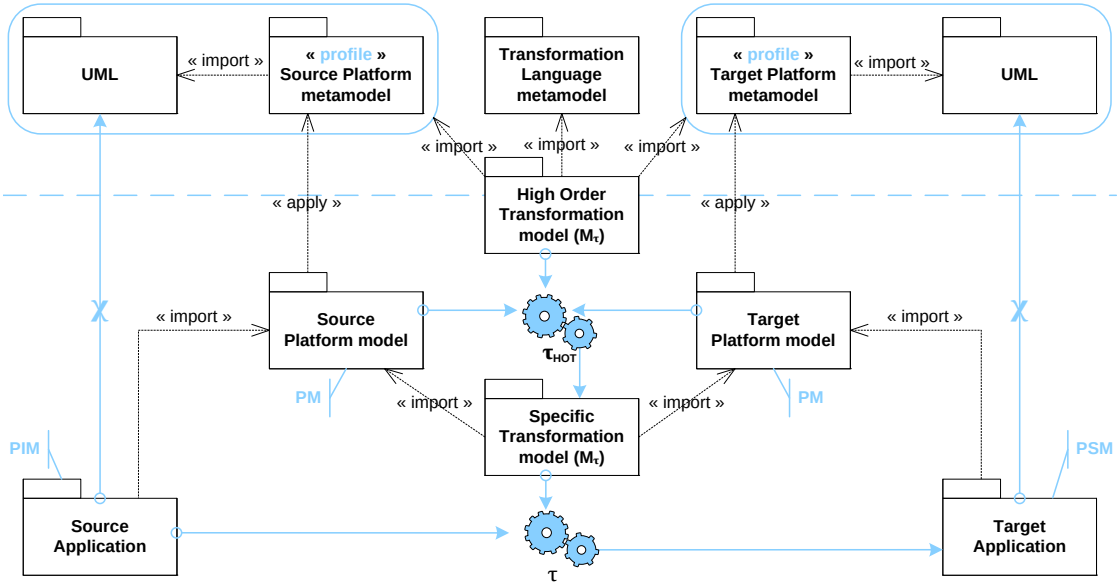

FIG. 5.2 – *Infrastructure intégrant des transformations d'ordre supérieur*

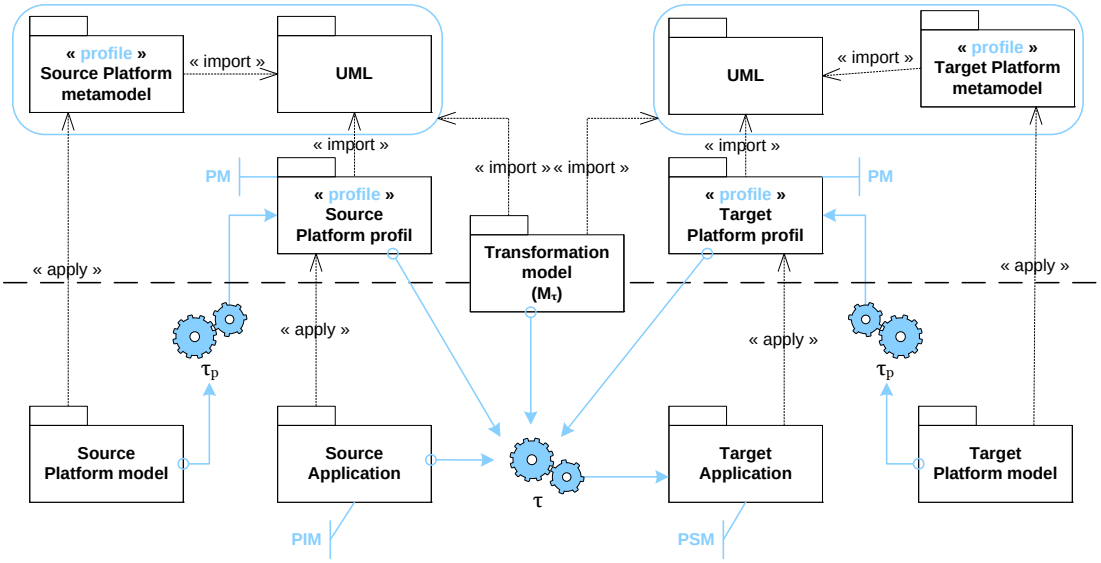

FIG. 5.3 – *Infrastructure de transformations exogènes basées sur des modèles de plates-formes implicites*

5.1.2 Définition des architectures endogènes

Un intérêt des métamodèles de plate-forme est la représentation d'une ontologie d'un domaine d'exécution, c'est-à-dire d'une base conceptuelle commune à différents supports d'exécution. Le métamodèle de plate-forme est alors utilisé comme un pivot pour porter les applications d'un ensemble de plates-formes sources vers un ensemble de plates-formes cibles. Le portage d'une application spécifique à un exécutif source vers un exécutif cible est un exemple d'utilisation d'un métamodèle pivot permettant de décrire ces plates-formes sources et cibles. La figure 5.4 page suivante illustre la mise en œuvre d'un métamodèle de plate-forme dans une architecture endogène. Dans cette figure, l'ensemble des plates-formes sources et cibles est décrit par le même ensemble de métamodèle. Certaines de ces plates-formes sont explicites, d'autres implicites.

Ces architectures de transformation peuvent être d'ores et déjà implantées dans des outils de transformations. Pour faciliter cette implantation, la section suivante spécifie un ensemble de services pour manipuler les modèles de plates-formes.

5.2 SPÉCIFICATION DES SERVICES DE TRANSFORMATION

Il existe deux grands ensembles de services : 1) les services liés aux architectures exogènes et 2) les services liés aux architectures endogènes. Ces deux ensembles ne sont pas disjoints puisque les services exogènes peuvent être utilisés dans des architectures endogènes.

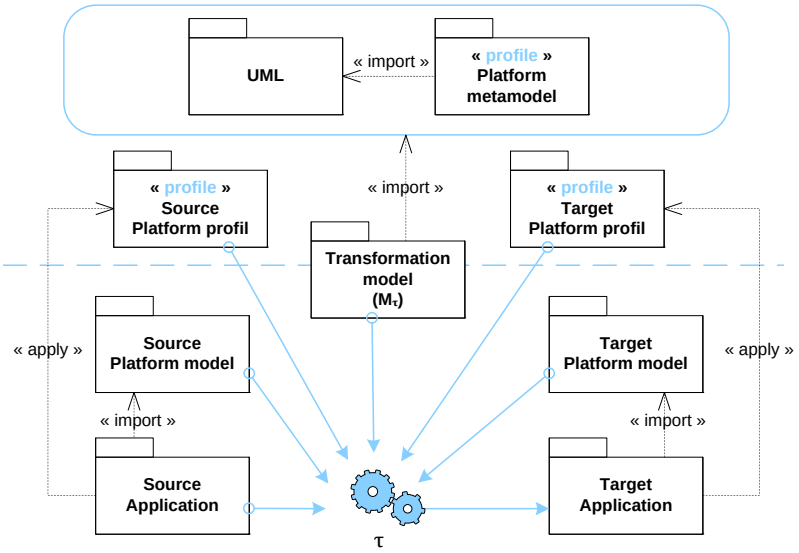

FIG. 5.4 – *Infrastructure de transformations endogènes basées sur des modèles de plates-formes*

5.2.1 Spécification des services exogènes

Les besoins liés aux transformations exogènes basés sur des métamodèles appliquant le motif comprennent :

1) l'obtention pour une ressource donnée de sa description, c'est-à-dire des stéréotypes et des valeurs de ces stéréotypes,
2) l'obtention des ressources du modèle de la plate-forme décrit par un stéréotype donné et
3) l'obtention des propriétés, des opérations et des paramètres du modèle de la plate-forme, référencés par une propriété d'un stéréotype donné.

La satisfaction de ces besoins passe par la formalisation des modèles et des métamodèles de plates-formes puis par la définition des ensembles retournés par ces services d'une part dans le contexte des ressources puis dans le contexte des stéréotypes.

Spécification des modèles et des métamodèles de plates-formes

Métamodèle de plate-forme

Soit $P_{resource-service}$ l'ensemble des profils appliquant le motif RESOURCE-SERVICE et χ la relation de conformité entre un élément et un métaélément (relation décrite dans la section 1.2.1 page 14), alors l'ensemble des métamodèles de plates-formes $P_{plate-forme}$, c'est-à-dire des profils UML dédiés à la description des modèles de plates-formes, est défini par

$$P_{plate-forme} = \{p_{plate-forme} \mid p_{plate-forme}\ \chi\ \text{Profile} \wedge p_{plate-forme} \in P_{resource-service}\} \quad (5.1)$$

Modèle explicite de plate-forme

Soit $P_{applied}(m)$ l'ensemble des profils appliqués sur le modèle m, alors

l'ensemble des modèles de plate-forme explicites M_{exp} est défini par

$$M_{exp} = \{m_{exp} \mid m_{exp}\ \chi\ \mathsf{Model} \wedge P_{applied}(m_{exp})\ \subset\ P_{plate-forme}\} \tag{5.2}$$

Modèle implicite de plate-forme

Soit τ_P une fonction de promotion d'un modèle UML en un profil UML tel que $\tau_P : M \rightarrow P$ avec $M = \{m|m\ \chi\ \mathsf{Model}\}$ et $P = \{p|p\ \chi\ \mathsf{Profile}\}$ alors, l'ensemble des modèles de plate-forme implicites M_{imp} est défini par

$$M_{imp} = \{m_{imp} \mid m_{imp}\ \chi\ \mathsf{Profile} \wedge P_{applied}(\tau^{-1}(m_{imp}))\ \subset\ P_{plate-forme}\} \tag{5.3}$$

Modèle de plate-forme

L'ensemble des modèles de plates-formes dans cette étude est l'ensemble $M_p = M_{exp} \cup M_{imp}$.

A partir de ces ensembles, il est possible de spécifier, d'une part, les ensembles manipulés dans le contexte d'une ressource et, d'autre part, les ensembles manipulés dans le contexte d'un stéréotype.

Spécification des services dans le contexte d'une ressource

Dans le contexte d'une ressource, les ensembles nécessaires sont les stéréotypes appliqués et les valeurs des propriétés de ces stéréotypes.

Stéréotype appliqué sur une ressource

Soit $S(e)$ l'ensemble des stéréotypes appliqués sur un élément UML e et soit $\tau_{p_r} : r_{exp} \rightarrow r_{imp}$ la fonction de promotion d'une ressource explicite r_{exp} en une ressource implicite r_{imp} (d'un élément en un métaélément), l'ensemble des stéréotypes appliqués sur une ressource r est défini par :

$$\begin{aligned} S_r(r) = \{s \mid \exists p_{plate-forme} \in P_{plate-forme} :\ & s \in p_{plate-forme}.\mathsf{ownedStereotype} \wedge \\ & s\ \chi\ \mathsf{Stereotype} \wedge \\ & s \in S(r) \vee \\ & s \in S(\tau_{p_r}^{-1}(r))\} \end{aligned} \tag{5.4}$$

De même soit $V(e, s, p_s)$ l'ensemble des valeurs d'une propriété p_s d'un stéréotype s appliqué sur un élément UML e, l'ensemble des valeurs pour une propriété p_s d'un stéréotype s appliqué sur la ressource r est défini par :

$$V_{tag}(r, s, p_s) = \{v \mid v \in V(r, s, p_s) \vee v \in V(\tau_{r_p}^{-1}(r), s, p_s)\} \tag{5.5}$$

Propriétés référencées

L'obtention des propriétés référencées par p_s est alors définie par :

$$P_{ref}(r, s, p_s) = \{p_{ref} \mid p_{ref} \in V_{tag}(r, s, p_s) \wedge p_{ref}\ \chi\ \mathsf{Property}\} \tag{5.6}$$

Services référencés

De même l'obtention des opérations référencées par p_s est définie par :

$$O_{ref}(r,s,p_s) = \{o_{ref} \mid o_{ref} \in V_{tag}(r,s,p_s) \wedge o_{ref}\ \chi\ \mathsf{Operation}\} \tag{5.7}$$

Paramètres référencés

Enfin, l'ensemble des paramètres référencés par p_s et appartenant à une opération o non forcément référencée est défini par :

$$P_{paramRef}(r,s,p_s,o) = \{p_{paramRef} \mid p_{paramRef} \in V_{tag}(r,s,p_s) \wedge p_{paramRef}\ \chi\ \mathsf{Parameter} \wedge p_{paramRef} \in o.\mathsf{ownedParameter}\} \tag{5.8}$$

Spécification des services dans le contexte d'un stéréotype

Ressources explicites stéréotypées

Soit $E(m)$ l'ensemble[1] des éléments d'un modèle m et $S(e)$ l'ensemble des stéréotypes appliqués sur l'élément e, alors l'ensemble des ressources décrites par s et appartenant à un modèle de plate-forme explicites $m_{exp} \in M_{exp}$ est défini par :

$$R_{exp}(s) = \{r_{exp} \mid \exists m_{exp} \in M_{exp} : r_{exp} \in E(m_{exp}) \wedge r_{exp}\ \chi\ (\mathsf{Type}) \wedge s \in S(r_{exp})\} \tag{5.9}$$

Ressources implicites stéréotypées

De même, l'ensemble des ressources implicites décrites par s et appartenant à un modèle de plate-forme implicite $m_{imp} \in M_{imp}$ est défini par :

$$R_{imp}(s) = \{r_{imp} \mid \exists m_{imp} \in M_{imp} : r_{imp} \in E(m_{imp}) \wedge r_{imp}\ \chi\ \mathsf{Stereotype} \wedge s \in S(\tau_{p_r}^{-1})\} \tag{5.10}$$

A partir des spécifications 5.9 et 5.10, L'ensemble des ressources décrites par s est alors défini par :

$$R(s) = R_{exp}(s) \cup R_{imp}(s) \tag{5.11}$$

Cet ensemble $R(s)$ représente l'ensemble des ressources décrites par s que l'utilisateur peut manipuler pour décrire son modèle applicatif. Cet ensemble peut être constitué de ressources appartenant à des modèles de plates-formes différents.

Références d'un élément

Étant donné l'agencement du motif RESOURCE-SERVICE sous la forme de références, il est nécessaire de pouvoir obtenir l'ensemble des références pour un élément typé ou une opération du modèle. Pour cela, l'algorithme 1 page suivante spécifie pour un élément e comment obtenir l'ensemble de ces références.

[1] L'obtention de l'ensemble $E(m)$ consiste à supprimer les relations de composition pour faire disparaitre les noeuds du modèle.

Algorithme 5.1 : Algorithme d'obtention des références pour un élément typé d'un modèle de plate-forme

Entrées :
e : Un élément susceptible d'être référencé,

R : L'ensemble des ressources des modèles de plate-forme (explicite et implicite)

Résultat :
Retourne l'ensemble des références Ref d'un élément e sous la forme d'un triplet comprenant la propriété p_s du stéréotype s appliqué sur la ressource r référençant e.
$Ref = \{ref \mid \exists\ p_s\ \chi\ \mathsf{Property},\ \exists\ s\ \chi\ \mathsf{Stereotype},\ \exists\ r \in R :\ ref = (\ p_s,\ s,\ r\ )\}$

Données :
$S_r(r)$: L'ensemble des stéréotypes appliqués sur r,

$T_{TypedElement}$: L'ensemble des éléments UML héritant de la métaclasse TypedElement

L'ensemble des propriétés du stéréotype s référençant des éléments du modèle $P_s(s) =$
$\{p_s | p_s \in s.\mathsf{property} \wedge p_s.\mathsf{type} \in T_{TypedElement} \vee p_s.\mathsf{type} = \mathsf{Operation}\}$

$V_{tag}(r, s, p_s)$: L'ensemble des valeurs de la propriété p_s du stéréotype s appliqué sur r

```
1  Initialisations :
2  Ref ← ∅;
3  début
4      pour chaque r ∈ R faire
5          pour chaque s ∈ S_r(r) faire
6              pour chaque p_s ∈ P_s(s) faire
7                  si v ∈ V_tag(r, s, p_s) alors
8                      Ref ← Ref ∪ (p_s, s, r);
9      retourner Ref;
10 fin
```

Dans cet algorithme, la prise en compte de l'ensemble des ressources du modèle est nécessaire puisque le motif Resource-Service n'impose pas que l'élément référencé appartienne à l'élément stéréotypé. Un élément peut être référencé par n'importe quels stéréotypes. Cet algorithme est terminal puisque tous les ensembles manipulés sont des ensembles finis et les boucles sont exemptes d'appels récursifs. En ce qui concerne la complexité, elle est dépendante des cardinalités des ensembles R, $S_r(r)$, $P_s(s)$ et $V_{tag}(r, s, p_s)$. Il est raisonnable de postuler que le nombre de stéréotype appliqué sur une ressource est une constante[2] et que le nombre des propriétés d'un stéréotype est aussi une constante[3] vis à vis du nombre de ressources manipulées. Ainsi la complexité au pire cas est en $O(n)$ avec $n \in \mathbb{N}$ le nombre maximum de ressource manipulée.

Ces équations et cet algorithme spécifient les ensembles que doivent retourner des services dédiés à la manipulation de modèle. Ces services sont utilisables dans les architectures exogènes illustrées dans les figures 5.1 page 78 , 5.3 page 79 et 5.2 page 78 ; elles peuvent être aussi utilisées dans des architectures endogènes.

5.2.2 Spécification des services endogènes

Équivalence

Dans des infrastructures endogènes, l'ensemble des métamodèles sources et cibles sont identiques. Par conséquent, il est possible d'écrire un algorithme d'équivalence ($\approx$) entre des ressources sources et des ressources cibles. Cet algorithme est spécifié en 5.2 page suivante. La description 2 page ci-contre compare, d'une part, les stéréotypes appliqués sur les ressources passées en paramètre et ,d'autre part , les valeurs des métapropriétés primitives, c'est-à-dire des types primitifs (booléen, entier et chaîne de caractère par exemple) et des énumérations. Deux ressources sont ainsi équivalentes si et seulement si les ensembles des stéréotypes appliqués sur la source et sur la cible sont égaux et si ces stéréotypes sont renseignés de la même façon.

La preuve de terminaison de l'algorithme 2 page suivante est triviale puisque touts les ensembles manipulés sont finis et il n'y a pas d'appels récursifs. En ce qui concerne la complexité, elle est dépendante des cardinaux des ensembles $S_r(r_s)$ et $P_s(s_s)$. Ainsi, la complexité est $O(n)$ avec $n \in \mathbb{N}$ le nombre maximal de propriété des stéréotypes puisque le nombre de stéréotype appliqué sur la source et la cible est négligeable vis à vis du nombre de propriété.

[2]Heuristique issue des expérimentations de la partie III page 91.
[3]Heuristique issue des expérimentations de la partie III page 91.

Algorithme 5.2 : Algorithme de l'opérateur d'équivalence $\approx$ dédié au motif RESOURCE-SERVICE

Entrées :
r_s : Ressource source,

r_c : Ressource cible

Résultat :
Renvoie Vrai si $r_s \approx r_c$, Faux sinon

Données :
$S_r(r_s)$: L'ensemble, sans doublon, des stéréotypes appliqués sur la ressource source r_s,

$S_r(r_c)$: L'ensemble, sans doublon, des stéréotypes appliqués sur la ressource cible r_c,

```
1  début
2      si S_r(r_s) = ∅ ∧ S_r(r_c) = ∅ ∧ | S_r(r_s) | ≠ | S_r(r_c) | alors
3          retourner Faux;
4      sinon
5          pour chaque s_s ∈ S_r(r_s) faire
6              si s_s ∈ S_r(r_c) alors
                   L'ensemble des propriétés du stéréotype s_s ne référençant
                   pas d'éléments du modèle
7                  P_s(s_s) =
                   {p_s | p_s ∈ s_s.property ∧ p_s.type χ PrimitiveType ∨ p_s.type χ Enumeration}
                   pour chaque p_s ∈ P_s(s_s) faire
8                      si V_tags(r_s, s_s, p_s) ≠ V_tagc(r_c, s_s, p_s) alors
                           Les valeurs primitives (booléenne,entière, chaîne de
                           caractère ou énumérées sources et cibles ne sont pas
                           identiques, les concepts ne sont donc pas
                           équivalents
9                          retourner Faux;
10             sinon
                   Le stéréotype source n'est pas appliqué sur la cible
11                 retourner Faux;
           Les mêmes stéréotypes sont appliqués et leurs propriétés primitives
           et énumérées sont renseignées par les mêmes valeurs, donc les
           concepts sont équivalents
12         retourner Vrai;
13 fin
```

5.3 Discussions

Est-ce que l'explicitation des plates-formes apporte la flexibilité ?

L'explicitation des plates-formes dans les transformations exogènes apporte aux ingénieries génératives un degré de flexibilité supplémentaire par rapport aux ingénieries dirigées par les modèles classiques. Pour une même description de transformation M_τ et pour un même modèle applicatif source (PIM), plusieurs modèles applicatifs cibles (PSM) peuvent être générés. Les modèles de plates-formes paramètrent les exécutions des transformations et donc capitalisent les descriptions de ces transformations. Cet apport n'est pas dû à la métamodèlisation des plates-formes en soi mais à l'architecture en couche de l'IDM et au couple (modèle, métamodèle).

La promotion inverse est-elle difficile à implanter ?

Dans les équations 5.3, 5.4, 5.5 et 5.10, le résultat de la transformation de promotion inverse τ_p^{-1} est utilisé. Une solution simple d'obtention de ce résultat est de tracer la transformation de promotion τ_p. Cette traçabilité peut être implicite en effectuant une équivalence syntaxique sur les noms ou explicite en générant un modèle de traçabilité référençant la source et la cible en sortie de la transformation de promotion. Ce modèle de traçabilité permet ainsi de facilement retrouver l'ensemble des ressources d'un modèle de plate-forme explicite qui a été promu et qui est utilisé comme un modèle de plate-forme implicite.

Est-ce que les services spécifiés sont génériques ?

Toutes les spécifications de ce chapitre sont basées sur la première équation formalisant ce qu'est un métamodèle de plate-forme $P_{plate-forme}$, c'est-à-dire un métamodèle appliquant le motif Resource-Service. Par la suite les spécifications ne sont pas dépendantes d'un métamodèle de plate-forme particulier. Ainsi ***ces spécifications sont valables et utilisables pour n'importe quel métamodèle de plate-forme appliquant le motif***. Les ingénieries résultantes acquièrent un haut degré de généricité. Les algorithmes d'équivalence et de recherche des ressources cibles dans une architecture de transformation endogène sont par exemple génériques.

Conclusion

Le premier objectif de ce chapitre était de définir les architectures de transformations basées sur des modèles de plates-formes implicites et explicites. Le second objectif était de spécifier les services dédiés au motif Resource-Service et utiles à la manipulation des modèles de plates-formes conforment à des métamodèles de plates-formes appliquant le motif.

Dans un premier temps les différentes architectures ont été identifiées. Dans un deuxième temps les services a) d'obtention pour une ressource donnée de sa description, b) d'obtention des ressources du modèle de la plate-forme décrit par un stéréotype donné, c) d'obtention des propriétés, des opérations et des paramètres référencés par une propriété d'un stéréotype donné et d) la recherche d'un ensemble de ressources cibles équivalentes à un ensemble de ressources sources, ont été décrits. Ces ser-

vices peuvent être implantés en JAVA et/ou en OCL par exemple. Dans le cadre de cette étude, ils font l'objet d'une librairie de transformation ATL[4].

A ce stade de l'étude, le motif RESOURCE-SERVICE a été défini (chapitre 3), un cadre méthodologique a été identifié (chapitre 4) et la spécification d'un cadre technologique est défini dans ce chapitre. Le motif est donc opérationnel pour être évalué dans une ingénierie générative expérimentale dirigée par les modèles. Cette ingénierie comprend un métamodèle de plate-forme logicielle d'exécution multitâche et une architecture de transformation endogène. Elle est détaillée dans la partie III page 91.

[4]L'initiative ATL implante une partie de la norme OCL [70] pour manipuler les éléments des modèles. Les limitations de cette implantation sont résumées dans le document [71].

Troisième partie

Contributions appliquées aux ingénieries génératives intégrant des modèles explicites de plates-formes d'exécution multitâche

6 Métamodèlisation des plates-formes logicielles d'exécution multitâche

Sommaire

Ce chapitre a deux objectifs. Le premier vise à utiliser le patron Resource-Service pour la conception du profil UML Software Resource Modeling (SRM). Celui-ci doit permettre la description de plates-formes logicielles d'exécution multitâche. Plus particulièrement, il doit décrire les agences définies au chapitre 1 page 7, c'est-à-dire les agences de concurrence, d'intéraction, d'ordonnancement et d'entrées/sorties des exécutifs. Le second objectif est d'évaluer le profil SRM pour la modélisation de plates-formes concrètes.

Pour cela, ce chapitre est décomposé en deux sections. Une première section s'intéresse à décrire l'ensemble des concepts constituant SRM, c'est-à-dire son modèle de domaine. Chaque concept est décrit et illustré succinctement. Dans la deuxième section, SRM est utilisé pour décrire des plates-formes. Les métriques de ces modélisations sont présentées et discutées afin d'évaluer l'adéquation du profil SRM pour la modélisation des exécutifs multitâches.

6.1 Définition d'un profil UML pour la modélisation des plates-formes logicielles d'exécution multitâche

Le modèle de domaine de SRM vise à décrire, de manière pragmatique, les principaux mécanismes et services représentatifs des exécutifs temps réel embarqués. Pour cela, six plates-formes représentatives du domaine ont été sélectionnées :

Définition des candidats étudiés

1) Osek/Vdx-OS [19] : norme dédiée à la spécification des exécutifs multitâches pour l'automobile,
2) POSIX IEEE std 1003.1 [72] : norme dédiée à la spécification des systèmes d'exploitation et plus spécifiquement à la spécification des systèmes multitâches,
3) VxWorks [21] : exécutif commercial largement utilisée dans les secteurs industriels de la télécommunication et de la robotique d'après des études de marchés récentes [73],
4) Rtai [74] : correctif pour utilisé des systèmes Linux dans des applications multitâches temps réel embarquées,
5) SCEPTRE 2 [75] : résultat d'un projet de recherche visant à unifier les interfaces de programmation des systèmes d'exploitation temps réel embarqués, et
6) Lacatre [76] : un langage, présentant un mode graphique et un mode textuel, destiné à faciliter la conception préliminaire et détaillée d'applications basées sur la mise en œuvre d'exécutifs multitâches temps réel.

Les concepts étudiés se concentrent sur la description de l'exécution concurrente, des interactions, de l'ordonnancement et des entrées/sorties. L'approche de conception de SRM consiste à comparer les propositions de chaque candidat puis à définir un concept commun. Cette ontologie est décrite dans le document [77]. Un complément est apporté dans l'annexe B page 147 pour les services. Ces deux documents cautionnent les choix de conception du modèle de domaine de SRM.

6.1.1 Synthèse du modèle de domaine

Le modèle de domaine de SRM est structuré en quatre packages. Le premier, ResourceCore, constitue le cœur du modèle. Il est le seul package à appliquer le motif Resource-Service. Les trois sous packages (Concurrency, Interaction, Brokering) réifient les concepts pour, respectivement, les exécutions concurrentes, les interactions entre les entités concurrentes et les gestionnaires des entités logicielles et matérielles. La figure 6.1 page suivante illustre cette structuration.

Définition du package ResourceCore

Ce package ResourceCore rassemble les concepts primitifs du domaine.

Resource et service logiciel

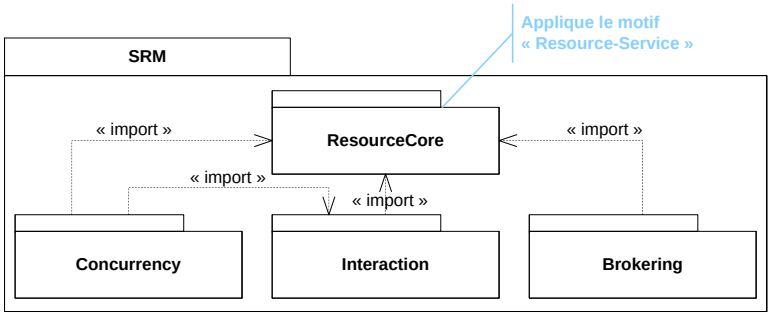

FIG. *6.1 – Structuration du modèle* SRM

Ces concepts sont décrits dans la figure 6.2 . Dans cette figure l'application du motif RESOURCE-SERVICE est représentée par un stéréotype. Le package applique le motif en décrivant les concepts de ressource logicielle et de service. Les ressources sont identifiées (identifierElements). Elles fournissent des services de création (createServices), de suppression (deleteServices) et d'initialisation (initializeServices).

Les plates-formes d'exécution étudiées sont des plates-formes programmées. Parmi les services, certains permettent d'accéder aux propriétés des ressources. En appliquant le motif, ces services sont modélisés comme des références. Néanmoins, pour ne pas surcharger le modèle de domaine et pour que ce modèle soit plus facilement utilisable, le concept de service est réifié par un concept d'accesseur : SwAccessService. Un accesseur est donc un service qui peut lire ou écrire (isModifier à vrai) une ou plusieurs propriétés.

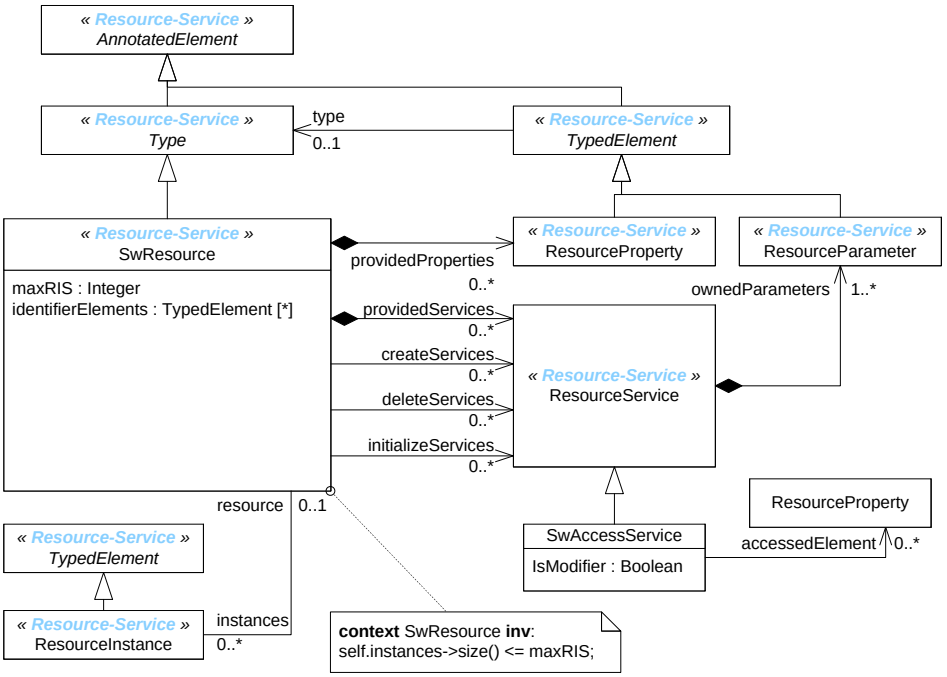

FIG. *6.2 – Structuration du package ResourceCore*

Définition du package Concurrency

Le package Concurrency regroupe l'ensemble des entités fournissant un ou plusieurs contextes pour exécuter des séquences d'actions en pseudo parallèle, c'est-à-dire en concurrence. Ces contextes sont des ConcurrentResource. Une séquence d'actions est alors identifiée par un point d'entrée (EntryPoint).

Resource d'exécution logicielle

Une ressource d'exécution concurrente permet d'exécuter les actions de son point d'entrée périodiquement, apériodiquement ou sporadiquement (type). Elle est caractérisée par des éléments permettant d'exprimer la priorité d'exécution (priorityElements), la période (periodElements) pour des exécutions périodiques ou enfin la taille de pile (stackesizeElements). Elle offre des services pour activer (activateServices), suspendre (suspendServices), reprendre (resumeServices) et terminer (terminateServices) une exécution. Certains services permettent d'autoriser (ou non) la préemption de cette exécution (enableConcurrencyServices, disableConcurrencyServices). La figure 6.3 illustre cette ressource. Dans cette figure et dans les figures suivantes, les concepts en couleur sont importés d'un autre package du modèle de domaine.

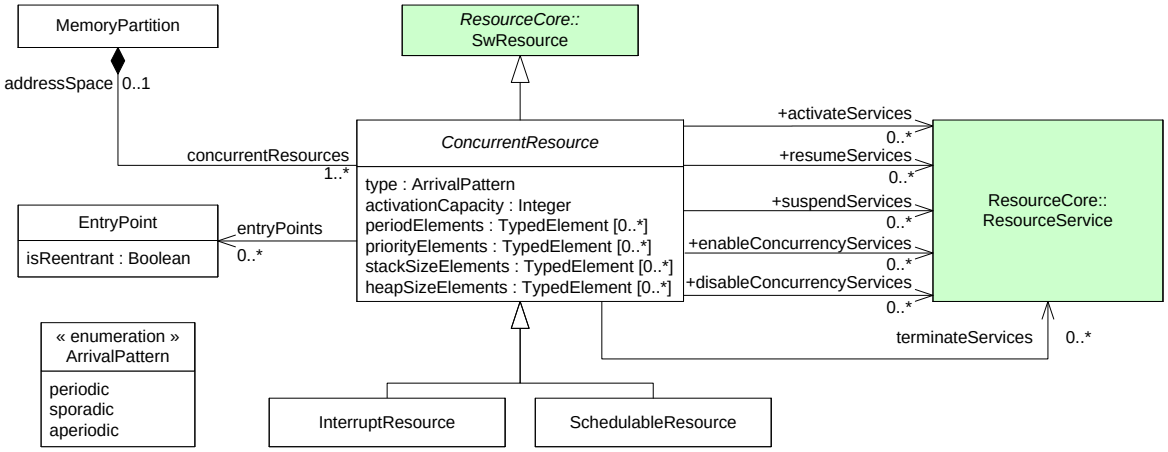

FIG. *6.3 – Hiérarchisation des ressources d'exécutions concurrentes*

Comme le montre la figure 6.4 page suivante , une ressource d'exécution interagit avec le reste du système par l'intermédiaire de ressources d'interaction qui sont, soit dédiées à la communication de données (SharedDataComRessource, MessageComRessource) ou, soit dédiées à la synchronisation des exécutions (MutualExclusionResource, NotificationResource) (voir la sous-section suivante). Les exécutions sont restreintes ou non à une partition mémoire MemoryPartition (voir figure 6.5 page suivante), c'est-à-dire un espace d'adressage virtuel qui confine les accès mémoire des ressources d'exécution. Le contexte d'exécution d'une partition parente peut être une copie d'une partition parente lors de la création de la partition (forkServices) Ce contexte d'exécution peut être libéré sans forcément être détruit (exitServices).

Partition mémoire

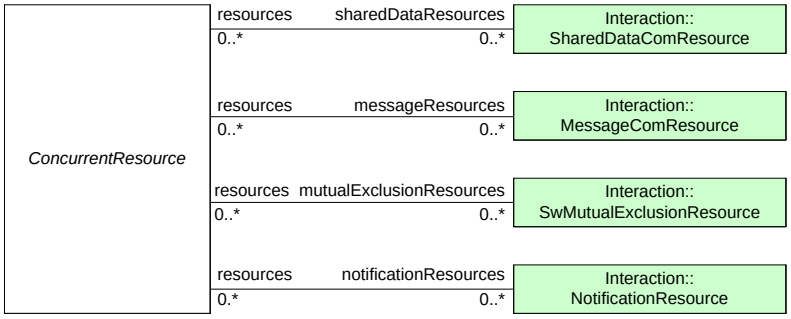

FIG. *6.4 – Description des interactions entre ressources concurrentes*

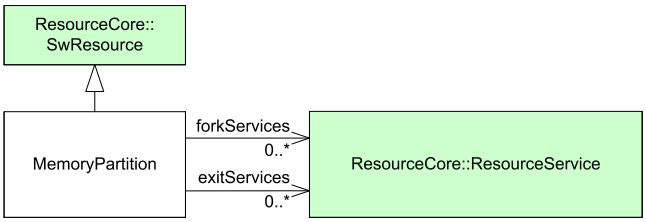

FIG. *6.5 – Description des partitions mémoires*

Concrètement, les ressources d'exécution concurrente s'ordonnent en deux grandes familles : les ressources d'interruption et les ressources ordonnançables. Ces familles se différencient par rapport à l'orchestration des exécutions dans le système. Ainsi, les exécutions des ressources d'interruption sont orchestrées par la plate-forme matérielle sous-jacente, c'est-à-dire par un contrôleur matériel (souvent le contrôleur programmable d'interruption PIC).

Resource d'interruption

Les ressources d'interruption (InterruptResource) proposent un contexte pour exécuter des instructions spécifiées dans une routine d'interruption (isrEntryPoint). L'exécution de cette routine est assujettie à l'occurrence d'un événement interrompant matériellement l'exécution du processeur. Cet événement peut être un matériel (HardwareInterrupt) ou logiciel. C'est alors une exception logicielle émise par le processeur lui même (ProcessorDetectedException) lorsqu'il détecte une condition anormale pendant l'exécution d'une instruction ou de manière programmée par l'utilisateur (ProgrammedException)[1].

Qu'elle soit logicielle ou matérielle, une ressource d'interruption est caractérisée par un vecteur (vectorElements) et un masque d'interruption (maskElements). Des services permettent de connecter

[1] Afin de mettre en oeuvre les mécanismes de protection nécessaires pour un système d'exploitation multitâches, le processeur possède deux modes de fonctionnement : utilisateur/superviseur. Ces deux modes de fonctionnement impliquent deux catégories de registres : les registres non-protégés (le compteur programme par exemple) et protégée (Le registre d'état par exemple). L'accès au registre protégé ne peut se faire que par des instructions privilégiées. Les appels à ces instructions imposent que le processeur passe en mode superviseur. La levée d'une exception par l'utilisateur est une façon de faire.

(routineConnectServices) ou de déconnecter (routineDisconnectServices) la routine de ce vecteur.

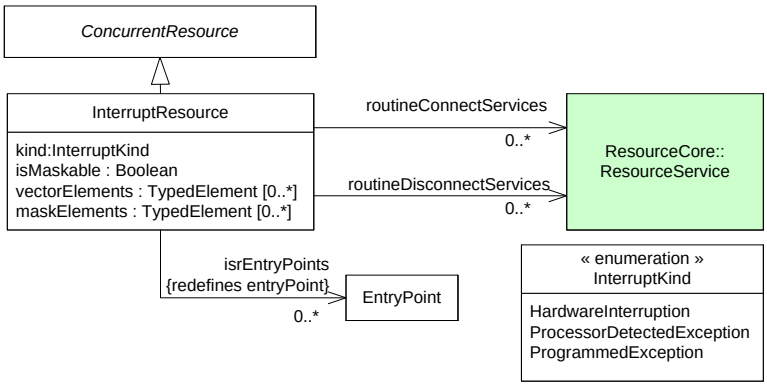

FIG. *6.6 – Description des ressources d'interruption*

Alarme, chien de garde et réveil

Une ressource d'interruption spécifique est l'alarme (Alarm) qui est connectée à un réveil (TimerResource TimerResource). Ce réveil est périodique ou non. Il propose des services pour le démarrer (startServices), l'arrêter (stopServices) ou le réinitialiser (resetServices). Ce couple alarme-réveil peut être spécialisé en un mécanisme de chien de garde (Watchdog)[2].

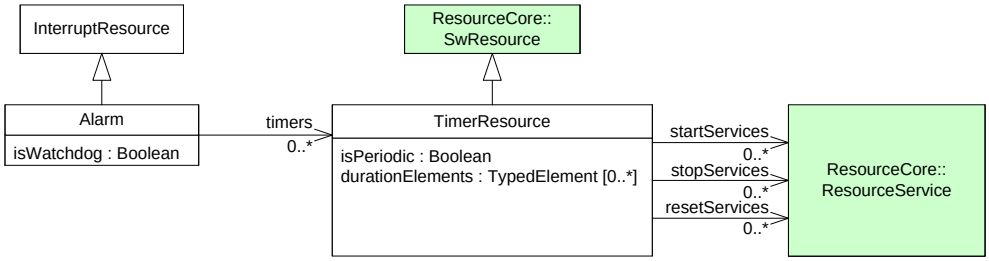

FIG. *6.7 – Description des alarmes et chiens de garde*

Resource ordonnançables

Les autres ressources concurrentes sont les ressources ordonnançables (SchedulableResource) pour lesquelles la concurrence d'exécution est gérée par un ordonnanceur logiciel qui détermine l'ordre et le temps dans lesquels les exécutions doivent s'effectuer. Cette ordonnancement peut être statique, c'est-à-dire effectué hors ligne, ou dynamique, c'est-à-dire effectué au cours de l'exécution. Outre les caractéristiques de priorités, de période, de taille de pile et de taille de tas, les ressources ordonnançables sont caractérisées par une échéance (deadlineElements) et une fenêtre de temps d'exécution (timeSliceElements) utilisée dans certaines politiques de partage du temps d'exécution[3]. En ce qui concerne les services, une res-

[2]Un chien de garde, encore désigné sous l'anglicisme watchdog, est un circuit électronique ou un logiciel utilisé pour s'assurer qu'un système ne reste pas bloqué à une étape particulière du traitement qu'il effectue. C'est une protection destinée généralement à redémarrer le système, si une action définie n'est pas exécutée dans un délai imparti.

[3]Une fenêtre de temps est utilisée pour les politiques d'ordonnancement par ***tourniquet*** (RoundRobin). Dans cette politique, les temps d'exécution sont partagées entre toutes les

source ordonnançable permet de stopper son exécution en attendant la terminaison de l'exécution d'une autre ressource (joinServices). Elle permet également de relâcher explicitement la ressource d'exécution sous-jacente (yieldServices) ou enfin de retarder son exécution (delayServices).

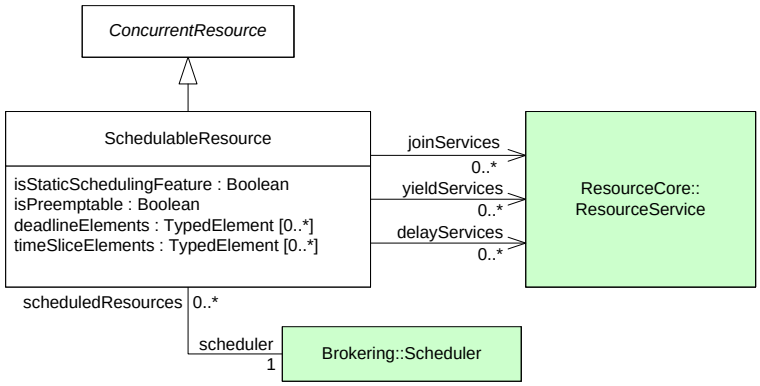

FIG. *6.8 – Description des ressources ordonnançables*

Définition du package Interaction

Resource d'interaction

Les ressources d'interaction (Interaction) proposent des mécanismes pour communiquer et synchroniser des contextes d'exécution concurrents. En effet, les ressources concurrentes n'évoluent généralement pas indépendamment dans un système. Il existe donc des ressources pour interagir par des flots de donnée (CommunicationResource) et pour interagir sur les flots de contrôle (SynchronisationResource) (voir figure 6.9).

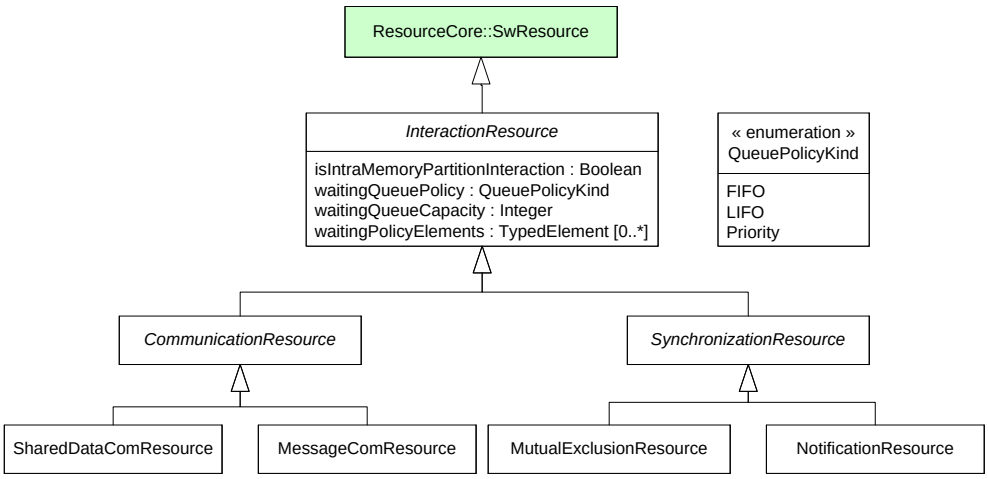

FIG. *6.9 – Hiérarchisation des ressources d'interactions*

ressources. Les ressources s'exécute tour à tour pendant cette fenêtre de temps et ce jusqu'à la fin de leurs exécutions.

Les interactions peuvent être limitées à une partition mémoire (isIntraMemoryPartitionInteraction). Elles sont régies par une politique d'attente (waitingPolicyElements) et un ensemble de file d'attente elle-même caractérisée par une capacité (waitingQueueCapacity) et une politique d'ordonnancement (waitingQueuePolicy). Par exemple, la prise d'un sémaphore[4] est caractérisée par une politique d'attente (attente indéfinie, attente par chien de garde, attente nulle par exemple) pour l'émetteur. De même, l'élection de la ressource prenant effectivement le sémaphore est soumise à une politique particulière (waitingQueuePolicy). La figure 6.10 schématise la liste d'attente d'un sémaphore s convoité par deux tâches t1 et t2. Dans cet exemple, t1 prendra le sémaphore lorsqu'il sera disponible puisque la liste d'attente est ordonnancée comme une FIFO (First In Fisrt Out). La date à laquelle le sémaphore sera disponible n'a aucune importance puisque l'appel pour prendre le sémaphore est régi par une politique d'attente indéfinie.

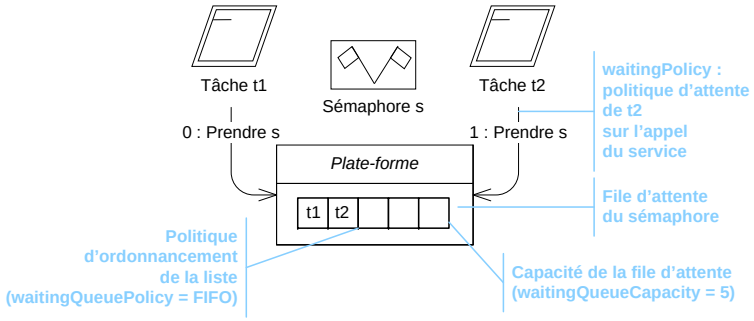

FIG. *6.10 – Schématisation des files d'attentes*

Resource de communication par message

Les mécanismes d'échange des données (CommunicationResource), c'est-à-dire les mécanismes de communication, sont classés en deux grandes familles : l'échange de message (MessageComResource) et le partage de zone mémoire (SharedDataComRessource). Les ressources de communication par message (voir MessageComResource dans la figure 6.11 page suivante) proposent des services d'envoi (sendServices) et de réception (receiveServices) d'une structure de données : un message. Ce message est caractérisé par une taille de donnée (messageSizeElements), fixe ou dynamique (isFixedMessageSize). Il est rangé dans une file régie par une taille (messageQueueCapacityElements) et une politique d'ordonnancement des messages (messageQueuePolicy).

Resource de communication par zone mémoire partagée

La seconde famille des ressources de communication est celle des ressources permettant directement de partager une zone mémoire (SharedDataComRessource). L'accès à cette zone mutualisée est protégé intrinsèquement par la ressource en fournissant des services de lecture (readServices) et d'écriture (writeServices) comme le montre la figure 6.12 page suivante .

[4]Un sémaphore est une variable protégée qui constitue la méthode utilisée couramment pour restreindre l'accès à des ressources partagées (par exemple un espace de stockage) dans un environnement de programmation concurrente. Un utilisateur prend un sémaphore pour accéder à la ressource. Il le rend lorsqu'il a terminé.

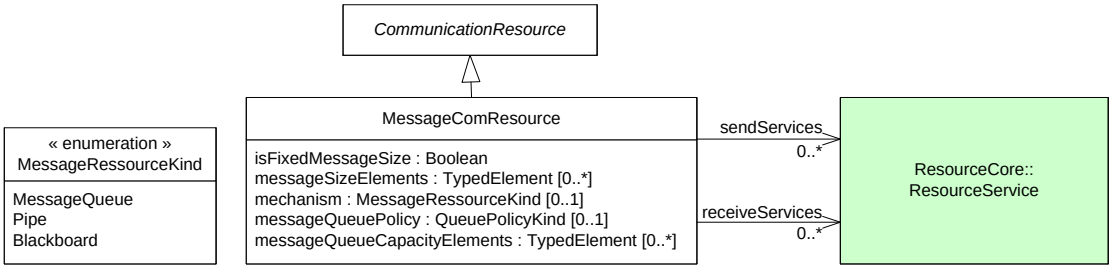

FIG. *6.11 – Description des ressources de communication par message*

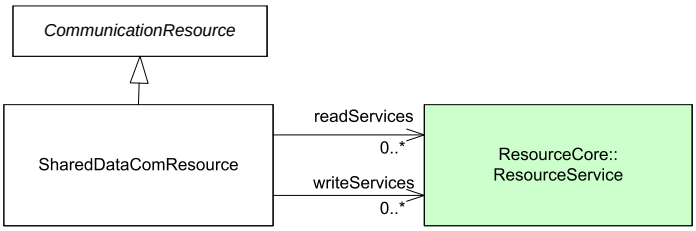

FIG. *6.12 – Description des ressources de partage mémoire*

Resource de synchronisation

Outre la communication de données, il existe aussi des mécanismes pour contrôler les flots d'exécutions, c'est-à-dire pour synchroniser les exécutions des ressources concurrentes (SynchronisationResource).

Resource de notification

Les ressources de notification (NotificationResource) proposent des services pour notifier à d'autres ressources du système un ou plusieurs événements. L'occurrence d'un événement est signalé (signalServices, flushServices) par une ressource source et attendue (waitServices) par une ou plusieurs ressources cibles. Cette occurrence peut être mémorisée (Memorized), comptée (Bounded[5] ou enfin être non mémorisée (Memoryless). La figure 6.13 page ci-contre illustre cette modélisation.

Resource d'exclusion mutuelle

La seconde famille de ressources dédiée à la synchronisation est celle des ressources d'exclusion mutuelle (MutualExclusionResource), c'est-à-dire des ressources synchronisant les accès mutuels à une même variable partagée. Cette synchronisation opère sur un ensemble de jeton pris (acquireServices) et vendus (releaseServices). Le protocole d'acquisition de ces jetons est décrit par une politique particulière (concurrentAccessProtocol).

Définition du package Brokering

Les ressources de gestion Brokering fournissent des intermédiaires pour manipuler les ressources logicielles et matérielles. Elles permettent, typi-

[5]Contrairement à l'option Memorized, l'option Bounded permet de connaitre le nombre de fois ou l'événement a été signalé. Dans la première option le signalement d'un événement est mémorisé par un booléen et donc deux occurrences successives ne sont par mémorisées.

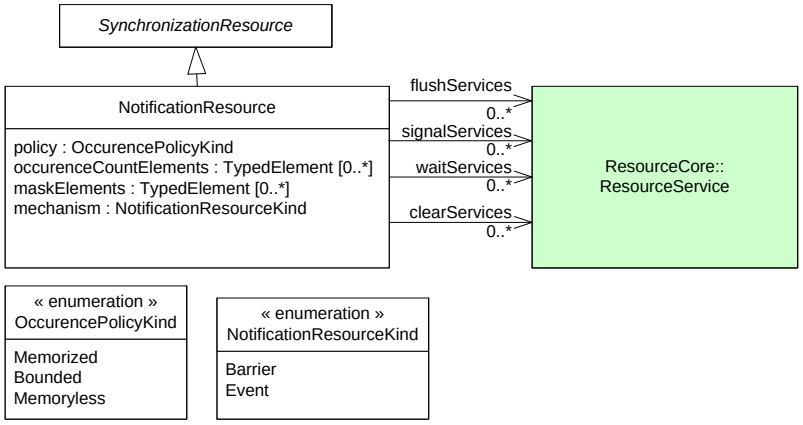

FIG. *6.13 – Description des ressources de notification*

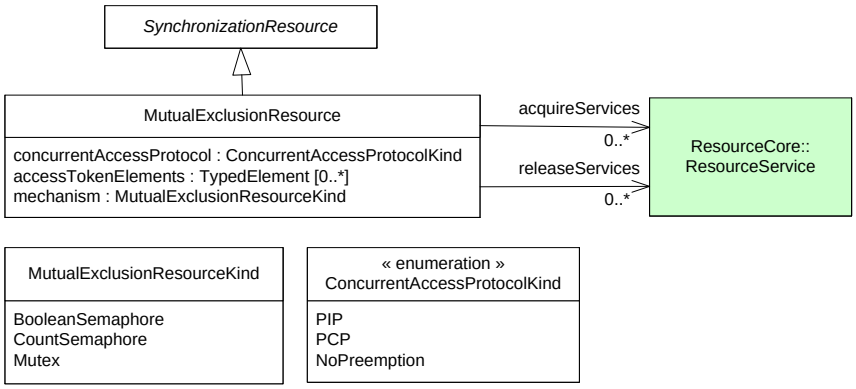

FIG. *6.14 – Description des ressources d'exclusion mutuelle*

quement, d'interfacer les périphériques, de gérer les mémoires virtuelles et d'ordonnancer les exécutions logicielles (voir figure 6.15).

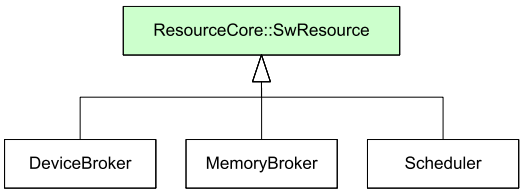

FIG. *6.15 – Hiérarchisation des ressources de gestion*

Resource d'interface des périphériques

Les ressources d'interface de périphériques (DeviceBroker) mettent en relation l'application et les ressources d'exécution des autres plates-formes. Ce sont typiquement les pilotes qui permettent de manipuler les périphériques sous-jacents . Ces pilotes permettent d'accéder à ces ressources par des services d'ouverture (openServices), de fermeture (closeServices), de control (controlServices), de lecture (readServices) et d'écriture (writeServices). Un périphérique ouvert est interfacé avec l'application. Suivant la politique d'accès, l'application peut lire et/ou écrire des données sur le périphérique. La figure 6.16 concrétise ces caractéristiques.

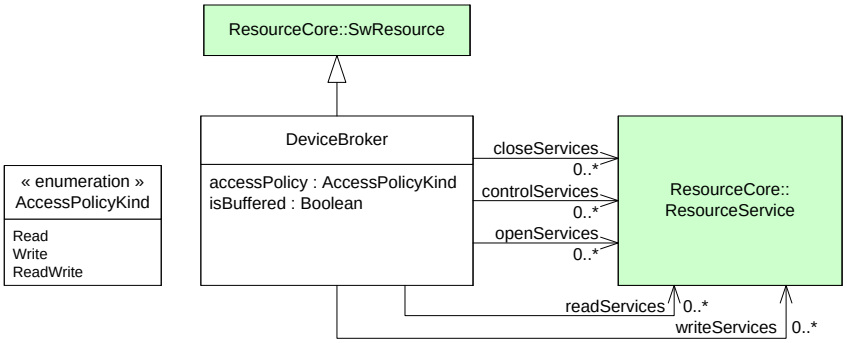

FIG. *6.16 – Description des ressources d'interface de périphériques*

Resource d'interface mémoire

Les ressources d'interface de mémoire (MemoryBroker) rassemblent les services de contrôle des espaces mémoires, c'est-à-dire les services liés aux mécanismes de pagination et de transfert mémoire. Comme le montre la figure 6.17 page suivante , une zone mémoire est ainsi caractérisée par une adresse (memoryBlockAddressElements) et une taille (memoryBlockSizeElements). Ce bloc peut être accédé en lecture et/ou en écriture (accessPolicy). Son transfert vers d'autres emplacements mémoire peut être interdit (lockServices) (typiquement dans le contexte des plates-formes munies de mémoire cache). Lorsque le système utilise des espaces mémoires virtuels, sa mise à jour vis à vis des données de la mémoire physique peut être explicitement demandée par l'application (mapServices).

Ordonnanceur

La dernière ressource décrite dans la figure 6.18 page ci-contre est responsable de l'ordonnancement des ressources d'exécution logi-

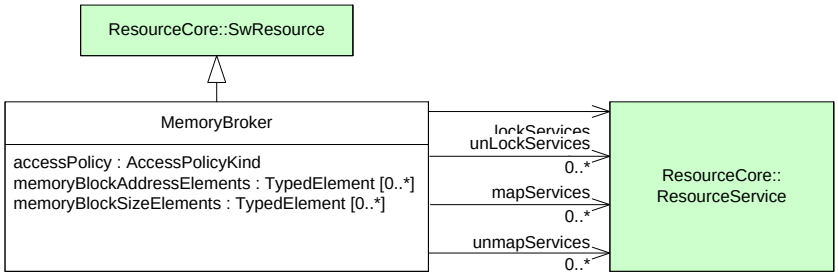

FIG. *6.17 – Description des ressources d'interface mémoire*

cielle. Cet ordonnanceur (Scheduler) possède ou non des capacités de préemption (isPreemptive). Il orchestre le temps et l'ordre d'un nombre fini de ressources. Cette orchestration suit une politique d'ordonnancement (schedPolicy). Le temps entre deux orchestrations peut être défini par l'utilisateur (tickerElements).

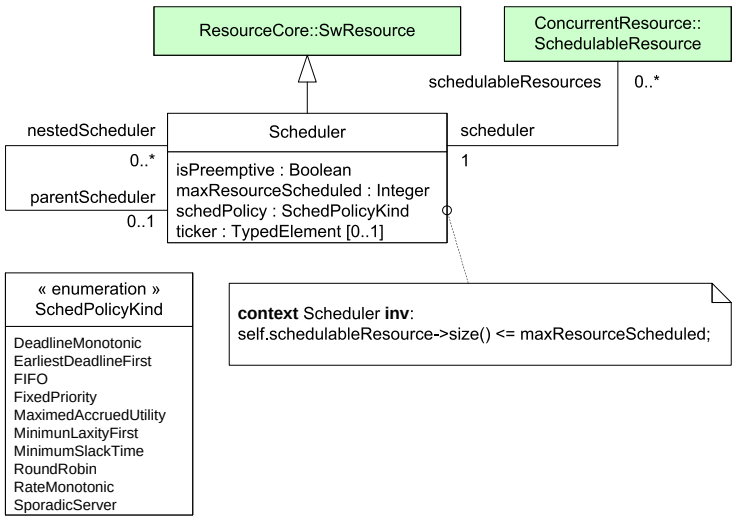

FIG. *6.18 – Description de l'ordonnanceur*

6.1.2 Implantation du modèle de domaine dans un profil UML SRM

L'implantation du profil correspondant au modèle de domaine précédent suit les règles d'implantation décrites dans la section 3.3 page 48 :

1. une SwResource étend la métaclasse Class
2. les ResourceService sont des Operations UML
3. les ResourceInstance sont des InstanceSpecification UML
4. les contraintes OCL sur le nombre d'instance et sur le nombre de classe par instance sont retranscrites dans le profil UML.

Outre les règles d'implantation du profil. Deux concepts ont été implantés comme des stéréotypes : SwAccesService et EntryPoint. Le premier est une extension aux Operations UML. Le second est utilisé pour lier une

application et une plate-forme, il est donc implanté sous la forme d'une dépendance. Le profil est décrit de manière détaillé dans le document [11]. Son implantation est rappelée dans l'annexe C page 155.

6.2 Évaluation du profil UML

L'objectif de cette section est d'évaluer la capacité de SRM à modéliser des plates-formes concrètes. Pour cela, trois plates-formes ont été modélisées : a) Rtai, b) Osek/Vdx-OS, et c) Arinc-653. Les deux premières plates-formes ont été étudiées pour la conception du modèle de domaine de SRM. La dernière est une norme dédiée à la spécification des exécutifs multitâches pour l'avionique Arinc-653[20]. Celle-ci permet d'évaluer l'utilisation de SRM sur une plate-forme non prise en compte dans l'étude ontologique.

6.2.1 Protocole de modélisation

Chacune plate-forme a été modélisée exclusivement avec le diagramme de classe. Les ressources sont modélisées comme des classes, les propriétés de ces ressources comme des attributs et les services de ces ressources comme des opérations. Les structures de données spécifiques à la plate-forme sont modélisées comme des types de données.

Les modèles des plates-formes Osek/Vdx-OS et Arinc-653 sont disponibles en annexe D.5 du document [11]. Le modèle de Rtai a été réalisé par des industriels dans l'étude [78].

6.2.2 Résultats expérimentaux

La figure 6.19 quantifie, par métatype, le nombre d'élément modélisé pour chacune des plates-formes.

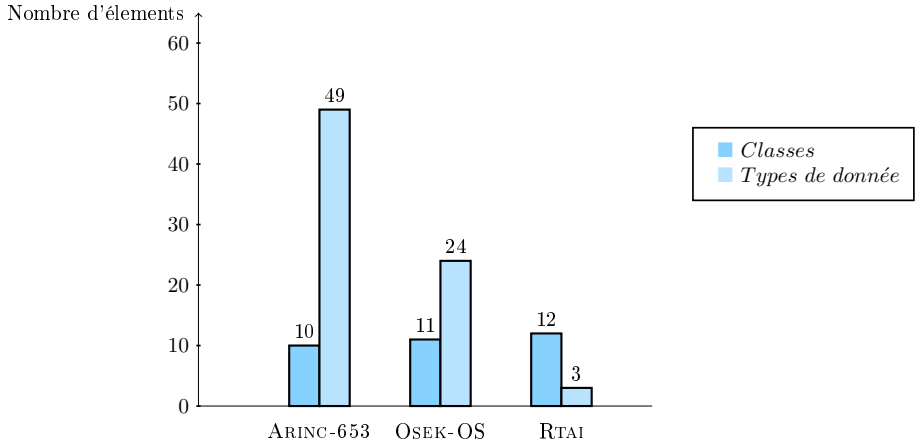

Fig. *6.19 – Nombre d'éléments modélisés pour l'évaluation de* SRM

Il n'est pas surprenant de constater un grand nombre de type de données pour les normes ARINC-653 et OSEK/VDX-OS, car leurs APIs spécifient des structures de données que l'utilisateur doit utiliser pour typer les variables de son application.

Taux de couverture des ressources

Le tableau 6.1 illustre les résultats de modélisation des classes. Dans ce tableau N décrit le nombre de classes par plate-forme, $N_{simple\ stéréotype}$ le nombre de classes stéréotypées une seule fois et $N_{multiples\ stéréotypes}$ le nombre de classes stéréotypées plusieurs fois. Le pourcentage représente la part des ressources stéréotypées au moins une fois $((N_{simple\ stéréotype} + N_{multiples\ stéréotypes}) * 100/N)$.

TAB. *6.1 – Métriques des classes modélisées par le profil* UML SRM

Plate-forme	N	$N_{simple\ stéréotype}$	$N_{multiples\ stéréotypes}$	%
ARINC-653	10	7	1	80
OSEK/VDX-OS	11	8	0	73
RTAI	12	8	3	92

Bien que le taux de couverture soit bon, il n'est pas maximal. Les ressources qui ne sont pas identifiées SRM sont : a) des mécanismes de tolérance aux fautes (par exemple, la gestion des modes de fonctionnement et la gestion des erreurs), b) des mécanismes de gestion des plates-formes en elles mêmes (par exemple, la gestion des démarrages et des arrêts du système), et c) des mécanismes de gestion du temps. Ces résultats étaient attendus puisque ces agences ne sont pas prises en compte dans la conception du profil SRM. Ils seraient tout à fait possible de les modéliser mais ce besoin n'a pas été considéré lors de la conception du profil et plus particulièrement lors de la description du contexte de cette étude (voir figure 1.7 page 21). Ils ne remettent donc pas en cause l'utilité de SRM.

A ce stade de l'expérimentation, les classes qui n'ont pas été décrites par SRM sont retirées pour ne pas fausser les résultats futurs.

Taux de couverture des propriétés et des paramètres

Le tableau 6.2 quantifie la part des éléments typés référencés par les stéréotypes de SRM. Seul les propriétés et les paramètres sont considérés dans ces résultats. Dans ce tableau N décrit le nombre d'éléments par plate-forme, $N_{simple\ référence}$ le nombre d'éléments référencés une seule fois. Le pourcentage représente la part de ces des éléments. Enfin, le dernier pourcentage représente la part des paramètres et des propriétés qui n'ont pas été modélisées et qui sont relatifs à la gestion d'erreur (par exemple un code d'erreur dans un retour de fonction).

TAB. *6.2 – Métriques des éléments typés décrits par le profil* UML SRM

Plate-forme	N	$N_{simple\ référence}$	%	$\%_{erreur}$
ARINC-653	208	107	51.4	18
OSEK/VDX-OS	70	34	48.5	27
RTAI	377	206	54.6	10

Contrairement à la modélisation des ressources en elle même, le profil SRM faillit pour la modélisation des caractéristiques et ceci pour deux raisons principales : a) les retours d'erreurs ne sont pas modélisées, et b) l'hétérogénéité des plates-formes. En effet, les éléments typés utilisés pour décrire des codes d'erreurs ne sont pas modélisé avec SRM. Ce besoin n'a pas été considéré lors de la conception du profil et plus particulièrement lors de la description du contexte de cette étude (voir figure 1.7 page 21). La modélisation de ces codes d'erreur serait pourtant importante puisqu'elle représente, en moyenne, 20% des éléments non modélisés. La deuxième cause d'erreur est due à l'hétérogénéité des plates-formes en elle-même. Etant donné les résultats du tableau 6.1 page précédente, il est possible de définir une base conceptuelle commune pour des ressources. Néanmoins, il est beaucoup plus difficile de déterminer cette même base pour des propriétés et des paramètres. L'utilisation des expressions suggérées au chapitre4 page 57 doit permettre d'améliorer ces résultats. Une telle étude n'a pas été réalisée dans cette thèse puisqu'elle dépend d'une méthodologie externe.

Taux de couverture des services

Le tableau 6.3 quantifie les opérations décrites par les stéréotypes de SRM. Ces opérations sont soit référencées par les propriétés des stéréotypes ou soit directement stéréotypées par SwAccesService dans le cas des accesseurs. Dans ce tableau N décrit le nombre d'éléments par plate-forme, $N_{simple\ référence}$ le nombre d'opérations référencées une seule fois, $N_{multiples\ référence}$ le nombre d'opérations référencées plusieurs fois et $N_{stéréotype}$ le nombre d'opérations stéréotypées par SwAccesService. Le pourcentage représente la part des opérations décrites par SRM $((N_{simple\ référence} + N_{multiples\ référence} + N_{stéréotype}) * 100/N)$.

TAB. 6.3 – *Métriques des opérations décrites par le profil* UML SRM

Plate-forme	N	$N_{simple\ référence}$	$N_{multiples\ références}$	$N_{stéréotype}$	%
ARINC-653	49	27	1	19	96
OSEK/VDX-OS	27	17	4	5	96
RTAI	146	77	5	13	65

Tout comme pour les ressources, la couverture de description des services est bonne. Les services qui ne sont pas modélisés dans la plate-forme RTAI sont principalement dédiés à la gestion des plates-formes matérielles distribuées (par exemple, Remote Protocol Call) ou multiprocesseurs, ce qui n'est pas pris en compte dans le contexte de travail de cette étude.

6.2.3 Conclusion de l'évaluation

Les résultats de cette expérimentation montrent que le profil SRM est utilisable pour décrire des plates-formes d'exécution multitâches. L'histogramme 6.20 page suivante résume les principaux résultats de cette étude. Ce profil ne permet pas de distinguer tous les éléments d'une plate-forme. Les éléments non modélisés concernent :

1) la tolérance aux fautes (par ex., gestion des modes de fonctionnement et des erreurs),
2) la gestion des plates-formes en elles mêmes (par ex., démarrage et arrêt des systèmes),
3) la gestion du temps, et
4) la gestion des architectures multiprocesseurs et des architectures distribuées.

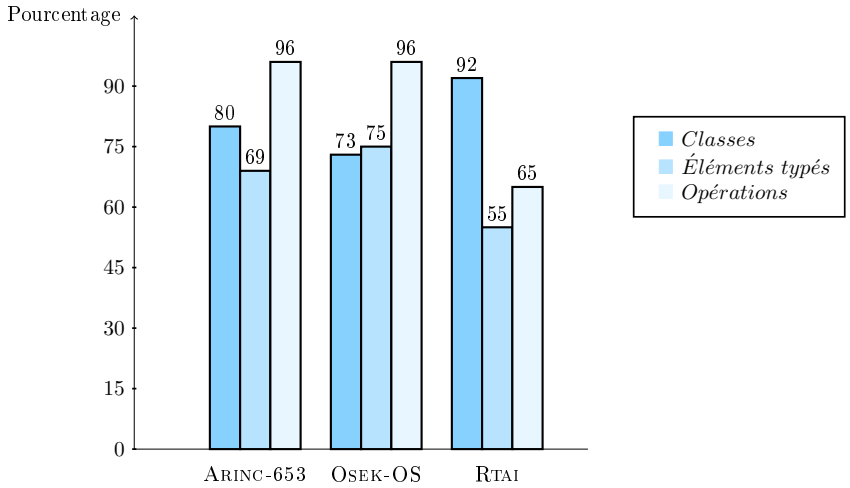

FIG. *6.20 – Synthèse des résultats expérimentaux sur* SRM

Plusieurs solutions sont envisageables pour y remédier :

1) étendre le modèle de domaine de SRM pour décrire les agences manquantes
2) utiliser des expressions pour décrire les éléments typés et des activités, pour les services dans le cas ou la correspondance avec SRM n'est pas directe (voir les heuristiques de modélisation du chapitre 4 page 57), et
3) utiliser SRM conjointement avec d'autres profils appliquant le motif RESOURCE-SERVICE pour identifier complètement chacun des concepts.

.

6.3 DISCUSSIONS

Une ressource modélise l'entité ou le gestionnaire de cette entité ?

Outre l'adéquation du motif RESOURCE-SERVICE pour concevoir des métamodèles de plates-formes, il apparait que son application nécessite une approche de conception particulière. Lorsque des métamodèles de plates-formes sont décrites avec le motif, le concept de ressource agrège au sein d'une même abstraction à la fois l'entité modélisée et le gestionnaire de cette entité. L'entité est caractérisée par des propriétés, le gestionnaire par des propriétés et des services. Par exemple, dans la modélisation des ressources ordonnançables SchedulableResource, c'est-à-dire des fils d'exécution (des tâches), ce n'est pas le fil d'exécution lui même qui est modélisé mais une abstraction combinant le fil d'exécution et le gestionnaire de ce fil

d'exécution. La figure 6.21 représente cette abstraction. Cette figure n'est pas un diagramme UML mais une illustration de ce qu'il faut considérer lorsque le métamodèle de la plate-forme est constitué. Dans cette figure, les propriétés de la ressource sont en fait les propriétés de l'entité et les services sont offerts par le gestionnaire. D'un point de vue de l'utilisateur ces deux concepts sont abstraits en un seul concept : la ressource.

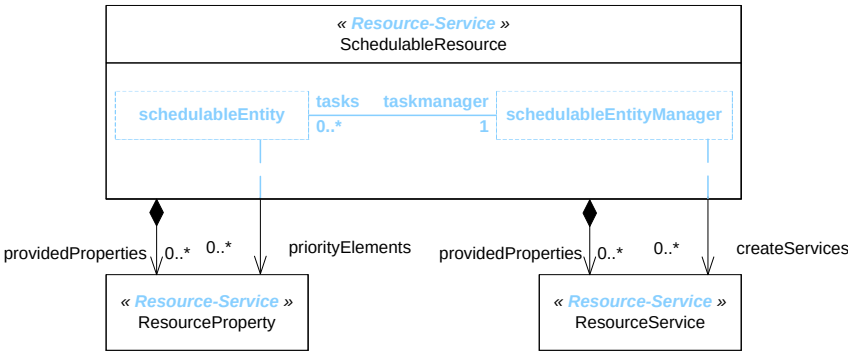

FIG. 6.21 – *Abstraction mise en œuvre par le motif* RESOURCE-SERVICE *lors de la conception des métamodèles de plates-formes*

Propriété ou référence ?

Dans le métamodèle SRM, certains attributs sont typés par des types primitifs (booléen, entier, énumération) et d'autres sont spécifiés pour référencer des éléments. Le choix de conception (typage primitif ou référence) n'est pas spécifié par le motif. Un choix par «typage» permet de caractériser la ressource. Un choix par référencement permet d'identifier une caractéristique. Ce choix appartient au concepteur du métamodèle ; il n'est cependant pas sans conséquence. Par exemple, une ressource ordonnançable dans SRM possède intrinsèquement un type qui spécifie si la ressource est périodique ou apériodique. Cette caractéristique intrinsèque impose au modeleur la description de plusieurs entités ordonnançables, une pour chaque type. La figure 6.22 page ci-contre illustre le cas d'un Process de la plate-forme ARINC-653 qui est périodique ou apériodique. Cette caractéristique est définie lors de la création (create_process) en attribuant une valeur particulière à l'argument période (valeur définie par l'implantation). Ainsi, la modélisation de ce Process par SRM impose de dissocier deux classes et d'associer au service une contrainte OCL imposant la valeur de la période.

Le point d'entrée appartient-il au métamodèle de plate-forme ?

Lors de l'implantation du profil SRM, il a été nécessaire de créer un stéréotype pour le concept de point d'entrée EntryPoint. Ce stéréotype étend la métaclasse Dependency pour permettre de décrire des connexions entre l'application et la plate-forme. Sa description au sein du métamodèle de plate-forme est donc discutable. Elle doit donc faire l'objet d'un métamodèle d'allocation spécifique au multitâche puisque ce n'est pas une ressource à part entière.

L'accesseur doit-il enrichir le motif ?

Outre ce stéréotype EntryPoint, il a été nécessaire de réifier le concept de service par le concept d'accesseur, c'est-à-dire de SwAccessService. Étant donné la quantité de services qu'il a permit d'identifier dans le tableau 6.3 page 106, il est naturel de vouloir enrichir le motif RESOURCE-

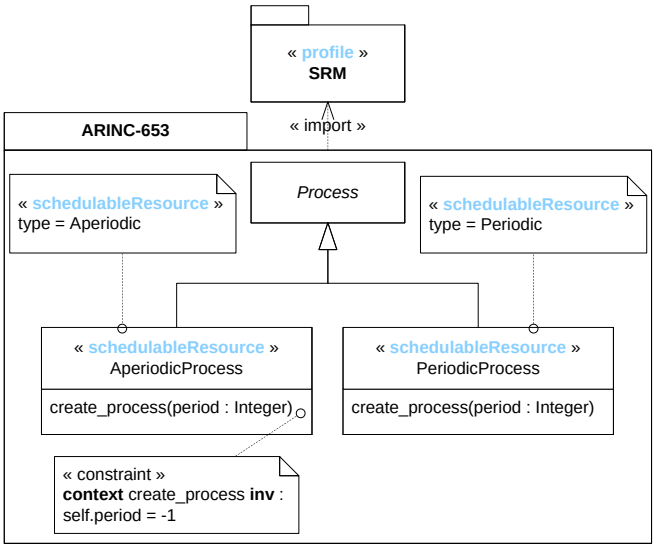

FIG. 6.22 – *Modélisation d'un Process* ARINC-653 *avec* SRM

SERVICE par ce concept. Le motif Resource-Service est décrit essentiellement pour les métamodèles de plates-formes logicielles d'exécution. Néanmoins, il semble, en première abord, utilisable pour décrire des métamodèles de plates-formes matérielles d'exécution. La figure 6.23 page suivante montre un extrait de modélisation d'une ressource arithmétique et logique (ALUResource) fournissant les services d'addition (addServices) et de soustraction (subServices). Le stéréotype résultant est utilisé pour décrire le processeur MC68000. Une telle modélisation doit pouvoir permettre la génération de code assembleur. Dans le domaine du matériel, la notion d'accesseur n'a pas de finalité (les services «set» et «get» ne sont pas des concepts du matériel). C'est pourquoi, le motif n'est pas enrichi dans cette étude.

Modélisation matérielle avec le motif

Enfin, des discussions peuvent être menées suite aux métriques sur les modèles de plates-formes décrits avec SRM. D'après ces mesures, il apparait que SRM est adéquate pour modéliser une grande partie des éléments d'une plate-forme. Néanmoins, la description n'est jamais complète. Elle est même parfois difficile, notamment pour les propriétés des ressources et les paramètres des services. En fait, même si, les éléments ne sont pas identifiés, ils ne sont pas pour autant non modélisés. Ils sont en effet modélisés sous la forme de ressources logicielles (SwResource est un stéréotype non abstrait), de propriétés UML (ResourceProperty dans le motif) et de paramètres (ServiceParameter dans le motif). Ainsi, des générateurs de code ou des transformations de modèles peuvent les manipulés. Ils ne seront par contre pas directement dissociables des autres éléments non identifiés.

SRM n'est pas complet, est-il toujours légitime ?

Un des intérêts des modèles de plates-formes est de capitaliser les outils manipulant les modèles de plates-formes. Cette capitalisation passe par une réutilisation et par une aide à la conception. Par exemple, pour deux ressources appartenant à deux plates-formes différentes décrites par le même stéréotype SRM, il est possible de dresser automatiquement un

Est-ce qu'un métamodèle de plate-forme peut aider à la conception des transformations ?

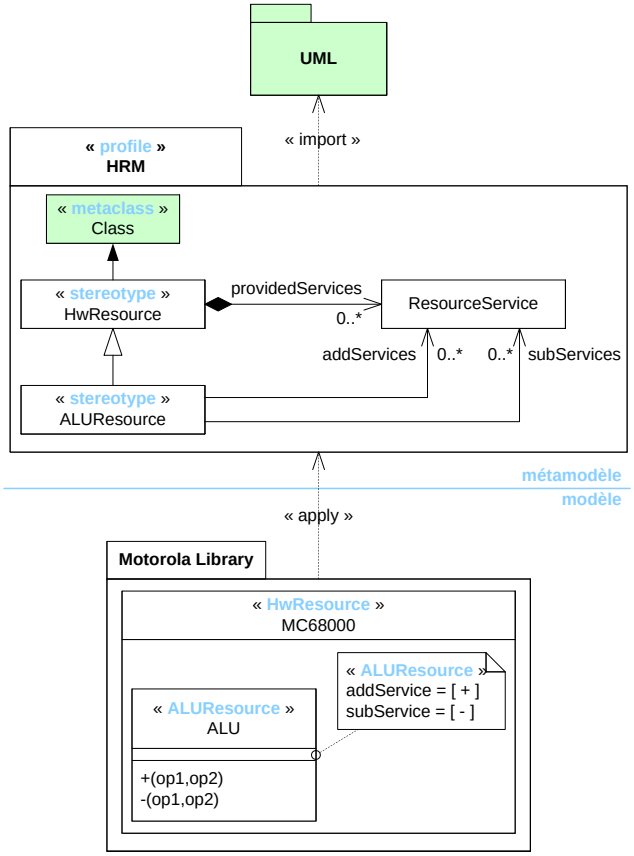

FIG. 6.23 – *Utilisation du motif* RESOURCE-SERVICE *pour la métamodélisation des plates-formes matérielles d'exécution*

lien de tissage entre ces éléments. Ce lien de tissage est en fait implicite par l'utilisation d'une description commune. Il permet de générer une partie des règles de transformation permettant de passer de la plate-forme source à la plate-forme cible. Puisque SRM n'est pas complet, il ne permettra donc pas de décrire automatiquement l'ensemble de ces règles (et donc l'ensemble des règles de transformation). Il peut cependant y participer. Par exemple, la figure 6.24 illustre le pourcentage de ressource décrit de la même façon entre les trois plates-formes de l'étude expérimentale, c'est-à-dire le pourcentage de ressources pour lesquelles l'opérateur d'équivalence spécifié dans l'algorithme 2 page 85 renvois *Vrai*. Pour chacun des couples (plates-forme source, plate-forme cible), les résultats sont illustrés dans la figure 6.24 .

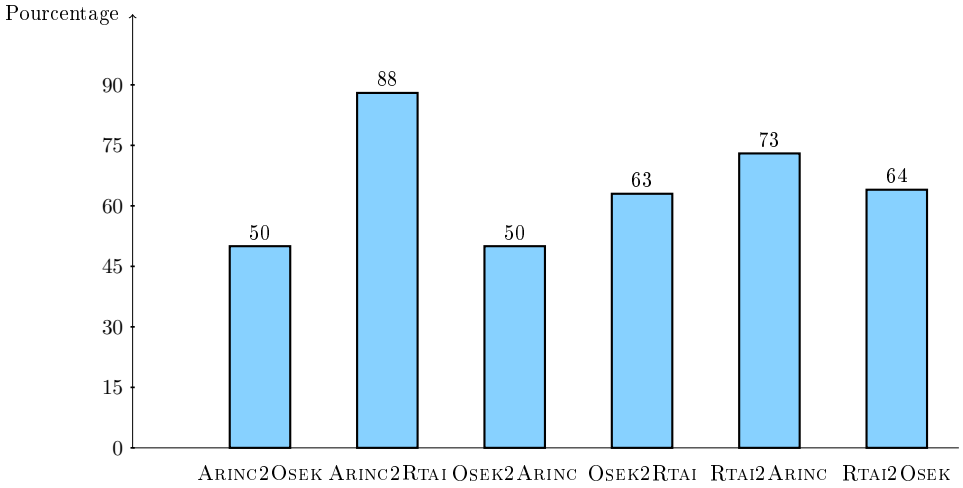

FIG. *6.24 – Pourcentage des classes décrites de la même façon par* SRM *dans différents modèles de plate-forme*

Ces résultats montrent les taux de productivité, au pire cas, d'une transformation endogène de portage basée sur SRM. Par exemple, dans le pire des cas, une transformation basée sur SRM peut déduire la moitié des liens de tissages entre les ressources ARINC et les concepts OSEK/VDX-OS. Ces tissages sont déduits parce que les ressources ARINC et OSEK/VDX-OS sont décrites par les même stéréotypes. Il apparait surprenant que dans certains cas, par exemple ARINC2RTAI et RTAI2ARINC, ce taux ne soit pas le même dans les deux sens de transformations (ARINC vers RTAI puis RTAI vers ARINC). Cela s'explique dans le tableau 6.1 page 105 par l'utilisation des mêmes stéréotypes pour décrire plusieurs ressources. Pour un même élément ARINC, il existe plusieurs éléments RTAI équivalents. La génération des liens de tissages n'est alors plus déterministe. Associé avec une méthodologie imposant qu'un stéréotype n'est appliqué qu'une seule fois[6], cette génération est à nouveau déterministe.

[6]Une étape de la méthodologie peux optimiser la modèle de plate-forme pour supprimer les stéréotypes appliqués plusieurs fois. Ainsi, si plusieurs ressources sont stéréotypées de la même façon, un choix est fait et certains stéréotypes sont supprimés dans un modèle intermédiaire. Ce choix peut être explicité par l'utilisateur ou déduit de contraintes de performances par exemple.

Conclusion

Dans ce chapitre le motif Resource-Service a été utilisé pour décrire le métamodèle SRM. Ce métamodèle est implanté dans un profil UML SRM. Des métriques sur la description de modèle de plate-forme ont été effectuées. Elles montrent que le profil SRM est utilisable pour décrire des plates-formes logicielles d'exécution multitâches. Ces modélisations ne sont pas complètes mais elles permettent d'affirmer que le profil SRM est une contribution à la métamodélisation des plates-formes du domaine multitâche. La figure 6.25 illustre la satisfaction des besoins de modélisation par le profil.

Le profil SRM et le motif ont été publiés dans [79, 80]. L'évaluation a été publiée dans [81]. Il a été utilisé par des partenaires industriels dans [78, 82]. L'OMG l'a normalisé par l'intermédiaire du profil UML MARTE[7] dédié à la modélisation et à l'analyse des systèmes temps-réel embarqués [11].

Possédant le motif, un exemple de métamodèle et des exemples de modèles de plates-formes, il est possible d'expérimenter des transformations de modèles basées sur des modèles de plates-formes explicites. C'est le sujet du chapitre 7 page ci-contre.

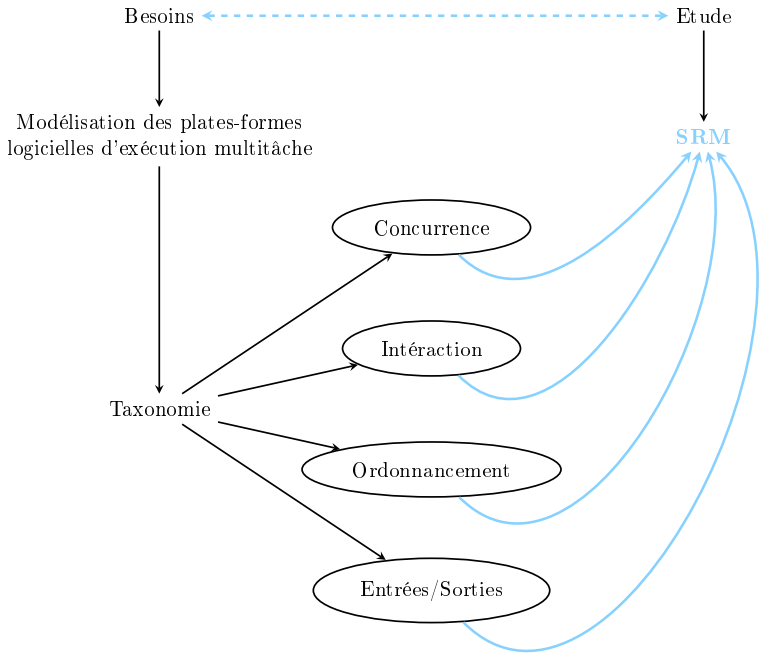

Fig. 6.25 – *Satisfaction des besoins de modélisation des plates-formes multitâches dans cette étude*

[7] MARTE : Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE).

7 Intégration des modèles de plates-formes multitâches dans une ingénierie générative expérimentale

Sommaire

Ce chapitre vise à expérimenter l'intégration des modèles de plates-formes dans un processus génératif outillé. Les résultats du chapitre 6 page 91 justifient l'expérimentation de transformations endogènes basées sur des modèles de plates-formes. Ils montrent, en effet, que le profil SRM peut être utilisé comme un pivot sémantique entre différentes plates-formes logicielles d'exécution multitâche. Ce pivot est utilisé pour porter une application spécifique à une ou plusieurs plates-formes sources vers une ou plusieurs plates-formes cibles.

Pour cela, ce chapitre est divisé en trois sections. Premièrement, le contexte expérimental des transformations est décrit. Deuxièmement, une architecture de transformation est définie. Enfin, dans une troisième et dernière section, cette architecture est évaluée.

7.1 Définition du contexte expérimental

La mise en place d'un contexte expérimental nécessite : a) la définition d'hypothèses de modélisation, b) la description des modèles fournis en entrée du processus, et c) la présentation d'un cas d'étude autour de ces modèles.

7.1.1 Définition des hypothèses de modélisation

Les chapitres 3 page 37 et 4 page 57 ont montré qu'il existait de nombreuses possibilités pour modéliser les plates-formes et les applications spécifiques à ces plates-formes. Dans le cadre de cette étude, il est admis que :

1. Les modèles de l'application et les modèles de plates-formes sont structuraux. Ils sont essentiellement composés de classes et de types de données possédant des attributs, des opérations des associations et des héritages,
2. Les classes des modèles applicatifs sont des singletons, c'est-à-dire qu'il n'existe qu'une et une seule instance pour chaque classe,
3. Pour toutes les plates-formes décrites comme des profils UML, les modèles UML correspondant à ces profils sont fournis. Ce modèle est une représentation UML du modèle de domaine utilisé pour décrire le profil,
4. Lorsque les modèles de plates-formes doivent être enrichis par des motifs de conception, ceux-ci sont représentés sous la forme de collaborations UML (voir la section 4.2.1 page 61). Les propriétés de ces collaborations référencent alors les ressources des modèles de plates-formes, et
5. Les connexions entre les éléments applicatifs et les ressources des plates-formes sont exclusivement des relations de typage et des relations de conformité par applications de stéréotypes (voir la section 3.1 page 39).

7.1.2 Présentation des modèles plates-formes

Dans cette étude trois plates-formes sont étudiées : Hlam, Osek/Vdx-OS et Cheddar.

► **HLAM –** Hlam est un sous ensemble de la spécification High Level Application Modeling (Hlam) de la norme UML MARTE (voir chapitre 13 de [11]). Ce profil UML permet de décrire à un haut niveau d'abstraction des exécutions concurrentes. Dans cette étude, seuls les concepts d'unité temps réel rtUnit, et de services temps réel rtService sont considérés. Dans cette étude, une unité temps réel est constituée d'un ordonnanceur qui orchestre l'exécution des services temps réel. Chaque service

temps réel est alors considéré comme une ressource ordonnançable périodique. Ce modèle, illustré dans la figure 7.1 en tant que profil et dans la figure 7.2 en tant que modèle, est très largement simplifié par rapport à la norme MARTE. Il permet de réaliser une première expérimentation.

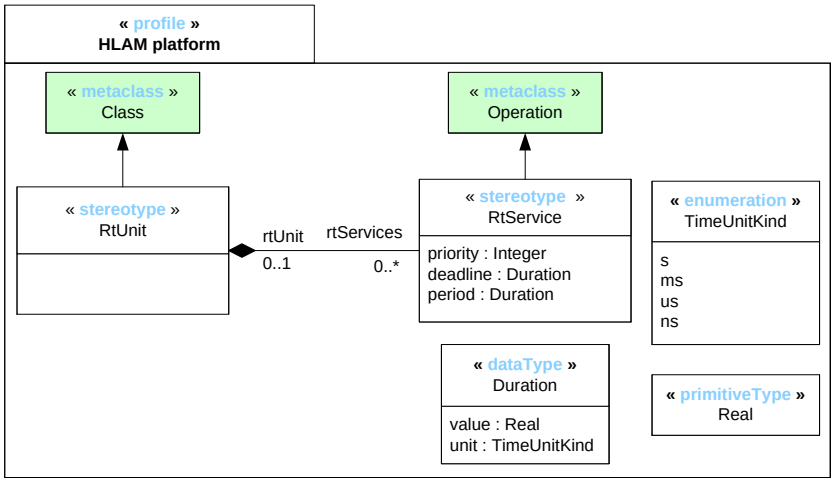

FIG. 7.1 – *Extrait du profil* HLAM *pris en compte dans cette étude*

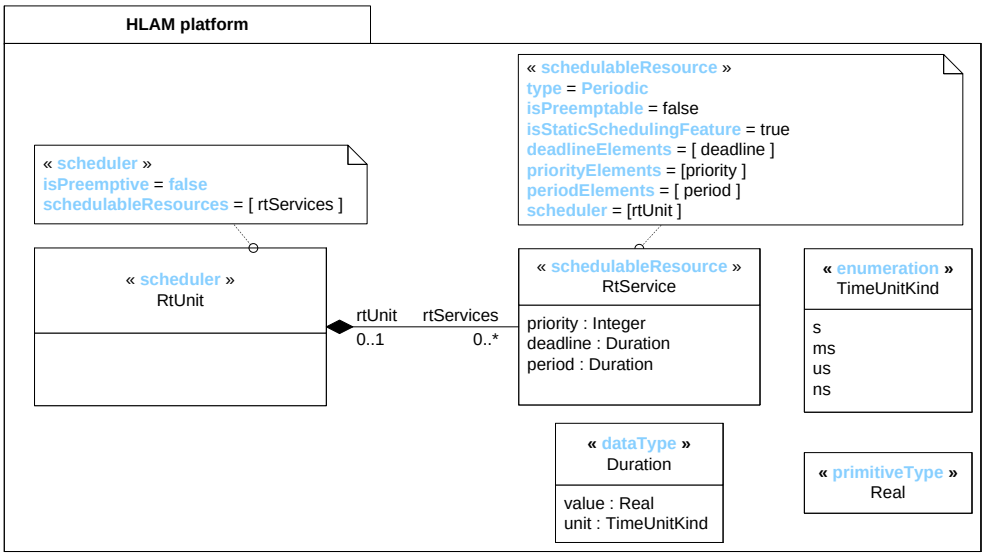

FIG. 7.2 – *Extrait du modèle de domaine de la plate-forme* HLAM *utilisé pour produire le profil* HLAM

► **OSEK/VDX-OS –** OSEK/VDX-OS est la spécification d'un exécutif temps réel multitâche [19]. La partie du modèle utilisée pour illustrer ce cas d'étude est décrite dans la figure 7.3 page ci-contre . Les ressources prises en compte sont dédiées à l'exécution. Les tâches sont des ressources ordonnancées par un ordonnanceur Scheduler. Ces tâches sont basiques ou étendues. Une tâche étendue peut se mettre en attente d'événements, ce que ne peut pas faire une tâche basique (cette caractéristique n'est pas

représentée dans l'extrait 7.3). Les alarmes Alarm sont des interruptions périodiques permettant d'activer des tâches, de signaler des événements ou d'activer des interruptions. Ces alarmes sont liées à des compteurs counter et elles sont configurées par des types de données permettant de préciser leurs périodes (AutoStart[1]) et leurs actions (action[2]).

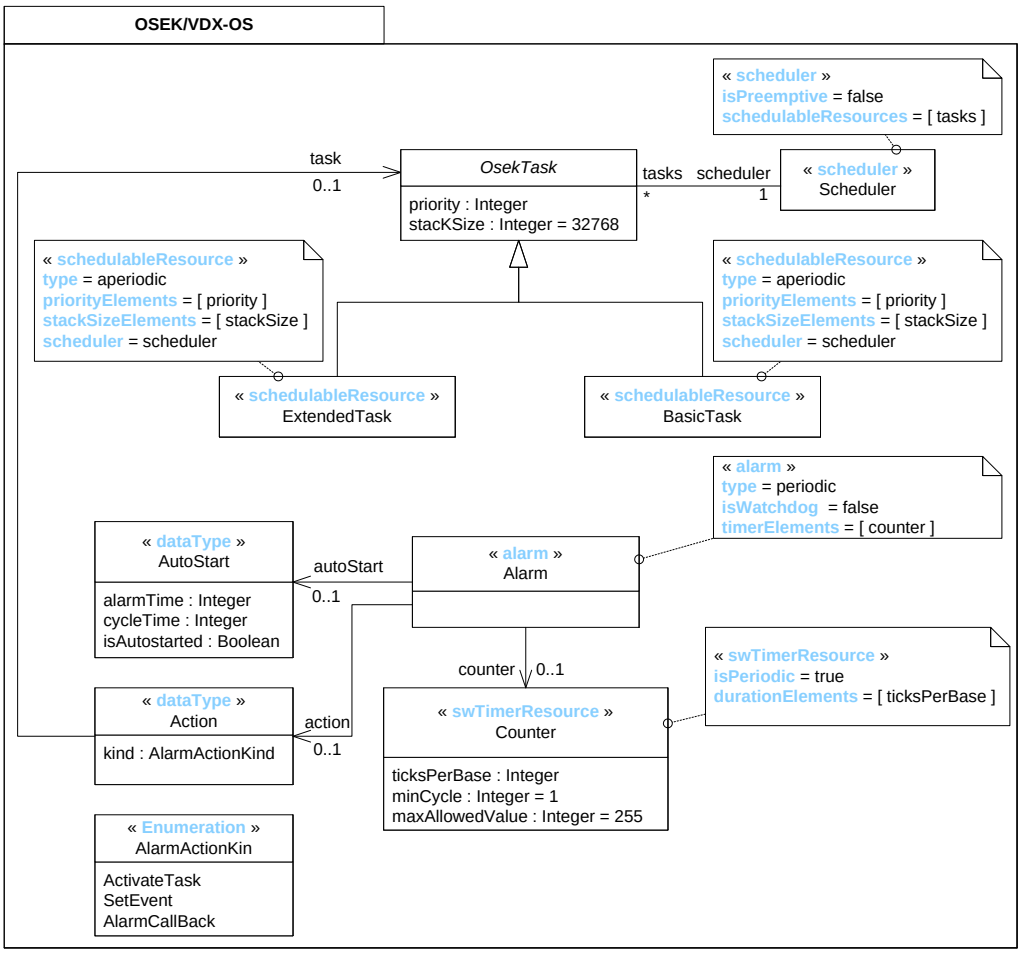

FIG. 7.3 – *Extrait du modèle de la plate-forme* OSEK/VDX-OS

Ce modèle est complété par un motif d'implantation des tâches périodiques sur la plate-forme OSEK/VDX-OS. Sur cette plate-forme, une tâche périodique est implantée en activant périodiquement une tâche étendue par l'intermédiaire d'une alarme. Comme le montre la figure 7.4 page suivante , la modélisation de ce motif implanté sur la plate-forme OSEK/VDX-OS se concrétise par une collaboration dans laquelle un compteur (50msTimer) est lié à une alarme (alarm). L'action (action) de cette alarme est l'activation (kind=ActivateTask) d'une tâche étendue (task). Dans

[1]Dans le cas des alarmes auto démarrées (isAutoStarted = true), l'attribut alarmTime précise le nombre de tick du compteur à partir duquel l'alarme se déclenche pour la première fois et l'attribut cycleTime précise le nombre de tick du compteur à partir duquel l'alarme se réarme (son échéance).

[2]Lorsque l'alarme se déclenche, elle peut activer une tâche, émettre une interruption ou signaler un événement.

l'implantation de ce motif il est imposé que le compteur soit périodique de période cinquante millisecondes (ticksPerBase=1 et minCycle=1) [3].

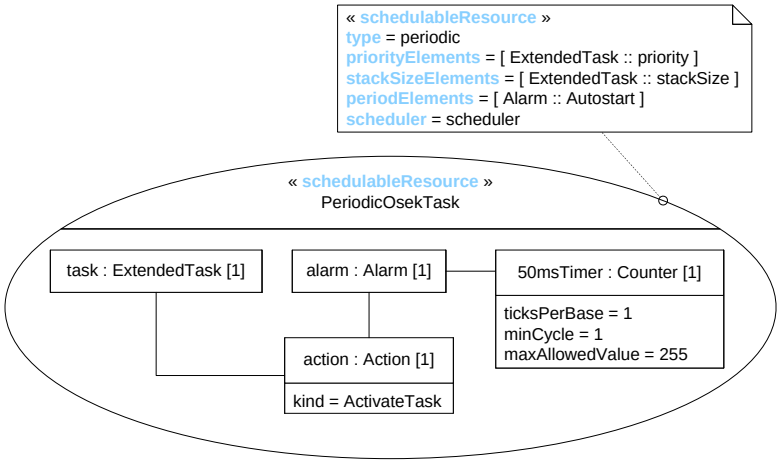

FIG. 7.4 – *Motif de conception d'une tâche périodique sur la plate-forme* OSEK/VDX-OS *de cette étude*

► **Cheddar –** CHEDDAR est un outil[4] destiné à l'analyse d'ordonnancement des applications temps réel [83]. Ce n'est pas une plate-forme d'exécution à proprement parlé. Cependant, pour les besoins de cette étude, elle est considérée comme une plate-forme exécutant une application multitâche. Cette exécution produit les chronogrammes représentant l'exécution des tâches. Ces chronogrammes permettent d'effectuer une analyse d'ordonnançabilité du système[5]. Une application CHEDDAR est composée de processeurs, d'espaces d'adressage, de tâches, de ressources partagées et de messages. Le modèle de cette plate-forme, plus conséquent que les modèles précédents, est détaillé dans l'annexe D.1 page 159.

7.1.3 Présentation du cas d'étude

L'application expérimentale considérée dans cette étude est représentée dans la figure 7.5 page ci-contre . Elle reprend la classe Controller de l'application de robotique décrite dans la section 3.1 page 39 en la spécialisant pour la plate-forme HLAM. Un Controller, destiné à contrôler la trajectoire d'une plate-forme robotique mobile, est une unité temps réel possédant deux services temps réel qui sont updateControl et compute. La commande des actionneurs, updateControl, doit s'effectuer toutes les cinquante millisecondes. Le calcul de la trajectoire, compute, s'effectue quant à lui toutes les cent millisecondes.

[3] Ces valeurs par défaut sont imposées par l'implantation OSEK/VDX-OS utilisé. Dans l'implantation OSEK/VDX-OS utilisée pour cette étude, l'implantation TRAMPOLINE (http ://trampoline.rts-software.org/) [60], le compteur est associé à un timer dont l'impulsion est fixée, par défaut, à cinquante millisecondes.

[4] http ://beru.univ-brest.fr/ s̃inghoff/cheddar/index-fr.html

[5] Dans le cadre de cette étude, un ordonnanceur OSEK/VDX-OS a été implanté dans cet outil.

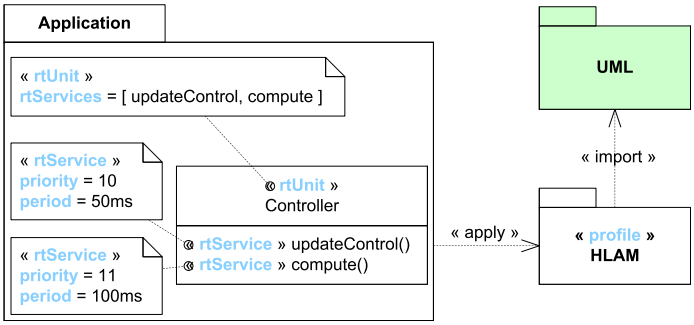

FIG. 7.5 – *Application prise en compte dans le processus génératif*

A partir de ce modèle applicatif (ce PIM), cette étude vise à produire les descriptions structurelles des applications multitâches spécifiques aux plates-formes OSEK/VDX-OS et CHEDDAR (les PSM). Concrètement, le processus génératif produit a) le fichier de configuration d'un exécutif OSEK/VDX-OS (l'implantation TRAMPOLINE) et b) le fichier de configuration de l'outil d'analyse d'ordonnancement CHEDDAR. Ce processus est décrit dans la figure 7.6 page suivante . Il est composé de quatre étapes :

1. L'application HLAM (figure 7.5) est transformée en une application OSEK/VDX-OS,
2. L'application OSEK/VDX-OS est transformée en une application CHEDDAR.
3. Le fichier de configuration de l'exécutif OSEK/VDX-OS est généré à partir du modèle de l'application OSEK/VDX-OS. Ce fichier est décrit dans un langage particulier nommé OSEK IMPLEMENTATION LANGUAGE (OIL). Il est utilisé en entré de la chaine de compilation pour configurer l'exécutif. Il décrit l'instanciation des ressources de la plate-forme OSEK/VDX-OS[6].
4. Le fichier d'entrée de l'outil CHEDDAR, fichier XML, est généré à partir du modèle de l'application CHEDDAR.

Dans ce processus tous les modèles de plates-formes sont décrits par le profil SRM. De plus, les générateurs des descriptions textuelles OIL et XML sont écrits spécifiquement pour les plates-formes visées[7].Ces générateurs ne possèdent aucunes informations métiers. Tous les choix d'instanciation des ressources sont spécifiés dans les modèles en amont[8].

Bien que les générateurs de code soient spécifiques, les transformations de modèles de la première et la deuxième étape (transformations utilisées

[6]Dans un exécutif OSEK/VDX-OS les ressources sont instanciées statiquement, c'est-à-dire au démarrage de l'application. Il n'y a pas de création dynamique de ressources pendant l'exécution de l'application.

[7]La description de la syntaxe des mécanismes et des services dans les modèles de plates-formes permettrait, à priori, de concevoir des générateurs de code génériques paramétrés par la syntaxe des ressources et des services. Par exemple, un générateur XML générique pourrait être paramétré par le modèle de la plate-forme CHEDDAR pour produire un fichier XML manipulable par l'outil.

[8]Dans la version de la chaine de compilation OIL utilisée, il n'y a pas possibilité de décrire des modes d'exécution ni plusieurs CPU. Ces deux informations sont donc implantées directement dans les générateurs de code.

pour les portages de HLAM vers OSEK/VDX-OS et de OSEK/VDX-OS vers CHEDDAR) sont capitalisées dans un même composant de transformation. En effet, le métamodèle de plate-forme (SRM) étant utilisé comme un pivot sémantique, il est possible de capitaliser un même composant de transformation réutilisable entre les différentes plates-formes. C'est ce composant, nommée τ_{p2p} (abréviation de $\tau_{platform2platform}$) qui est décrit dans la section suivante.

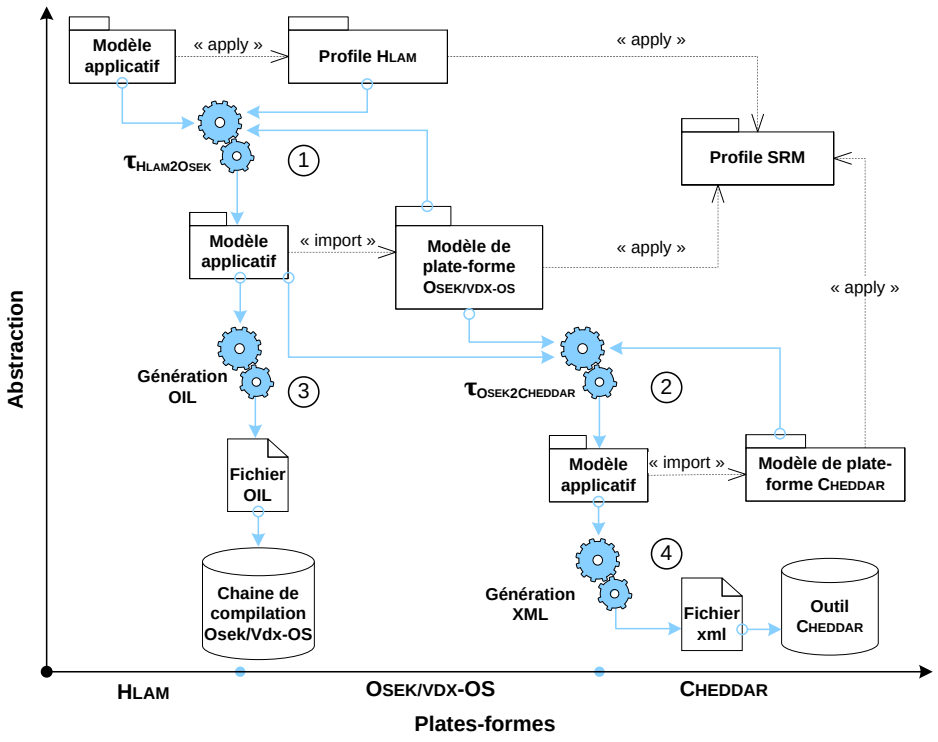

FIG. 7.6 – *Synoptique du processus génératif expérimental*

7.2 DESCRIPTION DE L'INFRASTRUCTURE DE TRANSFORMATION

L'objectif du composant de transformation τ_{p2p} est de porter des applications d'un ensemble de plates-formes sources vers un ensemble de plates-formes cibles. Pour cela, la conception du processus de transformation est tout d'abord décrite. Ensuite l'implantation de ce processus est détaillée.

7.2.1 Conception du composant de transformation τ_{p2p}

Les chapitres 3 page 37 et 4 page 57 ont montrés qu'il existait différents cas de modélisation pour les métamodèles de plates-formes. D'une part, certaines plates-formes sont décrites comme des profils UML et d'autres comme des modèles UML. D'autre part, un ou plusieurs éléments du modèle de la plate-forme peuvent être décrits par un ou plusieurs éléments du métamodèle. Réciproquement, un ou plusieurs éléments du métamo-

dèle peuvent être utilisés pour décrire un ou plusieurs éléments de la plate-forme. Il est donc illusoire de vouloir traiter tous ces cas de modélisation dans une même transformation monolithique. La diversité des types de plates-formes et des possibilités de modélisation (même restreintes) impose la mise en œuvre de plusieurs transformations unitaires chainées. Ces transformations unitaires transforment successivement le modèle applicatif d'entrée et les modèles de plates-formes impliqués (sources et cibles) pour faciliter l'écriture de la transformation de portage. Ces transformations sont au nombre de cinq :

1. τ_{inj} : une transformation d'injection qui explicite l'application. Elle transforme un modèle applicatif source implicitement implanté sur une ou plusieurs plates-formes décrites au niveau méta en une application cible, implantée explicitement sur une ou plusieurs plates-formes décrites au niveau modèle. Dans le cadre de cette étude, le modèle de cette plate-forme est fourni par l'utilisateur (le modèle utilisé pour concevoir le profil UML),
2. $\tau_{\int}$: une transformation d'intégration qui réduit et concatène les éléments des modèles sources et cibles. Cette transformation supprime tous les artéfacts de modélisation liés à la factorisation des modélisations. Elle supprime les héritages, les associations et intègre les motifs de conception comme des ressources explicites intrinsèques aux modèles de plates-formes,
3. $\tau_{\rightarrow}$: une transformation de portage qui transforme le modèle applicatif spécifique aux plates-formes sources en un modèle spécifique aux plates-formes cibles,
4. $\tau_{\dot{x}}$: une transformation de dérivation qui inverse l'opération d'intégration $\tau_{\int}$. Elle transforme les modèles applicatifs produits par la transformation $\tau_{\rightarrow}$ pour les rendre spécifiques aux modèles de plates-formes fournis par l'utilisateur (et non au modèle de plates-formes intégrés par la transformation $\tau_{\int}$),
5. τ_{extrac} : une transformation d'extraction qui inverse l'opération d'injection. Elle transforme le modèle applicatif pour rendre implicite les plates-formes décrites par des profils UML (Ces plates-formes ont été explicitées dans la transformation τ_{inj}).

Les transformations d'injection et d'intégration constituent le ***front end*** du composant de transformation τ_{p2p}. Les composants de dérivation et d'extraction constituent le ***back end***. La synoptique de ce composant est représentée dans la figure 7.7 page suivante .

7.2.2 Implantation du composant de transformation τ_{p2p}

L'implantation du composant τ_{p2p} consiste à décrire chacun des composants de transformation qui le compose. Ces composants sont τ_{inj}, $\tau_{\int}$,$\tau_{\rightarrow}$,$\tau_{\dot{x}}$ et τ_{extrac}.

► **La transformation d'injection : τ_{inj} –** La figure 7.8 page 123 présente la synoptique du composant d'injection.

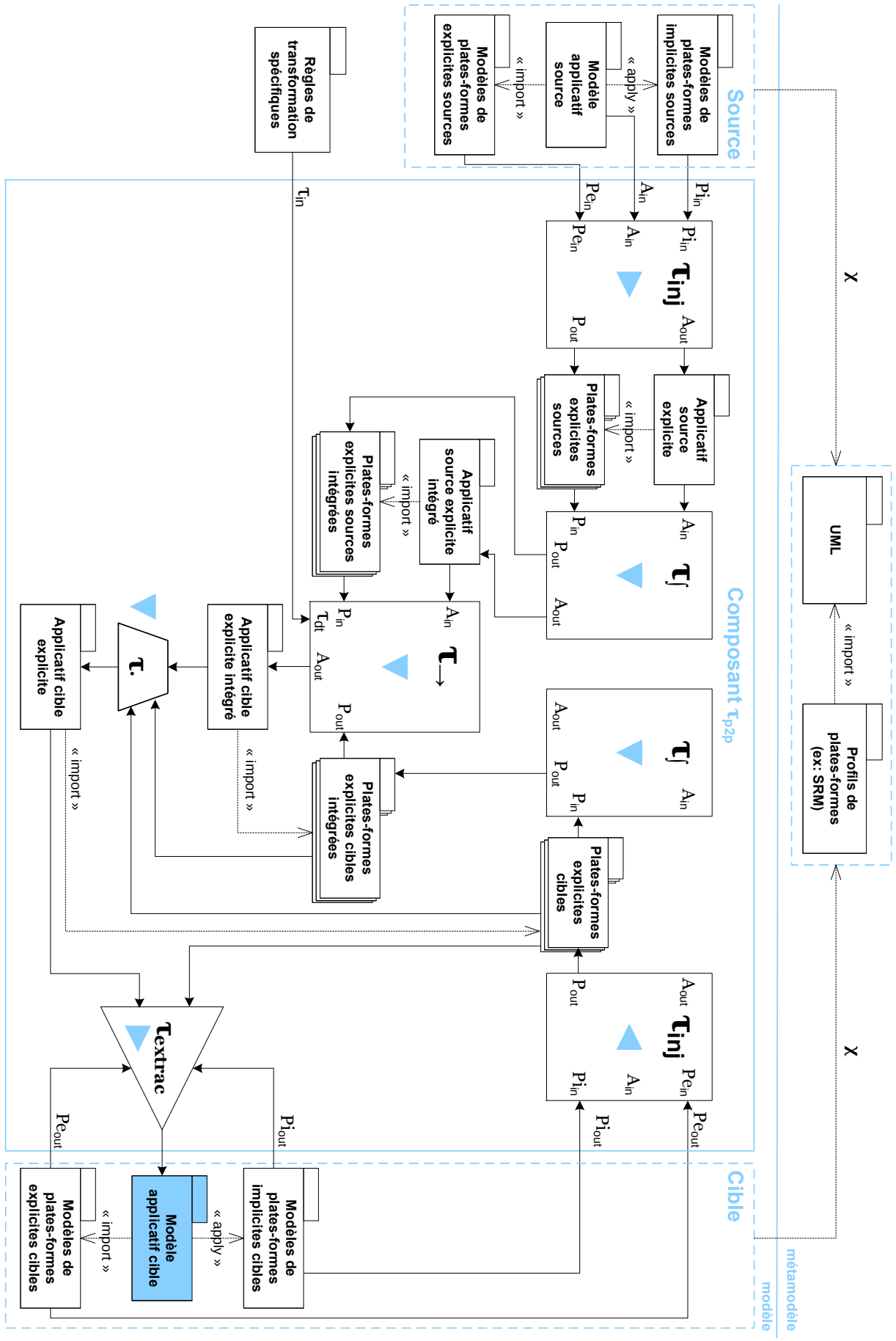

FIG. 7.7 – *Synoptique du composant de transformation* τ_{p2p}

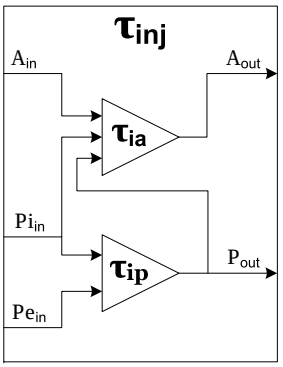

FIG. *7.8 – Synoptique du composant d'injection*

Le composant d'injection est composé de deux transformations :

1) τ_{ip} est la transformation d'injection des modèles de plates-formes. Elle transforme les plates-formes décrites sous la forme de profils UML en des modèles UML. Elle est la transformation inverse de la transformation de promotion ($\tau_{ip} = \tau_p^{-1}$) qui a permit d'obtenir ces profils. Dans cette étude, cette transformation est la transformation identité puisque le modèle de domaine est fourni en entrée du composant[9].
2) τ_{ia} est la transformation d'injection des modèles applicatifs. Elle rend explicite toutes les plates-formes d'exécution impliquées dans les modèles applicatifs. Son l'algorithme est détaillé en 7.1 page suivante. Il consiste à créer une instance pour chaque élément stéréotypé. Cette instance est typée par la ressource explicite produite en sortie de τ_{ip}. Un résultat de cette transformation est proposé dans la figure 7.11 page 129 . Elle concerne l'injection du modèle applicatif spécifique à la plate-forme HLAM (portage HLAM2OSEK) Les opérations updateControl et compute sont transformées en des instances de la ressource explicite RtService. Les valeurs affectées à ces instances sont déduites des méta-valeurs possédées par les stéréotypes.

► **La transformation d'intégration : $\tau_{\int}$ –** La synoptique du composant d'intégration est présentée dans la figure 7.9 page suivante .

L'objectif de l'intégrateur est de concaténer les différents artéfacts de modélisation dans un modèle simplifié pour ensuite plus facilement les manipuler en entré de la transformation de portage. Les modèles de sortie de cette intégration sont alors d'une part des modèles de nouvelles plates-formes (abstraites) et d'autre part des modèles applicatifs spécifiques à ces nouvelles plates-formes (abstraites). Comme le montre la figure 7.9 page suivante , l'intégrateur est constitué de deux transformations :

1. $\tau_{\int p}$ intègre les éléments des modèles de plates-formes. Elle intègre les motifs de conception associés aux modèles de plates-formes

[9]Dans les cas ou la le modèle de domaine n'est pas fourni en entrée de la transformation, des informations de traçabilité issues de la transformation de promotion peuvent être utilisées. Lagarde *et al.* [57] produit par exemple ces informations dans son outil d'aide à la conception des profils UML.

Algorithme 7.1 : Algorithme de la transformation τ_{ia} dédiée à l'injection les modèles applicatifs

Entrées :
A_{in} : Le modèle applicatif source
Pi_{in} : Le profil de la plate-forme implicite
P_{out} : L'ensemble des modèles explicites de sorties

Résultat :
A_{out} : Le modèle applicatif de sortie

Données :
$S(e)$: L'ensemble des stéréotypes décrivant l'élément e

```
1  début
2      Transformer le modèle A_in en un modèle A_out;
3      pour chaque element ∈ A_in faire
4          si S(element) != ∅ alors
5              Créer une InstanceSpecification nommée par le nom de la
               classe ;
6              Typer cette instance par la classe issue de la
               transformation τ_ip ;
7              Créer un slot pour chaque propriété du stéréotype
               possédant une valeur ;
8              Résoudre les dépendances ;
9          sinon
10             Recopier l'élément.;
11         Ranger l'élément créé dans A_out;
12 fin
```

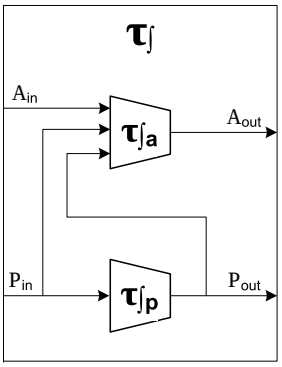

FIG. *7.9 – Synoptique du composant d'intégration*

comme des ressources explicites de ces modèles de plates-formes. Elle transforme les associations par des références (les associations n'apportent aucune sémantique supplémentaire dans ces modèles). Elle supprime les héritages en dupliquant les informations (les héritages complexifient les transformations). Enfin, elle supprime tous les éléments du modèle qui ne sont pas identifiés par les métamodèles de plates-formes, excepté les types de données (DataType, PrimitiveType et Enumeration). Les informations non décrites explicitement par les métamodèles de plates-formes ne peuvent être traitées automatiquement par des outils de transformations. La figure 7.14 page 131 illustre le résultat de l'intégration de la plate-forme OSEK/VDX-Os décrite dans la figure 7.3 page 117 . Dans cet exemple, la classe OsekTask a disparu puisqu'elle n'était pas décrite par le profil SRM. De même les relations d'héritages et d'associations ont été supprimées. Une nouvelle ressource a été ajoutée pour décrire des tâches étendues périodiques : PeriodicOsekTask.

2. $\tau_{\int a}$, intègre les instances des modèles applicatifs. Elle transforme le modèle applicatif pour le rendre spécifique au modèle de la plate-forme généré par $\tau_{\int p}$. Elle recopie les instances dont les types sont des classes, des éléments typés ou des types primitifs. Enfin, lorsque des collaborations sont présentes dans le modèle de plate-forme source, elle détecte les instances collaborant conformément à ces collaborations puis les intègre dans une instance unique. Cette détection est réalisée par une librairie spécifique codée pour les besoins de cette étude[10]. Un exemple d'utilisation de la transformation $\tau_{\int a}$ est décrite dans la figure 7.15 page 132 . Cet exemple est issu du portage de OSEK/VDX-Os vers CHEDDAR. Dans ce portage, certaines instances du modèle applicatif (OSEK/VDX-OS) collaborent pour réaliser des tâches périodiques. Ce sont ces instances qu'il faut détecter et intégrer. Dans cet exemple, les quatre instances task_updateControl, action_updateControl, alarm_updateControl et 50msTimer_updateControl sont détectées comme des instances collaborant conformément à la collaboration PeriodicOsekTask. Elles sont donc concaténées en une instance updateControl typée par la ressource PeriodicOsekTask (créée par la transformation $\tau_{\int p}$).

Si l'intégration des motifs dans les modèles de plates-formes n'est pas un problème, la détection et la concaténation des instances dans le modèle applicatif cible est difficile à mettre en œuvre. Elle nécessite de traduire la collaboration en un ensemble de contrainte, puis d'explorer le modèle applicatif source pour rechercher les instances satisfaisant l'ensemble des contraintes. Ces contraintes sont classées en trois grandes familles :

1. les contraintes génériques au modèle UML, c'est-à-dire les contraintes invariantes qui ne sont pas dépendantes du motif lui-même mais génériques au métamodèle UML. Ce sont typiquement les contraintes de types, de cardinalités et de référencement. Par exemple, les instances dont le type n'est pas référencé dans la collaboration ne sont pas de bonnes candidates. Ces contraintes sont

[10]Cette librairie est inspirée des algorithmes de la théorie de la satisfaction de contraintes décrite dans [84].

génériques au métamodèle UML. Elles sont codées explicitement dans la librairie.

2. Les contraintes dynamiques, c'est-à-dire les contraintes spécifiques au motif. Par exemple, dans la figure 7.15 page 132 , une instance susceptible de collaborer est une instance typée ExtendedTask référençant une et une seule Action. Ces contraintes sont automatiquement induites par la librairie au cours de l'exécution de la transformation.
3. Les contraintes utilisateurs, c'est-à-dire les contraintes écrites explicitement par l'utilisateur sous la forme d'invariants attachés à la collaboration[11].

Ces contraintes sont détaillées dans la description de la librairie en annexe E page 169.

► **La transformation de portage : $\tau_{\rightarrow}$ –** La synoptique du composant de portage est représentée dans la figure 7.10 .

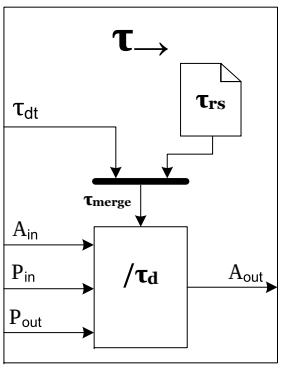

FIG. *7.10 – Synoptique du composant de portage*

La transformation de portage vise à transformer un modèle applicatif spécifique à une ou plusieurs plates-formes sources en un modèle applicatif spécifique à une ou plusieurs plates-formes cibles. Plus particulièrement, l'objectif est de transformer les instances spécifiques aux ressources sources en des instances spécifiques aux ressources cibles. Le choix du portage est opéré par équivalence. A partir d'une instance source, l'algorithme recherche pour chaque ressource typant cette instance, une ressource cible équivalente. Si celle-ci existe, il crée une instance. Cet algorithme est décrit en 7.2 page ci-contre. Il s'appuie sur l'opérateur $\approx$ défini dans le chapitre 5. Un algorithme similaire traite les propriétés de ces ressources. Pour chaque propriété ayant une valeur dans l'instance source, une valeur est affectée à la ou les propriétés référencées de la même façon dans la ressource cible.

[11] Dans l'expérimentation de cette étude, un pilote spécifique au moteur ATL a été développé pour lui permettre de vérifier des contraintes OCL au cours de l'exécution de la transformation. Ce pilote délègue le traitement de la contrainte OCL au plug-in OCL ECLIPSE. Celui retourne *Vrai* si la contrainte est satisfaite et *Faux* sinon.

Algorithme 7.2 : Algorithme de portage des modèles applicatifs

Entrées :
L'ensemble des instances des modèles applicatifs sources A_{in}
$I_s = \{i_s | \exists a_{in} \in A_{in} : i_s \in E(a_{in}) \wedge i_s \chi \text{ InstanceSpecification}\}$,

R_s, R_c : L'ensemble des ressources sources et cibles

Résultat :
L'ensemble des couples (Instance source, Ensemble des Ressources cibles sélectionnées) $T_{ic} = \{t_{ic} \mid \exists\, i \in I_s, R_{select} \subset R_c : t_{ic} = (i, R_{select})\}$

Données :
R_{select} : Ensemble des ressources sélectionnées

```
1  Initialisations :
2  T_ic ← ∅,
3  R_select ← ∅
4  début
5      pour chaque i_s ∈ I_s faire
6          R_select ← ∅;
           L'ensemble des ressources sources typant i_s
7          R_is = {r_is | r_is ∈ R_s ∧ r_is ∈ i_s.classifier};
8          pour chaque r_is ∈ R_is faire
9              pour chaque r_c ∈ R_c faire
10                 si r_is ≈ r_c alors
11                     R_select ← R_select ∪ {r_c};
12             T_ic ← T_ic ∪ (i_s, R_select);
13     retourner T_ic;
14 fin
```

Bien sur, il est illusoire de considérer que tous les éléments du modèle de sortie peuvent être générés par cette simple relation d'équivalence. Il est utopique de vouloir déterminer une relation d'équivalence automatique entre des types de données ou entre des types primitifs. Par exemple dans le portage d'une application HLAM vers OSEK/VDX-OS (voir figures 7.2 page 116 et 7.3 page 117), il est illusoire de vouloir déterminer automatiquement une équivalence entre une durée (Duration) et un AutoStart. Ce sont tous deux des types de donnée qui ne sont pas identifiés par le profil SRM. C'est pourquoi, un ensemble de règles de transformation (τ_{dt}, abréviation de $\tau_{dataType}$) sont spécifiées par l'utilisateur en entré du composant de transformation. Dans cette étude, ces règles sont limitées aux types de données et aux types primitifs. Elles sont fusionnées aux règles génériques τ_{rs} (abréviation de $\tau_{ressourceService}$) implantant les algorithmes d'équivalences précédents (équivalences entre les ressources et entre les propriétés). Le résultat de cette fusion, la transformation dérivée τ_d, porte effectivement le modèle applicatif source en un modèle applicatif cible.

La figure 7.12 page 130 illustre un résultat obtenu par la transformation $\tau_{\rightarrow}$ pour le premier portage du processus : HLAM vers OSEK/VDX-OS. Dans cet exemple, les plates-formes ont été intégrées dans des plates-formes ABSTRACT_HLAM[12] (voir figure 7.2 page 116) et ABSTRACT_OSEK/VDX-OS (voir figure 7.14 page 131). L'unité temps réel est portée vers l'ordonnanceur de la plate-forme ABSTRACT-OSEK/VDX-OS. Les services temps réel sont transformés en des tâches périodiques PeriodicTask. Les durées Duration sont transformées en des AutoStart (des instances du type de donnée AutoStart). Cette dernière règle est fournie explicitement par l'utilisateur. Elle est détaillée dans l'annexe D.2.1 page 164.

► **La transformation de dérivation : $\tau_{\dot{x}}$ –** La transformation de dérivation vise à transformer le modèle applicatif cible pour inverser l'opération d'intégration appliquée sur le modèle de plate-forme cible. Le modèle applicatif résultant est alors spécifique à la plate-forme cible et non plus à la plate-forme abstraite générée par la transformation d'intégration. Concrètement, la transformation de dérivation instancie chacun des motifs intégrés au modèle de plate-forme et complète les valeurs des instances par les valeurs par défaut spécifiées dans le modèle de plate-forme. Par exemple, la figure 7.13 page 130 montre la dérivation du modèle généré par l'opération de portage précédente (figure 7.12 page 130). Dans cette figure, les instances typées par la ressource PeriodicOsekTask disparaissent. Elles sont remplacées par un ensemble d'instance, une pour chaque partie de la collaboration PeriodicOsekTask (voir figure 7.4 page 118). De même, les valeurs des tailles de piles, stackSize, sont ajoutées puisqu'une valeur par défaut est spécifiée dans le modèle de plate-forme OSEK/VDX-OS (voir figure 7.3 page 117).

► **La transformation d'extraction : $\tau_{extract}$ –** La transformation d'extraction inverse la transformation d'injection. Elle transforme le modèle ap-

[12] L'opération d'intégration ne modifie pas le modèle de HLAM, le modèle de ABSTRACT_HLAM est donc celui de HLAM.

plicatif pour rendre implicite les ressources des plates-formes décrites en entrée du composant τ_{p2p} par des profils UML (les plates-formes implicites). Les instances typées par des ressources implicites sont transformées en des éléments conforment à la méta-classe étendue par le stéréotype [13]. Dans le cas de cette expérimentation, la transformation $\tau_{extract}$ est la fonction identité puisqu'il n'y a pas de plates-formes cibles implicites.

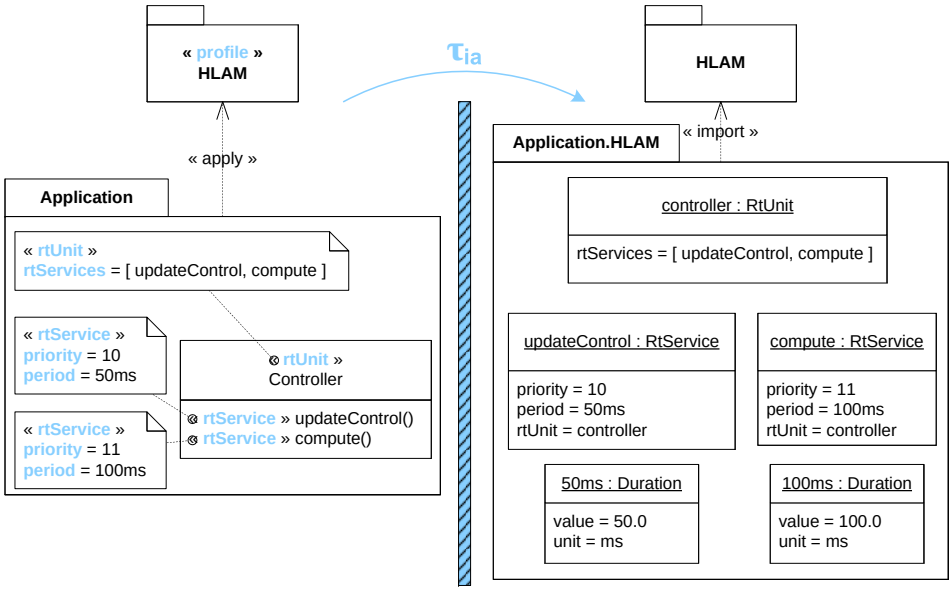

FIG. *7.11 – Injection d'un modèle applicatif spécifique à la plate-forme* HLAM

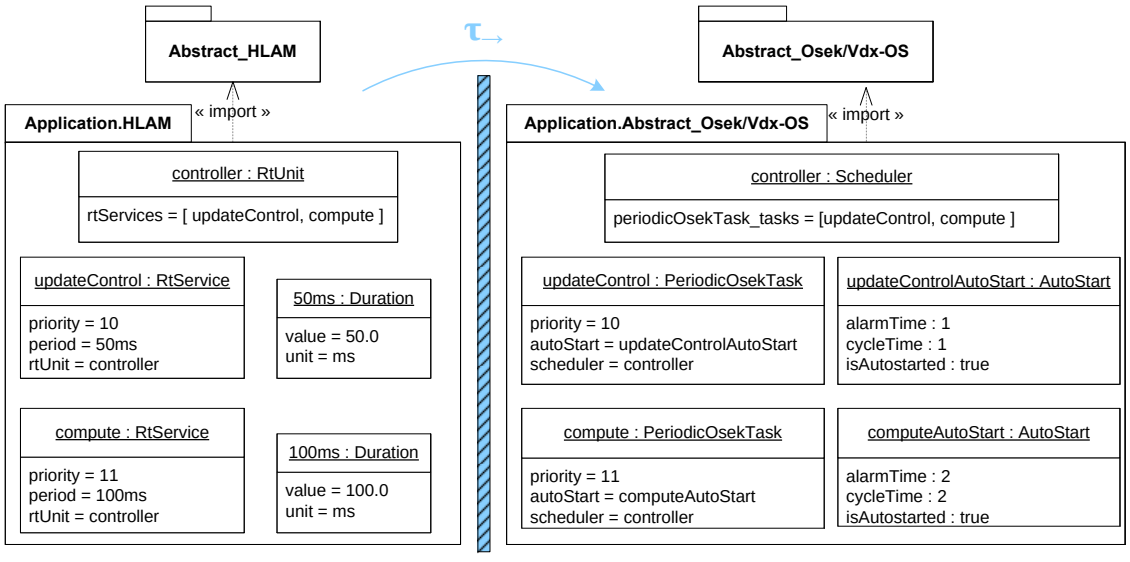

FIG. *7.12 – Génération du modèle applicatif spécifique à la plate-forme abstraite* ABSTRACT_OSEK/VDX-OS

[13] Le composant τ_{p2p} impose qu'il n'y ait par dans la modèle de plate-forme implicite d'extension multiple.

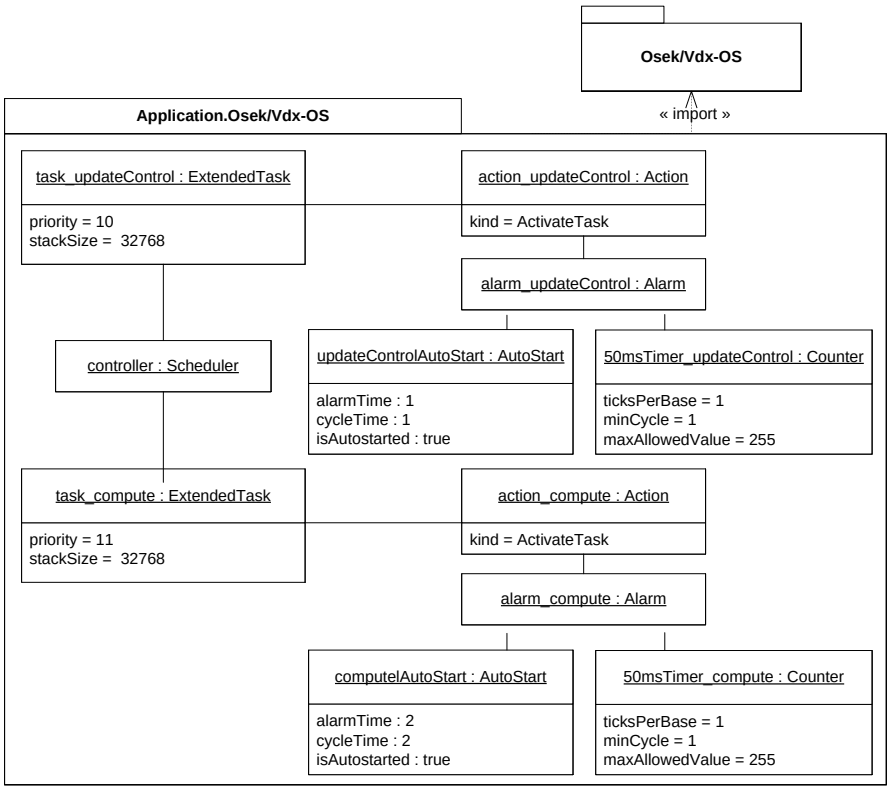

FIG. *7.13 – Dérivation du modèle applicatif* OSEK/VDX-OS

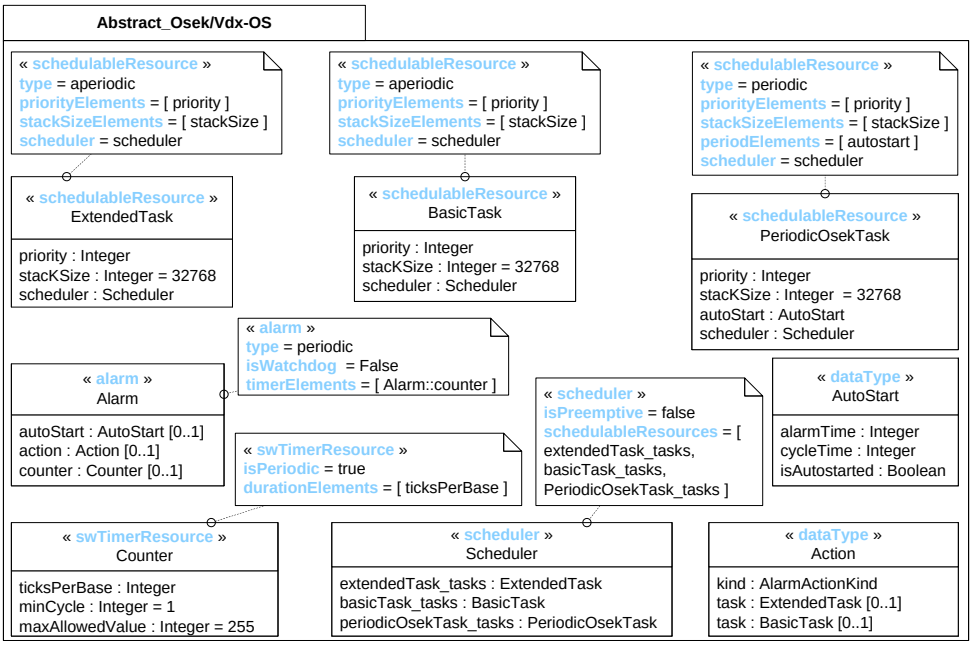

FIG. *7.14 – Intégration du modèle de plate-forme* OSEK/VDX-OS

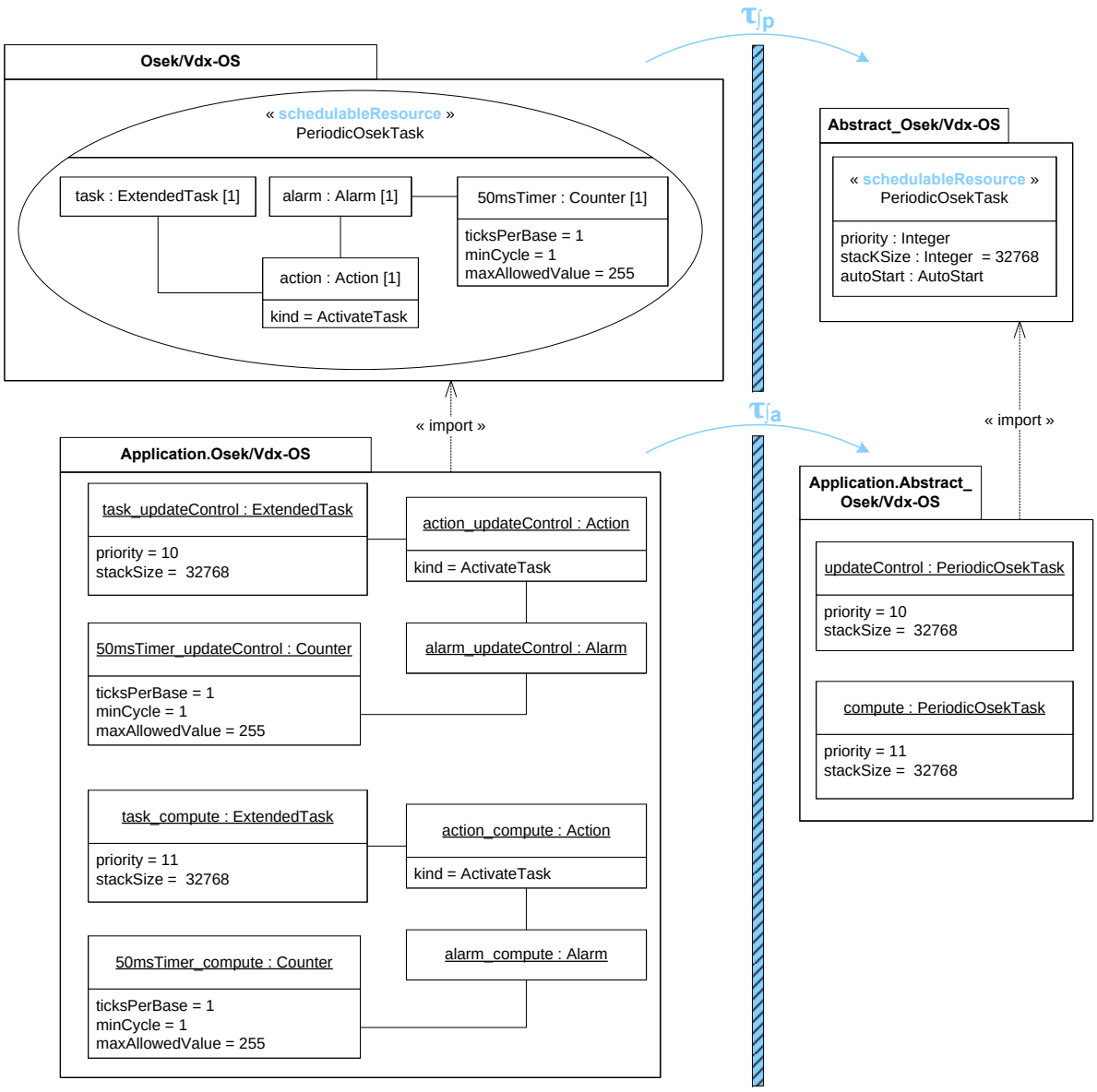

FIG. *7.15 – Intégration d'un modèle applicatif spécifique à la plate-forme* OSEK/VDX-OS

7.3 Évaluation de l'infrastructure de transformation

7.3.1 Réalisation expérimentale

Les différentes transformations du composant τ_{p2p} ont été décrites dans le langage de transformation ATL [24]. Elles sont intégrées dans un ***plugin***[14] Eclipse[15] nommé *P2P*. La figure 7.16 page 133 illustre l'utilisation de l'outil *P2P* dans cette étude.

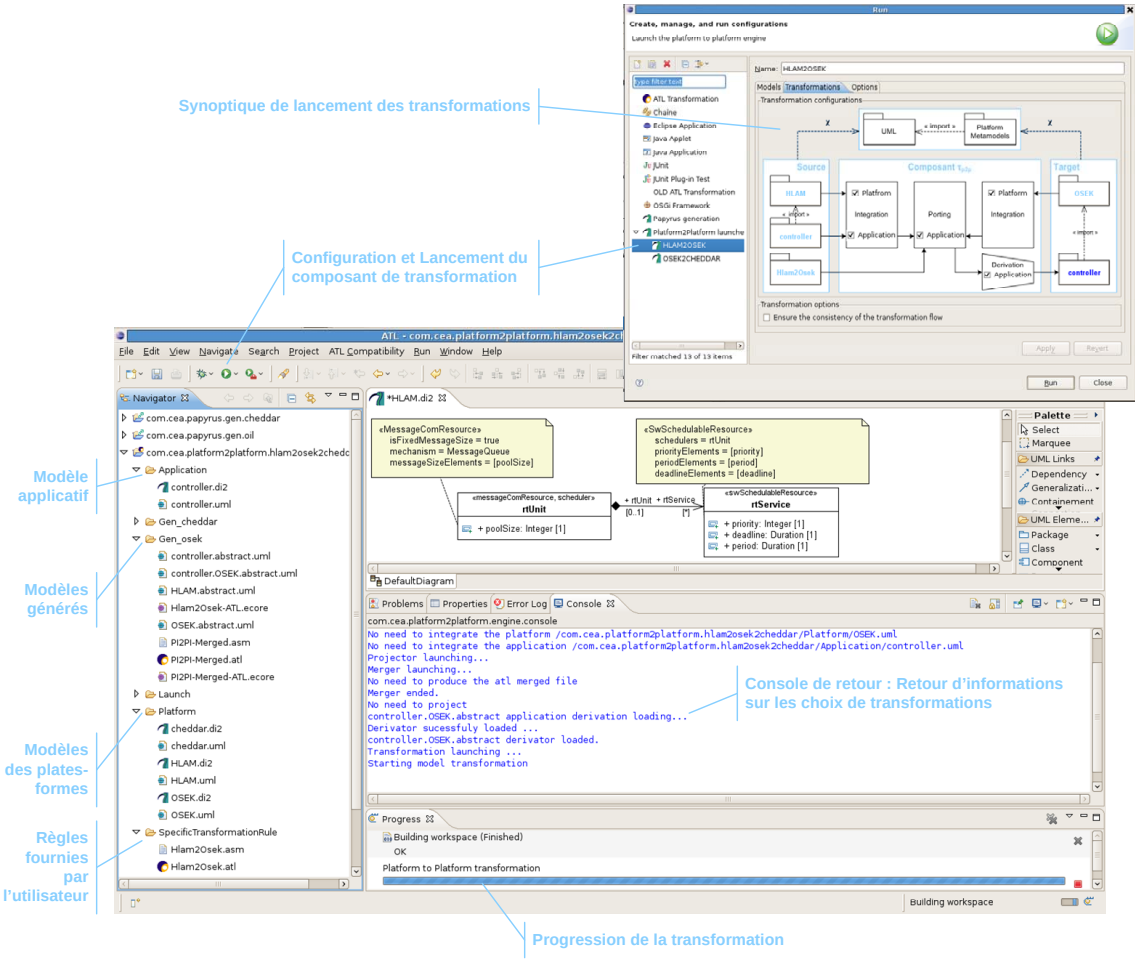

Fig. 7.16 – *Réalisation du composant de transformation τ_{p2p} sous la forme d'une extension* Eclipse

Dans le processus expérimentale de cette étude, cet outil est utilisé à deux reprises. Une première fois dans le portage d'une application spécifique à Hlam vers Osek/Vdx-OS ($\tau_{\text{Hlam2Osek}}$), et une seconde fois dans le portage de l'application spécifique à Osek/Vdx-OS vers Cheddar ($\tau_{\text{Osek2Cheddar}}$). A partir des résultats de ces deux transformations, des générateurs de code OIL et Cheddar-XML ont été écrits en Acceleo[16][33].

[14] Une extension à la plate-forme Eclipse.
[15] www.eclipse.org
[16] www.acceleo.org

Les fichiers générés ont été utilisés pour configurer l'exécutif TRAMPOLINE [60] et l'outil CHEDDAR [83].

7.3.2 Résultats expérimentaux

Les figures 7.17 page 133 et 7.18 page 134 synthétisent les résultats de chacune des transformations $\tau_{\text{HLAM2OSEK}}$ et $\tau_{\text{OSEK2CHEDDAR}}$. La première figure illustre les équivalences entre les ressources et la seconde figure illustre les équivalences entre les propriétés de ces ressources. Un trait noir est une équivalence automatiquement déduite, un trait de couleur est une équivalence spécifiée explicitement par l'utilisateur. en entrée du composant τ_{p2p} (l'entrée τ_{in} de la synoptique 7.7 page 122).

Ces résultats montrent qu'une grande partie des équivalences ont été déduites des descriptions des plates-formes. Par exemple dans la première transformation, seul les règles pour transformer les instances du type de donnée Duration en les instances du type de donnée AutoStart ont été écrites par l'utilisateur. De même dans la seconde transformation, une seule règle a été écrite pour transformer un AutoStart en un Integer et les capacités et les échéances des tâches ont été complétées dans l'outil d'analyse lui même puisque ces données sont spécifiques à l'analyse d'ordonnancement. Les règles de transformation écrites par l'utilisateur sont détaillées dans les annexes D.2.1 page 164 et D.2.2 page 167.

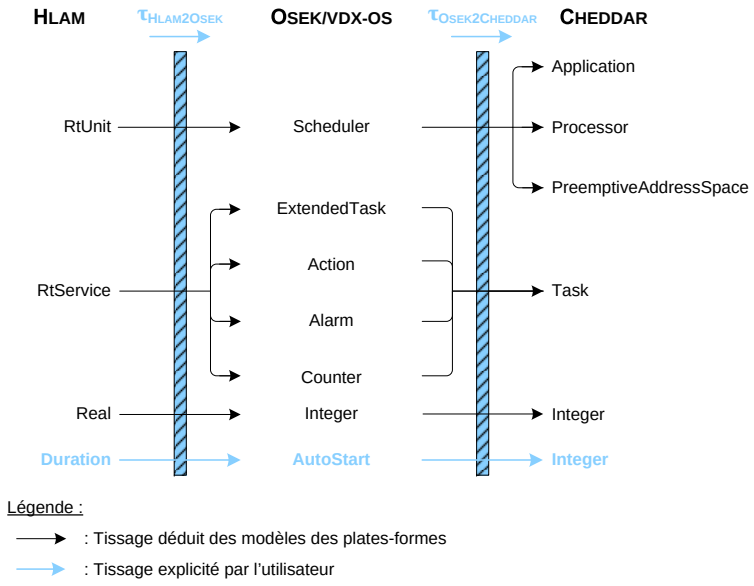

FIG. 7.17 – *Synthèse des résultats expérimentaux de génération pour les ressources*

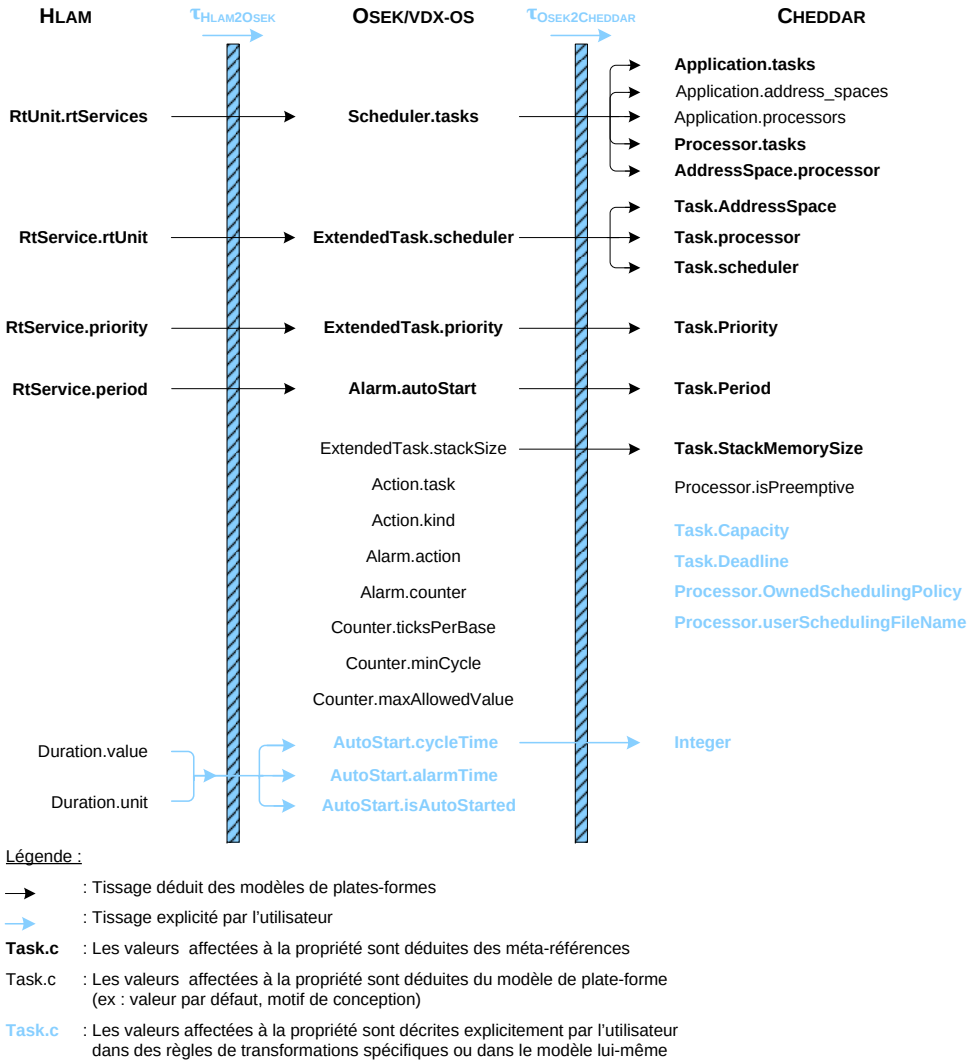

FIG. *7.18 – Synthèse des résultats expérimentaux de génération pour les propriétés*

7.3.3 Conclusion de l'évaluation

L'architecture proposée dans cette expérimentation a été implantée et utilisée dans un processus génératif expérimental. Les premiers résultats sont encourageants puisque, d'une part, ils montrent que l'implantation d'une infrastructure de transformation basée sur des plates-formes existe et que, d'autre part, une infrastructure générique est envisageable pour aider à la construction des transformations entres plates-formes. L'explicitation des plates-formes a permis de capitaliser une grande partie des règles de transformation. Ces règles sont dépendantes :

Capitalisation du noyau de transformation

1. des hypothèses de modélisation de l'application (classes singletons)
2. des hypothèses de connexions entre l'application et la plate-forme. L'architecture est dépendante des connexions par stéréotypes et par typages.
3. du motif Resource-Service, plus particulièrement, des services spécifiés dans le chapitre 4 page 57.

Considérant ces limitations, le composant de transformation est donc générique pour les modèles applicatifs et les modèles de plates-formes respectant ces hypothèses. Les règles capitalisées sont en fait dépendantes du motif Resource-Service. Elles peuvent donc être réutilisées à différentes étapes du cycle de développement, pour différents modèles de plates-formes et pour différents métamodèles de plates-formes appliquant le motif. Elles ne sont d'ailleurs pas impactées par les faiblesses de description des métamodèles de plates-formes. En effet, même si un métamodèle comme SRM ne modélise pas entièrement la plate-forme manipulée, plusieurs métamodèles de plates-formes peuvent être utilisés conjointement. Par exemple le profil Time de MARTE, qui applique le motif Resource-Service, peut être utilisé conjointement avec SRM pour représenter les horloges et les échelles de temps. Dans ce cas, les règles de transformation restent opérationnelles.

7.4 Discussion

Un remède contre la complétude des transformations ?

Même si, le composant τ_{p2p} déduit automatiquement un certain nombre de liens de tissage [85] entre la plate-forme source et la plate-forme cible, il ne pourra jamais les déduire tous. Une solution pour rendre plus flexible l'utilisation d'un tel composant est de générer le modèle de tissage entre les plates-formes sources et cibles. L'utilisateur peut ensuite compléter manuellement ou automatiquement ce modèle. Ce modèle de tissage est ensuite utilisé pour paramétrer une transformation d'ordre supérieur qui génère, non pas l'application cible, mais la transformation spécifique aux plates-formes sources et cibles (voir figure 5.2 page 78). Cette solution permet à l'utilisateur de spécifier des règles de transformations spécifiques qui ne concernent pas forcément les types de données et les types primitifs (limitation du composant τ_{p2p} actuellement implanté). Dans cette utilisation, le composant τ_{p2p} est un assistant d'aide à l'écriture des transformations.

Comment ajouter de la flexibilité au composant de transformation τ_{p2p} ?

En ce qui concerne la génération, il n'existe, malheureusement à l'heure actuelle, aucun artéfact permettant d'optimiser les liens que le composant de transformation déduit automatiquement. Par exemple, lors de la génération de l'application Osek/Vdx-OS à partir de l'application Hlam (voir figure 7.12 page 130), un compteur unique peut être utilisé pour les deux alarmes. Dans l'état actuel, le composant τ_{p2p} génère un compteur par alarme. Le composant τ_{p2p} est en fait limité par le pouvoir d'expression des collaborations UML. La définition d'un profil UML pour guider les transformations peut solutionner ce problème. Par exemple, un stéréotype Unique appliqué sur la partie 50msTimer de la collaboration PeriodicOsekThread peut spécifier le partage d'un même compteur pour toutes les instanciations du motif PeriodicOsekThread. De même, un stéréotype Singleton sur l'ordonnanceur de la plate-forme Osek/Vdx-OS peut éviter la génération de plusieurs instances de cet ordonnanceur. Une plate-forme Osek/Vdx-OS ne possède en effet qu'un seul ordonnanceur.

Les transformations sont-elles toujours déterministes ?

Enfin, une hypothèse forte du portage définit dans le composant τ_{p2p} est qu'il n'y ait pas plusieurs éléments stéréotypés de la même façon dans les modèles des plates-formes sources et cibles. Dans le cas contraire, le portage n'est plus déterministe. Un choix est en effet nécessaire pour savoir quelle ressource doit être effectivement instanciée. Pour pallier cette limitation un composant supplémentaire peut être ajouté au composant τ_{p2p}. Ce composant vise à optimiser les modèles de plates-formes sources et cibles pour un contexte donné. Concrètement, il sélectionne une ressource parmi l'ensemble des ressources décrites par le même stéréotype. Cette sélection s'effectue, par exemple, en analysant des propriétés non fonctionnelles telles que la taille mémoire ou les temps d'exécution par exemple.

Conclusion

Dans ce chapitre, une infrastructure de transformation a été définit. Celle-ci vise à porter un modèle spécifique à un ensemble de plates-formes sources en un modèle spécifique à un ensemble de plates-formes cibles. Cette architecture de transformation est endogène. Les modèles de plates-formes sources et cibles sont en effet décris par les mêmes métamodèles de plates-formes. Cette architecture n'est cependant pas spécifique à ces métamodèles de plates-formes mais générique au motif Resource-Service. Elle peut être utilisée pour n'importe quel métamodèle appliquant le motif. Ainsi, dans ce chapitre, elle a été utilisée à deux reprises pour produire des modèles d'applications multitâches spécifiques à différentes plates-formes logicielles d'exécution. Ces plates-formes étaient alors décrites par le profil SRM.

Cette étude a donc montré qu'une architecture générique basée sur des modèles de plates-formes était faisable et utilisable. Elle a mis en avant l'importance de l'explicitation des plates-formes en capitalisant l'architecture de transformation manipulant ces plates-formes. C'est pourquoi, cette architecture a été publiée dans la conférence internationale ISORC 2008 [81].

Conclusion

Bilan

Rappel de la problématique

Ce travail avait pour objectif d'étudier la prise en compte des plates-formes logicielles d'exécution dans une ingénierie générative dirigée par les modèles. Plus particulièrement, il devait contribuer à : a) définir ce qu'est un modèle de plate-forme au sens de l'IDM, b) définir ce qu'est un métamodèle de plate-forme au sens de l'IDM et c) intégrer ces descriptions dans une infrastructure générative dirigée par les modèles.

Démarche suivie

Pour cela, le contexte applicatif a été centré sur les systèmes logiciels multitâches, c'est-à-dire sur les systèmes applicatifs s'exécutant sur des plates-formes logicielles d'exécution multitâche (chapitre 1 page 7). Ensuite, la démarche d'étude a été agencée en trois parties.

Premièrement, un état de l'art sur la modélisation des plates-formes logicielles d'exécution multitâche a été dressé (chapitre 2 page 23). Il est apparu que les concepts proposés dans la littérature sont très abstraits. Ils sont souvent distants des mécanismes et des services réellement fournis par les plates-formes. Par conséquent, la deuxième partie de cette thèse a été orientée sur la métamodèlisation des plates-formes.

La seconde partie de cette étude s'est attachée à définir les briques élémentaires d'un métamodèle de plate-forme. Elle a défini le motif Resource-Service. Ce motif est une bonne pratique de métamodèlisation (chapitre 3 page 37) puisqu'il permet, d'une part, de réduire le nombre de métatypes et, d'autre part, de structurer efficacement un métamodèle de plate-forme pour effectivement décrire des modèles de plates-formes d'exécution. Cette bonne pratique a d'ailleurs été confrontée, avec succès, aux différentes possibilités de modélisation (chapitre 4 page 57) des plates-formes. Elle a aussi été outillée pour permettre son utilisation dans des ingénieries opérationnelles (chapitre 5 page 75). Tous ces résultats sont alors génériques à la métamodèlisation des plates-formes. Ils ont été appliqués dans la troisième partie.

La troisième, et dernière, partie s'est attachée à réaliser un métamodèle de plate-forme multitâche et à intégrer ce métamodèle dans un processus de transformation outillé. Pour cela, le profil UML Softawre Resource Modeling a été décrit et évalué pour la modélisation des plates-formes logicielles d'exécution multitâche (chapitre 6 page 91). Son évaluation sur différentes plates-formes réelles a certes montré qu'il n'est pas complet, mais, qu'il permet d'ores et déjà d'expérimenter la réalisation d'une architecture de transformation générique. Cette architecture a été implantée, utilisée et évaluée dans un processus génératif expérimental (chapitre

7 page 113). Elle permet de porter les applications spécifiques à un ensemble de plates-formes sources sur un ensemble de plates-formes cibles. Toutes ces plates-formes sont alors décrites par un même ensemble de métamodèles de plates-formes appliquant le motif Resource-Service. L'architecture de transformation est ainsi générique à tous les métamodèles appliquant ce motif et donc réutilisable à différentes étapes de développement.

Synthèse des contributions

En somme, cette étude a contribué aux problématiques liées à la modélisation et à l'intégration des plates-formes d'exécution. Elle a contribué à :

- **La définition du motif Resource-Service** pour décrire des modèles de plates-formes d'exécution. Un modèle de plate-forme d'exécution est alors un modèle de son API, c'est-à-dire un modèle de la signature, de la sémantique et du comportement des concepts de cette API. Un métamodèle de plate-forme d'exécution est alors un modèle capable de décrire ces APIs. C'est un modèle appliquant le motif Resource-Service. Cette étude s'est d'ailleurs limitée à la description de la signature et de la sémantique de ces APIs,
- **La définition de l'extension UML** Software Resource Modeling (SRM) dédiée à la modélisation des plates-formes logicielles d'exécution multitâche. Cette extension permet plus particulièrement la description de la concurrence d'exécution, des interactions entre ces exécutions et de la gestion des ressources logicielles et matérielles d'exécution,
- **L'expérimentation d'une architecture de transformation**, nommée τ_{p2p}, dont l'exécution est dirigée par des modèles de plates-formes. Ce noyau de transformation reste générique à tout métamodèle appliquant le motif Resource-Service. Il peut être utilisé à différentes étapes de développement et ainsi capitaliser les transformations au cours du cycle de développement.

Discussion générale

L'IDM est aujourd'hui essentiellement centré sur le couple (modèle, métamodèle). Ce couple constitue le cadre architectural à partir duquel les méthodologies et les outils sont développés. Cette architecture en couches (une pour les modèles et une seconde pour les métamodèles) permet de capitaliser les transformations autours des métamodèles. Désormais, l'explicitation des plates-formes impose aussi le couple (application, plate-forme). Les outils développés sont alors capitalisés autour des métamodèles applicatifs et des métamodèles de plates-formes. Ils sont utilisables quelque soit les modèles applicatifs et quelque soit les modèles de plates-formes. Ce concept de plate-forme permet ainsi de capitaliser des transformations tout en assurant une plus grande flexibilité. Il contribue à définir des ***transformations schizophrènes*** capables de cibler plusieurs plates-formes logicielles d'exécution.

Néanmoins, cette capitalisation est dépendante des relations qu'il existe entre une application et une plate-forme. Contrairement au couple (modèle, métamodèle) ou seule la relation de conformité est possible, les relations entre une application et une plate-forme sont nombreuses (héritage, référencement, typage et conformité). Le cadre architectural basé sur des modèles de plates-formes doit donc inévitablement être complété par une méthodologie restreignant ces relations. Les outils de transformations seront alors dépendants de cette méthodologie. Ce point dur n'est pas rédhibitoire. Il alimente les perspectives de cette thèse.

Perspectives

La poursuite de ce travail nécessite, tout au moins, deux types d'études : a) les études à court terme pérennisant le travail effectué et b) les études à long terme enrichissant le domaine de solutions. Pour chacune de ces études, trois axes principaux sont proposés.

Perspectives à court terme

Les études à court terme concernent essentiellement le profil SRM et l'infrastructure de transformation expérimentée dans cette thèse. Elles doivent :

a) enrichir le profil SRM pour la modélisation des mécanismes et des services liés à a) la tolérance aux fautes (par exemple, gestion des modes de fonctionnement et des erreurs), b) la gestion du temps, et c) la gestion des architectures multiprocesseurs et des architectures distribuées. Cet enrichissement peut se concrétiser par une extension lourde du profil ou par la définition de profils spécifiques utilisés conjointement avec le profil SRM,

b) enrichir le prototype de transformation pour gérer les autres cas de modélisation des plates-formes tels que l'utilisation des expressions par exemple et ainsi poursuivre l'expérimentation d'une transformation générique basée sur des modèles de plates-formes explicites,

c) générer le modèle de tissage entre les plates-formes sources et cibles par une transformation d'ordre supérieur et ainsi assister la fusion des règles de transformation spécifiques à l'utilisateur avec celles du noyau générique de transformation.

Perspectives à long terme

Les études à long terme concernent la modélisation et l'intégration des plates-formes dans l'IDM. Elles doivent par exemple :

a) étendre les langages de transformation pour manipuler nativement des métamodèles de plates-formes et ainsi faciliter l'utilisation des modèles de plates-formes explicites dans l'IDM,

b) étudier la modélisation du comportement des plates-formes. L'intégration de cette modélisation dans un processus génératif doit ainsi enrichir l'utilisation des modèles de plates-formes pour la génération, pour l'analyse ou encore pour la simulation par exemple,

c) définir une méthodologie outillée basée sur des modèles de plates-formes. Une extension de la méthodologie ACCORD|UML [40] permettrait ainsi de cibler plusieurs exécutifs multitâches par génération.

Toutes ces études contribueront à satisfaire les besoins d'une modélisation et d'une intégration des plates-formes d'exécution dans une ingénierie dédiée aux applications temps réel embarquées.

Annexes

Librairies ATL dédiées à la manipulation de modèles de plates-formes

A

Ce chapitre est une illustration d'une implantation des ensembles et des algorithmes spécifiés dans la section 5.2 page 79. Il décrit les services d'une librairie ATL [24]. Cette librairie est dédiée à la manipulation de modèles de plates-formes. Ces modèles sont conformes à des métamodèles appliquant le motif Resource-Service (motif décrit dans le chapitre 3 page 37.

Pour cela, ce chapitre est constitué de deux sections. La première illustre une implantation des ensembles mathématiques définis dans la section 5.2 page 79. La seconde illustre une implantation de l'algorithme d'équivalence spécifié en 5.2 page 85.

A.1 Implantation des services primitifs

```
context UML::Model
  def: getAllResourcesAs(name : String) : Sequence(UML::↘
     Type) =
    self.getAllType()->select(e|e.hasStereotype(name))

context UML::Class
  def : getAllPropertiesReferencedAs(tagName : String, ↘
     resourceName : String) : Sequence(UML::Property) =
    self.getValue(self.getAppliedStereotype(resourceName),↘
       tagName)
     ->select(e|e.oclIsKindOf(UML::Property))

context UML::Class
  def : getAllOperationsReferencedAs(tagName : String, ↘
     resourceName : String) : Sequence(UML::Operation) =
    self.getValue(self.getAppliedStereotype(resourceName),↘
       tagName)
     ->select(e|e.oclIsKindOf(UML::Operation))

context UML::Operation
  def : getAllParametersReferencedAs(stereotypedResource: ↘
     UML::Type, tagName : String, resourceName : String) :↘
      Sequence(UML::Parameter) =
```

```
    stereotypedResource.getValue(stereotypedResource.↘
       getAppliedStereotype(resourceName),tagName)
      ->select(e|e.oclIsKindOf(UML::Parameter))
      ->select(e|self.ownedParameter->includes(e))
```

Code A.1 – Librairie ATL *dédiée à la manipulation de modèle de plate-forme*

A.2 Implantation d'une librairie d'équivalence

```
library EquivalenceLib; -- Library Template

uses UMLStereotypeLib;
uses UMLElementLib;
--limitation link to the Traceability profile

helper context UML!Element def : isSourceAndTargetMatched(↘
   stereotyedTarget : UML!Element, src_stereotype : UML!↘
   Stereotype, dest_stereotype : UML!Stereotype) : Boolean↘
    =
  let tags : Sequence(UML!Property) = src_stereotype.↘
     getPrimitiveTypeAndEnumerationProperty().debug('Tags ↘
     evaluated to be able to match')in
  tags->iterate(tag;
        accBoolean : Boolean = true
        | accBoolean and
        if tag.type.oclIsKindOf(UML!Enumeration) then
          self.getTagValue(src_stereotype,tag.debug('Tag')↘
             ).debug('Source tag value').name = ↘
             stereotyedTarget.getTagValue(src_stereotype,↘
             tag).debug('Target tag value').name
        else
          self.getTagValue(src_stereotype,tag.debug('Tag')↘
             ).debug('Source tag value')=stereotyedTarget.↘
             getTagValue(src_stereotype,tag).debug('Target↘
              tag value')
        endif
        ).debug('Tags matched ?');

helper context UML!Element def : ↘
   isAllSourceAndAllTargetMatched(stereotypedTarget : UML!↘
   Element) : Boolean =
  let source_stereotypes : Set(UML!Stereotype) = self.↘
     getAppliedStereotypeSet()->reject(e|e.name = ↘
     thisModule.traceabilityStereotype.name) in
  let target_stereotypes : Set(UML!Stereotype) = ↘
     stereotypedTarget.getAppliedStereotypeSet()->reject(e↘
     |e.name = thisModule.traceabilityStereotype.name) in
  source_stereotypes->iterate(src_stereo;
                accBoolean : Boolean = true
                |accBoolean and target_stereotypes->↘
                   iterate(dest_stereo;
                                   accBoolean : Boolean↘
                                       = true
                                   |accBoolean and self↘
                                      .↘
                                      isSourceAndTargetMatched↘
```

```
                                        (↘
                                        stereotypedTarget↘
                                        ,src_stereo,↘
                                        dest_stereo)
                                    )

                );
```

```
helper context UML!Element def : isStereotypedInASameWay(↘
    target :UML!Element) : Boolean =
  let source_stereotypes : Set(UML!Stereotype) = self.↘
      getAppliedStereotypeSet()->reject(e|e.name = ↘
      thisModule.traceabilityStereotype.name) in
  if source_stereotypes->isEmpty() or target.↘
      getAppliedStereotypeSet()->reject(e|e.name = ↘
      thisModule.traceabilityStereotype.name).isEmpty() ↘
      then
    false
  else
    source_stereotypes->iterate(stereotype;
                accBoolean : Boolean = true
                |accBoolean and
                        if target.isStereotypeAppliedOn(↘
                            stereotype) then
                          true
                        else
                          false
                        endif
                )
  endif;
```

```
helper context UML!Element def : isEquivalentTo(target : ↘
    UML!Element) : Boolean =
  if self.isStereotypedInASameWay(target) then
    self.isAllSourceAndAllTargetMatched(target)
  else
    false
  endif;
```

Code A.2 – Librairie ATL *dédiée à l'implantation de l'opérateur d'équivalence (≈) spécifique au motif* Resource-Service

Ontologie des services des systèmes multitâches

B

L'objectif de ce chapitre est d'illustrer une ontologie des concepts et des services fournis par les systèmes d'exécution multitâche. Cette ontologie a été réalisée pour concevoir le profil UML SRM (voir chapitre 6 page 91). Pour cela, six systèmes représentatifs du domaine sont sélectionnés :

1) SCEPTRE 2 [75] : résultat d'un projet de recherche visant à unifier les interfaces de programmation des systèmes d'exploitation temps réel embarqués,
2) VxWorks [21] : exécutif commercial largement utilisé dans les secteurs industriels de la télécommunication et de la robotique d'après des études de marchés récentes [73],
3) POSIX IEEE std 1003.1 [72] : norme dédiée à la spécification des systèmes d'exploitation et plus spécifiquement à la spécification des systèmes multitâches,
4) RTAI [74] : correctif pour utiliser des systèmes Linux dans des applications multitâches temps réel embarquées,
5) Osek/Vdx-OS [19] : norme dédiée à la spécification des exécutifs multitâches pour l'automobile,
6) Lacatre [76] : un langage, présentant un mode graphique et un mode textuel, destiné à faciliter la conception préliminaire et détaillée d'applications basées sur la mise en œuvre d'exécutifs multitâches temps réel.

Une première étude bibliographique décrite dans [77] a permis d'identifier les principales ressources de ces systèmes. Onze ressources ont été retenues :

- **Les ressources ordonnançables**, c'est-à-dire les ressources fournissant un contexte d'exécution concurrent. La concurrence d'exécution de ces ressources est alors gérée par un ordonnanceur logiciel,
- **Les ressources d'interruptions**, c'est-à-dire les ressources fournissant un contexte d'exécution pour les interruptions matérielles et logicielles,
- **Les alarmes** fournissant un contexte d'exécution pour les interruptions issues de réveils,
- **Les réveils**,
- **Les partitions mémoires** restreignant les exécutions à une zone mémoire,

- **Les ressources de communications par message** permettant de réaliser des communications de données entre contextes d'exécution concurrent,
- **Les ressources d'exclusion mutuelle** permettant de protéger l'accès à des zones mémoires partagées,
- **Les ressources de notification** dédiées à la synchronisation des exécutions concurrentes,
- **Les ressources d'interface de périphériques** permettant d'accéder aux périphériques matériels,
- **Les ressources d'interface mémoire** permettant de gérer la mémoire,
- **L'ordonnanceur** orchestrant l'exécution des ressources ordonnançables.

Pour chacune de ces ressources, une ontologie des services qu'elles fournissent a été réalisée. Cette ontologie est décrite dans les tableaux suivants. Dans ces tableaux, les six premières colonnes représentent les services fournis par chacun des systèmes multitâches étudiés. La dernière colonne décrit le concept ontologique retenu.

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Créer_Tache	TaskInit TaskSpawn	pthread_create	Declare_Task (it's a static declaration)	rt_task_init rt_task_make_periodic rt_task_make_periodic_relative_ns	create_task	createServices
Supprimer_Tache	TaskDeleteExitTaskDeleteFc	pthread_exit pthread_detache pthread_kill	∅ (static system)	rt_task_delete	kill_task	deleteServices
Arreter_Tache	TaskSuspend	pthread_cancel pthread_join	Terminate_Task Chain_Task	rt_task_suspend	Suspend	suspendServices
Lancer_Tache	TaskActivate	pthread_create	Activate_Task	rt_task_make_periodic rt_task_make_periodic_rel rt_task_resume.	Activate	activateServices
Continuer_Tache	TaskResume	pthread_resume	Activate_Task	rt_task_resume	Resume	resumeServices
∅	∅	pthread_yield	Schedule	rt_task_yield	∅	yieldServices
Inspecter_Tache	TaskisReady TaskisSuspend	∅	GetTaskState	rt_get_task_state	Inquire	SWAccessService
Obtenir_ID_Tache_Courar	TaskIdSelf	pthread_self	GetTaskID	∅	∅	SWAccessService
∅	TaskPriorityGet	pthread_getschedparam	∅	rt_get_prio	∅	SWAccessService
Changer_Priorite_Tache	TaskPrioritySet	pthread_setshedprio pthread_setschedparam	∅	rt_change_prio	Change_Priority	SWAccessService
Proteger_Tache	TaskSafe	∅	∅	∅	∅	enableConcurrencyServices
Deproteger_Tache	TaskUnsafe	∅	∅	∅	∅	disableConcurrencyServices
Attendre_Délai Attendre_Date	sleep nanosleep alarm taskdelay	nanosleep clock_nanosleep sleep usleep	SetRelAlarm (relatif)	rt_task_wait_period rt_sleep rt_sleep_until	∅	delayServices

FIG. *B.1 – Ontologie des services des ressources ordonnançables*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Connecter_Interruption	intHandlerCreate intVecSet intStackEnable (catégorie ISR n°1)intConnect Sigaction Sigset Signal	Sigaction Sigset Signal	ceci est réalisé statiquement : ISR nom Catégory, Ressource (tache), message	rt_request_linux_irq	create_interrupt	routineConnectServices
Déconnecter_Interruption	∅	∅	∅	rt_free_linux_irq	Kill_interrupt	routineDisconnectServices
Masquer_Interruption	intLock après un intLockLevelSet intLevelSet intDisable	sigpromask pthread_sigmask sigemptyset sigfillset sigpending	DisableAllInterrupts SuspendAllInterrupts SuspendOSInterrupts	rt_mask_and_ack_irq rt_enable_irq rt_startup_irq	Disable_IT	enableConcurrencyServices
Demasquer_Interruption	intUnLock intLevelSet intEnable	sigpromask pthread_sigmask sigemptyset sigfillset	EnableAllInterrupts ResumeAllInterrupts ResumeOSInterrupts	rt_unmask_irq rt_disable_irq rt_shutdown_irq	Enable_IT	disableConcurrencyServices
Inspecter_Interruption	GetVect	Sigaction	∅	∅	∅	SWAccessService
Lever_Exception	Kill raise	Kill raise	∅	Kill raise	∅	activateServices

FIG. *B.2 – Ontologie des services des ressources d'interruptions*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Créer_Réveil	timer_create wdCreate	timer_create	Création statique : COUNTER (rattache le timer) ACTION (rattache l'action à l'expiration) AUTOSTART (booléen définissant le démarrage automatique) CYCLETIME (périodicité)	rt_set_oneshot_mode rt_timer_insert rt_set_periodic_mode rt_timer_insert	create_Alarm	createServices
Supprimer_Réveil	timer_delete wdDelete	timer_delete	∅	rt_timer_remove	Kill_Alarm	deleteServices
Démarrer_Réveil	timer_settime wdStart taskDelay()	timer_settime alarm	SetRelAlarm (relatif) SetAbsAlarm (absolu)	start_rt_timer (spécifique au tâche périodique)	Resume_Al Activate_Al	activateServices
Arrêter_Réveil	wdCancel	timer_delete	CancelAlarm	stop_rt_timer (spécifique au tâche périodique)	Suspend_Al	suspendServices
Inspecter_Reveil	timer_gettime	timer_gettime	GetAlarm	rt_get_time	∅	SWAccessService

FIG. *B.3 – Ontologie des services des alarmes*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
∅	clock_setres	∅	oil		∅	SWAccessService
∅	clock_getres	clock_getres	∅		∅	SWAccessService
Lire_Date Lire_heure	clock_gettime	clock_gettime	∅	rt_get_time rt_get_time_ns rt_get_cpu_time	∅	SWAccessService
Initialiser_date Reinitialiser_date	clock_settime	clock_settime	oil	update_times	∅	SWAccessService

FIG. *B.4 – Ontologie des services des réveils*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Charger_Capsule Demarrer_Caspule	∅	exec	∅	∅	∅	createServices
Créer_Capsule Charger_Capsule Demarrer_Caspule	∅	fork	∅	∅	∅	forkServices
Arreter_Capsule Supprimer_Capsule	∅	exit	∅	∅	∅	exitServices

FIG. *B.5 – Ontologie des services des partitions de mémoire*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Créer_Boite_a_lettres	msgQCreate	mq_open	Création statique : PROPERTY : RECEIVE_QUEUED_INTERNAL	rt_mbx_init	create_mailbox	createServices
Supprimer_Boite_a_lettres	msgQDelete	mq_close mq_unlink	∅	rt_mbx_delete	kill_mailbox	deleteServices
Recevoir_Message_Court	msgQSend	mq_send	∅	rt_mbx_send _rt_mbx_ovrwr_send rt_send	Send_To_Mbx	sendServices
Envoyer_Message_Court	msgQReceive	mq_receive	∅	rt_mbx_receive rt_receive : m rt_mbx_receive_if rt_mbx_receive_wp rt_receive_if : m	wait_On_Mailbox	receiveServices
∅	msgQshow msgQInfoGet msgQNumMsg	mqPxShow	GetStatus	∅	∅	SWAccessService

FIG. *B.6 – Ontologie des services des ressources de communication par message*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Semaphore_create	SemCCreate semBCreate semMCreate	sem_init sem_open pthread_spin_init pthread_mutex_init	∅ DeclareResource	rt_sem_init rt_typed_sem_init	Create_Sempahore	createServices
Semaphore_delete	semDelete	sem_Destroy sem_Close sem_unlink pthread_mutex_destroy pthread_spin_destroy	∅	rt_sem_delete rt_spl_delete	Kill_Semaphore	deleteServices
Take_semaphore Try_Take_Semaphore Prendre_Ressource Prendre_Ressource_Conditionnel	semTake	sem_wait pthread_spin_lock pthread_mutex_unlock sem_trywait sem_timedwait	GetResource	rt_sem_wait rt_spl_lockrt_sem_wait_until rt_sem_wait_timed rt_sem_wait_if rt_spl_lock_if,until,timed	WaitToSem	acquireServices
Give_semaphore Liberer_Ressource	semGive semFlush	sem_post pthread_spin_unlock pthread_mutex_unlock	ReleaseResource	rt_sem_signal rt_spl_unlock	SendToSem	releaseServices
Inquire_Semaphore Inspecter_Ressource	semShow	sem_getValue	∅	∅	∅	SWAccessService

FIG. *B.7 – Ontologie des services des ressources d'exclusion mutuelle*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Créer_Evénement	semEvStart	Sigaction sigpromask pthread_sigmask (sigevent structure)	∅	∅	Create_Event	createServices
Supprimer_Evénement	semEvStop	∅	∅	∅	kill_Event	deleteServices
Attendre_Evénement	eventReceive	sigwait sigtimedwait sigsuspend	WaitEvent	∅	Wait_Event	waitSservices
Signaler_Evénement	eventSend semFlush	pthread_kill kill sigqueue	SetEvent	∅	Signal_Event	signalServices
Effacer_Evénement	eventClear	sigwait sigtimedwait	ClearEvent	∅	Clear_Event	clearServices
Inspecter_Evenement	∅	∅	GetEvent	∅	∅	SWAccessService

FIG. *B.8 – Ontologie des services des ressources de notification*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Créer_Unité Assigner_Unité	create	create	∅	∅	∅	createServices
Ouvrir_Unite	open fopen	open fopen	∅	∅	∅	openServices
Demande_entree_sorties	read write ioctl fread fwrite aio_read aio_write	read write ioctl fread fwrite aio_read aio_write	∅	∅	∅	readServices writeServices
Inspecter_Unite	∅	∅	∅	∅	∅	SWAccessService
Fermer_Unite	close	close	∅	∅	∅	closeServices
Supprimer_Unite	delete remove	delete remove	∅	∅	∅	deleteServices

FIG. *B.9 – Ontologie des services des ressources d'interface de périphériques*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Créer_Segment	∅ (flat memory)	∅	∅	∅ (flat memory)	∅	createServices
Supprimer_Segment	∅ (flat memory)	∅	∅	∅ (flat memory)	∅	deleteServices
Allouer_Zone	Valloc Calloc realloc memPartSmCreate SemMemcalloc	Malloc, mmap	∅	xnheap_alloc	∅	MemoryBlockAlloc
Restituer_Zone	Free semMemFree	Free, munmap	∅	xnheap_free	∅	MemoryBlockFree
Allonger_Segment	∅	∅	∅	∅	∅	∅
Raccourcir_Segment	∅	∅	∅	∅	∅	∅
Inspecter_Segment	∅	∅	∅	∅	∅	∅
∅	∅ (pas de page et d'adresse virtuelle)	Mlock mlockAll	∅	∅	∅	MemoryBlockLock
∅	∅ (pas de page et d'adresse virtuelle)	Munlock munlockAll	∅	∅	∅	MemoryBlockUnLock

FIG. *B.10 – Ontologie des services des ressources d'interface mémoire*

SCEPTRE 2	VxWorks	POSIX	OSEK	RTAI	LA4	SRM
Partager_Temps	KernelTimeSlice	pthread_setschedparam	∅	∅	∅	SWAccessService
Arreter_Partage_Temps	KernelTimeSlice	pthread_setschedparam	∅	∅	∅	SWAccessService
∅	∅	sched_setparam sched_setscheduler	∅	rt_set_sched_policy	∅	SWAccessService
∅	∅	sched_getparam sched_getpriority_Max sched_getorioriy_Min sched_rr_get_interval	∅	∅	∅	SWAccessService
Bloquer_Ordonnanceur	TaskLock		RES_SCHEDULER (lock the Res_scheduler ressource)	rt_sched_lock	∅	enableConcurrencyServices
Liberer_Ordonnanceur	TaskUnlock		RES_SCHEDULER (unlock the Res_scheduler ressource)	rt_sched_unlock	∅	disableConcurrencyServices

FIG. *B.11 – Ontologie des services de l'ordonnanceur*

Intégration de SRM dans MARTE

C

sectionPrésentation générale de MARTE

Un des objectifs du consortium OMG est de normaliser des profils pour des domaines informatiques donnés. Des profils pour la modélisation système, SysML [86], pour l'ingénierie des processus, SPEM [87], ont par exemple été proposés. En vue de remplacer l'extension normalisée pour le domaine du temps réel embarqué, SPT [9], l'OMG a émis un nouvel appel à soumission pour un profil UML dédié à la modélisation et à l'analyse des systèmes temps-réel embarqués. Le profil UML MARTE UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) proposé par le consortium ProMARTE[1] vise à répondre à cet appel.

La structure générale du profil UML MARTE est illustrée dans la figure C.1 page suivante . Un premier package, nommé MARTE Foundation, définit les fondations du profil. Ce sont des artéfacts abstraits ou concrets. Ils sont utilisables aussi bien pour l'analyse que pour la conception des systèmes. Ils permettent la représentation du temps (package TIME), des propriétés dites non-fonctionnelles (package Non Functional Properties), des technologies composants (package General Component Model), des plates-formes à un au niveau d'abstraction (package Generic Ressource) et de l'allocation (package Allocation). Un second package (MARTE Design Model) définit les concepts utiles à la conception des systèmes temps réel embarqués. Il définit notamment la modélisation de l'application (package High Level Application Modeling (Hlam)) ainsi que la modélisation des plates-formes logicielles et matérielles (packages Software Ressource Modelling et Hardware Ressource Modelling). Un troisième package (MARTE Analysis Model) complète l'ancien profil SPT pour la description des concepts d'analyse d'ordonnancement (package Schedulability Analysis Model) et des concepts d'analyse de performance (package Performance Analysis Model).

[1] www.promarte.org

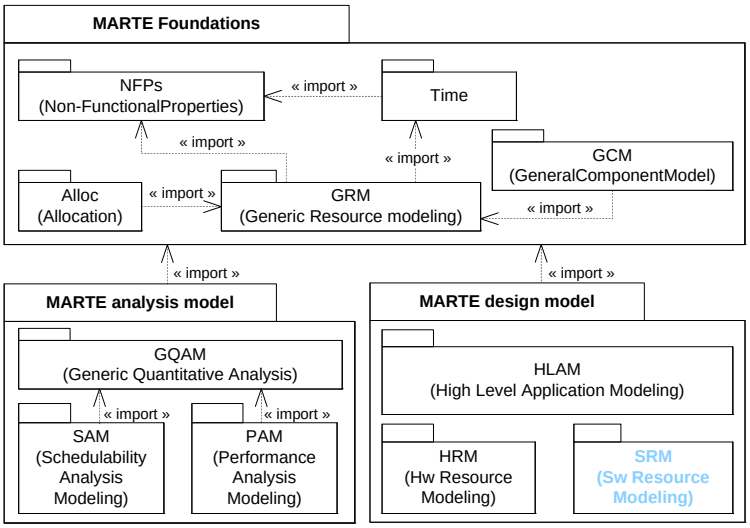

FIG. *C.1 – Synoptique du profil UML MARTE*

C.1 Intégration du profil SRM dans le profil MARTE

Le profil SRM est intégré à MARTE. Pour cela, les concepts de SRM héritent des concepts de GRM. De plus, certaines règles d'implantations ont été suivies :

- Le noms des packages sont précédés du préfixe SW_. Le package Concurrency devient SW_Concurrency,
- Les noms redondants dans MARTE sont préfixés, pour le logiciel, par Sw. Ainsi ConcurrentResource devient SwConcurrentResource, SchedulableResource devient SwSchedulableResource, TimerResource devient SwTimerRessource, InteractionResource devient SwInteractionResource, CommunicationResource devient SwCommunicationResource et SynchronisationResource devient SwSynchronisationResource ,
- Les concepts de ResourceService, ResourceProperty, ResourceParameter et ResourceInstance sont implantés par GRM,
- Le concept de l'ordonnanceur Scheduler est intégré dans le chapitre GRM,
- les services des horloges TimerRessource sont implantés par GRM (voir GRM::TimerRessource),

Enfin, une dernière règle d'implantation concerne les associations entre les concepts, et plus particulièrement entre les différentes ressources. Chaque extrémité navigable de l'association dans le modèle de domaine est traduite en un attribut typé TypedElement et une contrainte spécifiant que le type de ce TypedElement doit être stéréotypé par le concept originel du modèle de domaine. Par exemple dans la figure C.2 page suivante , à partir de la propriété addressSpace, un attribut et une contrainte sont créés. La justification de ce choix est purement technique et pratique. Elle répond à la faiblesse de certains outils pour manipuler des références entre des stéréotypes appliqués. Cette règle de transformation résout le problème en référençant un élément conforme à un élément UML et non à un stéréotype.

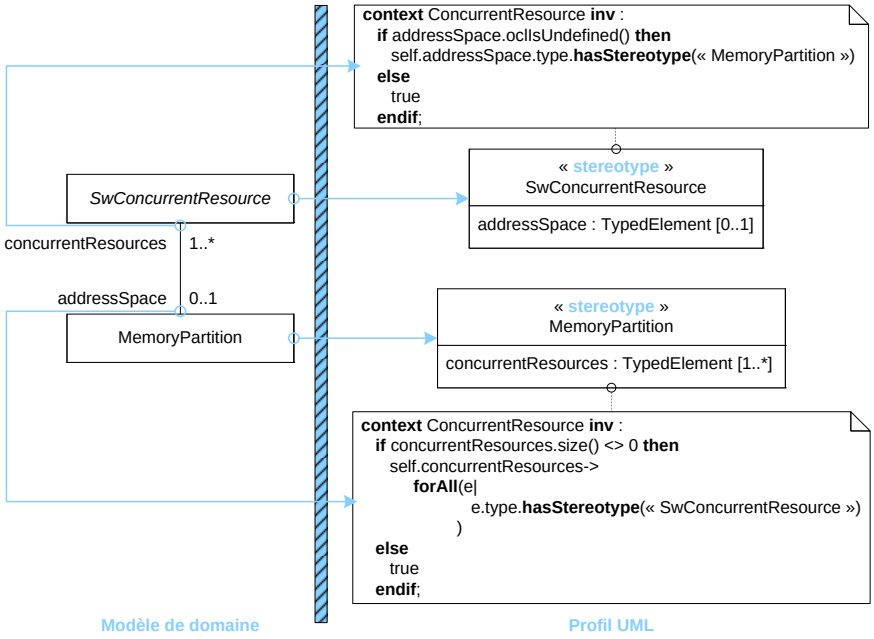

FIG. *C.2 – Règles d'implantation des associations entre stéréotypes*

Description des modèles et des règles de transformation utilisés dans l'expérimentation du composant τ_{p2p}

D

Ce chapitre vise à décrire le modèle de la plate-forme Cheddar et les règles de transformation utilisées dans le processus expérimental de cette étude (voir chapitre 7 page 113).

D.1 Modélisation de la plate-forme Cheddar

L'outil Cheddar est un outil[1] destiné à l'analyse d'ordonnancement des applications temps réel [83]. Ainsi, cet outil fournit les concepts d'application, de processeur, de tâche, d'espace d'adressage, de ressource partagée et de message. Les modélisations de ces concepts sont illustrées dans les figures[2] D.1, D.2, D.3, D.4 et D.5.

[1] http ://beru.univ-brest.fr/šinghoff/cheddar/index-fr.html
[2] Modèles réalisées avec l'outil Papyrus, http ://www.papyrusuml.org.

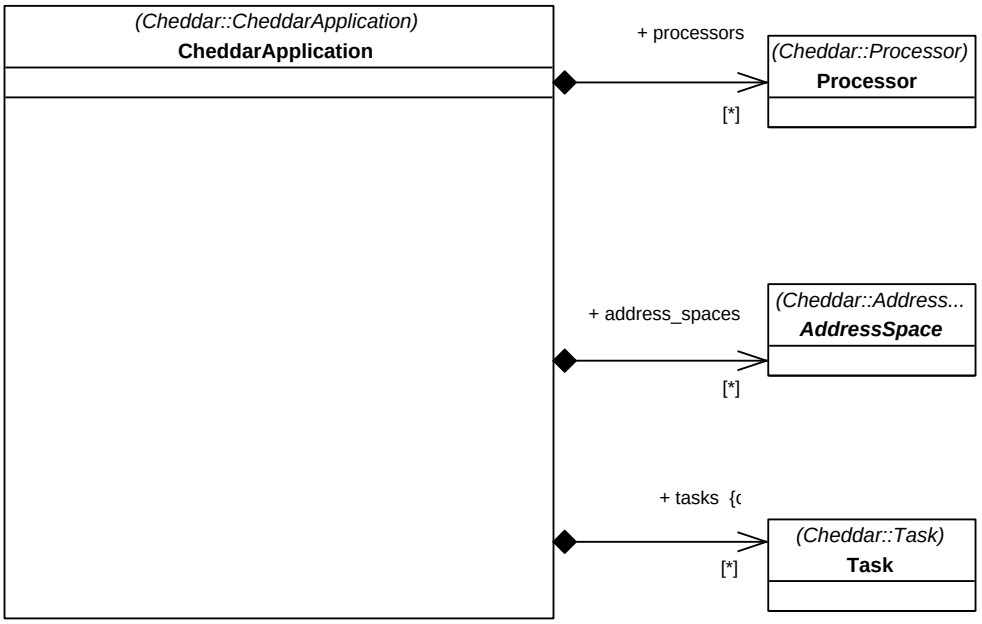

FIG. *D.1 – Modélisation du concept d'application dans la plate-forme* CHEDDAR

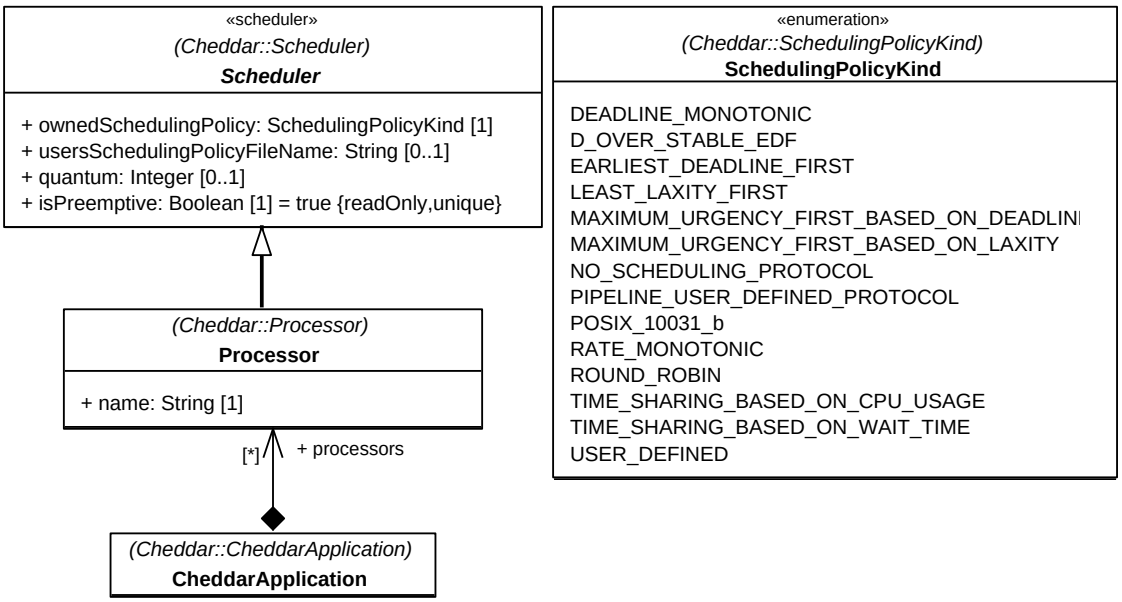

FIG. *D.2 – Modélisation de la ressource processeur fournis par la plate-forme* CHEDDAR

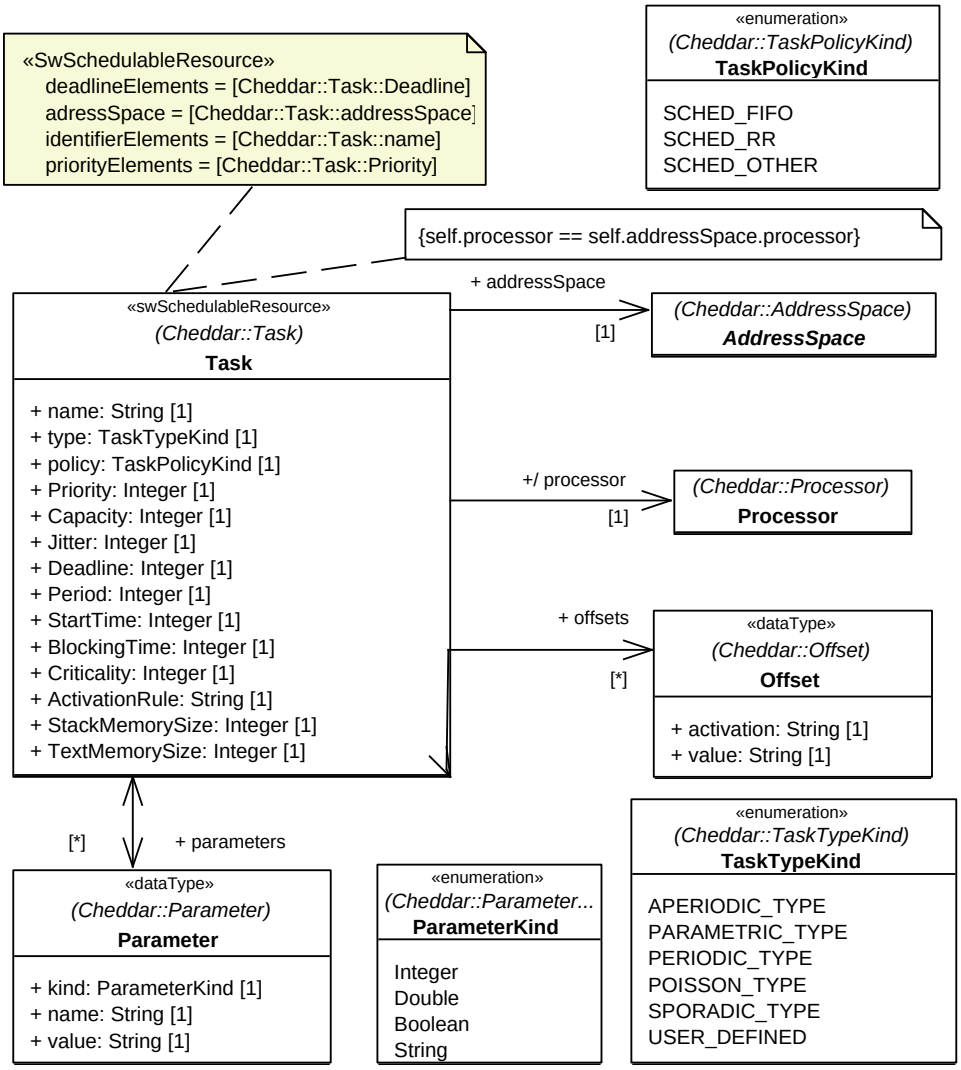

FIG. *D.3 – Modélisation de la ressource ordonnançable fournis par la plate-forme* CHEDDAR

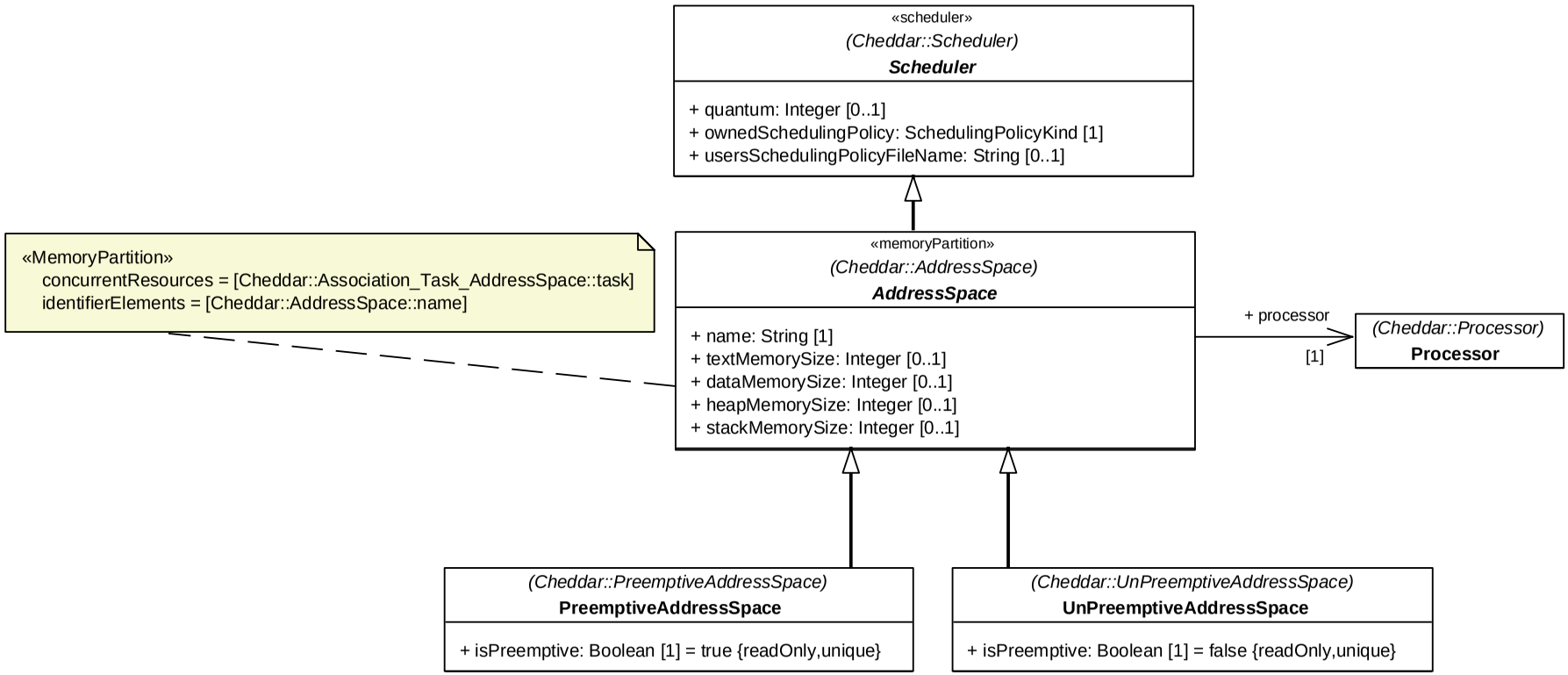

FIG. *D.4 – Modélisation des partitions mémoires dans* CHEDDAR

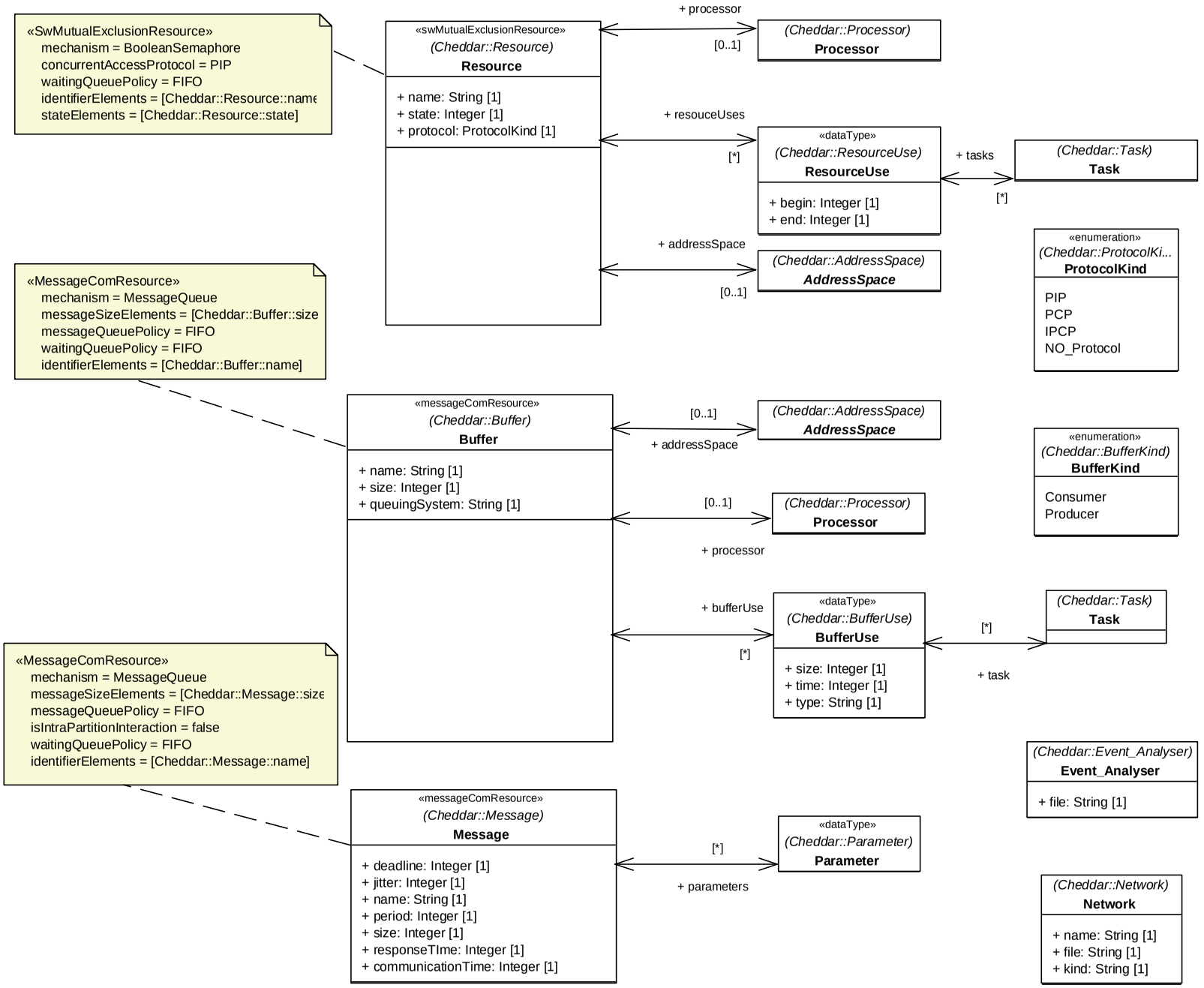

FIG. *D.5 – Modélisation des ressources d'interaction du modèle* CHEDDAR

D.2 Description des règles de transformation spécifiées par l'utilisateur dans le processus expérimental

Cette section vise à présenter les règles de transformation ATL [24] décrites en entrée du composant τ_{p2p}. Ces règles sont spécifiées par l'utilisateur. Elles concernent, d'une part, le portage de la plate-forme Hlam vers la plate-forme Osek/Vdx-OS et, d'autre part, le portage de la plate-forme Osek/Vdx-OS vers la plate-forme Cheddar (voir la synoptique du processus en figure 7.6 page 120).

D.2.1 Description de la règle de transformation spécifique au portage Hlam2Osek

Dans le cas du portage Hlam2Osek, les types de donnée Duration et AutoStart ne sont pas identifiés comme des ressources SRM. L'utilisateur doit donc spécifier comment transformer une instance source Duration en une instance cible AutoStart. Le noyau générique de transformation τ_{p2p} détermine ensuite qui référence cette instance source et transforme ces références pour cibler l'instance AutoStart générée. Ainsi, la règle de transformation DurationInstanceValueToAutoStartInstanceValue écrite spécifiquement pour ce portage est définie dans la description D.1 page ci-contre.

Le type de donnée Duration est constitué de deux propriétés : un réel exprimant une durée (value) et une unité (unit) associée à cette durée. Le type de donnée AutoStart possède quant à lui trois propriétés : un booléen pour l'auto-démarrage (isAutostarted), une valeur entière temporelle pour spécifier l'échéance au bout de laquelle l'alarme expire pour la première fois (alarmTime) et une valeur entière pour spécifier la période d'expiration de l'alarme (cycleTime). Les valeurs des propriétés alarmTime et cycleTime sont exprimées dans l'unité d'impulsion compteur[3]. Les échéances sont donc des multiples de la période compteur.

Dans cette étude, le compteur émet des impulsions toutes les cinquante millisecondes[4] (voir la collaboration 7.4 page 118). La valeur des échéances est donc un multiple de cinquante millisecondes. De plus la première échéance est égale aux autres échéances, ainsi dans cette étude alarmTime=cycleTime. La règle de transformation DurationInstanceValueToAutoStartInstanceValue consiste donc à déterminer ce multiple à partir de la valeur réelle spécifiée dans l'instance Duration source et à l'affecter aux propriétés alarmTime et cycleTime . Il est donc égale à la partie entière de la formule $value * 1000/50$ si l'unité de la durée Duration est la seconde (s), sinon il est égale à la partie entière de la formule $value/50$. Le noyau générique implante, de manière native, la

[3] Les unités des propriétés cycletime et alarmTime sont en nombre d'impulsions compteur, c'est-à-dire en nombre d'impulsions du timer matériel nécessaires pour faire progresser le compteur d'une unité.

[4] Dans l'implantation Osek/Vdx-OS utilisée pour cette étude, l'implantation Trampoline (http ://trampoline.rts-software.org/) [60], le compteur est associé à un timer dont l'impulsion est fixée, par défaut, à cinquante millisecondes.

transformation des types primitifs comme par exemple la transformation d'une valeur réelle en une valeur entière.

Dans la description suivante, une flèche ↘ précise que la ligne a été coupée dans ce document pour des raisons éditoriales. Cette flèche n'apparait pas dans la description finale.

```
1  rule DurationInstanceValueToAutoStartInstanceValue{
2    --motif source imposée par le composant
3    --de transformation
4    --pour faciliter la fusion
5    --avec les règles de transformation du noyau
6    from  src : UML2!InstanceValue,
7          targetDefiningFeature : UML2!StructuralFeature,
8          namespace : UML2!Package
9          (src.instance.classifier->first().name = 'Duration↘
               ')
10
11   to dest : UML2!InstanceValue (
12           instance<-instanceAutoStart
13   ),
14   --création de l'instance autoStart
15   instanceAutoStart : UML2!InstanceSpecification(
16     name<- src.instance.name+'_'+targetDefiningFeature.↘
           name,
17     classifier<-targetDefiningFeature.type,
18     slot<-slotAlarmTime,
19     slot<-slotCycleTime,
20     slot<-slotAutoStarted
21   ),
22   --création de l'emplacement dans l'instance
23   --pour donner une valeur
24   --à la propriété alarmTime
25   slotAlarmTime : UML2!Slot(
26     definingFeature <- targetDefiningFeature.type.↘
           ownedAttribute
27             ->select(e|
28             e.name = 'alarmTime')->first(),
29     value<-valueAlarmTime
30   ),
31
32   --la première échéance est égale à src.value*1000/50
33   --si l'unité source
34   --est la seconde et src.value/50 sinon.
35   valueAlarmTime : UML2!LiteralInteger(
36     value<-if src.instance.slot
37         ->select(e|
38         e.definingFeature.name = 'unit')
39         .first().value = #s then
40           (src.instance.slot
41           ->select(e|
42           e.definingFeature.name = 'value'
43            ).first().value.first().value*1000 / 50).round↘
                 ()
44       else
45             (src.instance.slot
46           ->select(e|
```

```
47          e.definingFeature.name = 'value'
48          ).first().value.first().value / 50).round()
49      endif
50  ),
51  --création d'un emplacement dans l'instance
52  --pour donner une valeur
53  --à la propriété cycleTime
54  slotCycleTime : UML2!Slot(
55    definingFeature <- targetDefiningFeature.type.↘
        ownedAttribute
56            ->select(e|
57            e.name = 'cycletime')->first(),
58    value<-valueCycleTime
59  ),
60
61  --l'échéance est égale à src.value*1000/50
62  --si l'unité source
63  --est la seconde et src.value/50 sinon.
64  valueCycleTime : UML2!LiteralInteger(
65    value<-if src.instance.slot
66        ->select(e|
67        e.definingFeature.name = 'unit'
68        ).first().value = #s then
69          (src.instance.slot
70          ->select(e|
71          e.definingFeature.name = 'value'
72          ).first().value.first().value*1000 / 50).round()
73       else
74            (src.instance.slot
75          ->select(e|
76          e.definingFeature.name = 'value'
77          ).first().value.first().value / 50).round()
78      endif
79  ),
80  --création d'un emplacement dans l'instance
81  --pour affecter une valeur à la propriété isAutostarted
82  slotAutoStarted : UML2!Slot(
83    definingFeature <- targetDefiningFeature.type.↘
        ownedAttribute
84            ->select(e|
85            e.name = 'isAutostarted'
86            )->first(),
87    value<-autostartValue
88  ),
89  --l'alarme est par défaut auto-démarrée
90  autostartValue : UML2!LiteralBoolean(
91    value<-true
92  )
93
94  do{
95  --lignes imposées par l'opération de fusion
96  --(première version)
97  dest;
98  namespace.packagedElement<-namespace.packagedElement->↘
      append(instance);
99  }
```

```
100 }
```

Code D.1 – Règle de transformation spécifiée par l'utilisateur dans le portage HLAM2OSEK

D.2.2 Description de la règle de transformation spécifique au portage Osek2Cheddar

La transformation spécifique au portage OSEK2CHEDDAR consiste à transformer les instances AutoStart en des valeurs entières. Ces valeurs sont alors affectées automatiquement, par le noyau générique, aux propriétés spécifiant la période des tâches (la propriété désignant la période de la tâche est référencée par SRM). L'outil CHEDDAR n'impose pas d'unité de temps particulière. Les périodes sont par exemple représentées par des entiers. Ainsi, dans le cas du portage OSEK2CHEDDAR, l'utilisateur peut considérer l'unité comme l'impulsion compteur, c'est-à-dire cinquante millisecondes. Il peut donc directement spécifier la période des tâches par la valeur de la propriété CYCLETIME. L'analyse d'ordonnançabilité dans l'outil CHEDDAR sera alors effectuée toutes les cinquante millisecondes (valeurs du compteur). Comme le montre la description D.2, l'utilisateur transforme la valeur affectée à la propriété cycleTime en une valeur entière.

Dans la description suivante, une flèche ↘ précise que la ligne a été coupée dans ce document pour des raisons éditoriales. Cette flèche n'apparait pas dans la description finale.

```
1  rule AutoStartValuetoIntegerValue{
2    --motif source imposée par le composant
3    --de transformation
4    --pour faciliter la fusion
5    --avec les règles de transformation du noyau
6    from  src : UML2!InstanceValue, targetDefiningFeature : ↘
        UML2!StructuralFeature, namespace : UML2!Package (src↘
        .instance.classifier->first().name = 'Autostart' )
7        to dest : UML2!LiteralInteger(
8      value<-(src.instance.slot
9        ->select(e|
10       e.definingFeature.name = 'cycletime'
11       ).first().value.first().value)
12     )
13   do{
14   --lignes imposées par l'opération de fusion
15   --(première version)
16   dest;
17       namespace.packagedElement<-namespace.packagedElement↘
           ->append(dest);
18     }
19 }
```

Code D.2 – Règle de transformation spécifiée par l'utilisateur dans le portage OSEK2CHEDDAR

UMLPatternDetectionLib : Une librairie de détection de schémas de conception UML

E

Dans certaines usines logicielles utilisant l'ingénierie dirigée par les modèles comme technologie socle, il existe un besoin spécifique pour la détection d'un ensemble d'entités collaborant suivant un motif précis. Ceci est particulièrement vrai pour les transformations de modèles décrites par un ensemble de règles déclaratives. Ces règles sont décrites à partir d'un ensemble de types sources et un ensemble de types cibles. Ces types sont des métatypes dans le cas de métamodèle ou des types appartenant à un modèle dans le cas de librairies de modèle. Dans certaine de ces règles les types sources sont multiples. Il doit donc exister un ensemble de librairie capable de découvrir automatiquement ou semi-automatiquement qu'elles sont les éléments du modèle d'entrée qui collaborent conformément aux types sources de la règle. La collaboration de ces types sources forme alors un motif, c'est-à-dire un schéma de conception [55], qu'il faut détecter dans le modèle d'instance fourni en entré de la transformation.

La détection de motifs de conception est un travail de recherche à part entier. Des travaux tels que [88, 89, 90, 91] sont par exemple dédiés à la détection de motifs de conception dans des modèles et dans du code source (majoritairement du Java). Afin d'expérimenter le noyau de transformation τ_{p2p}, une librairie ATL [24],nommée UMLPatternDetectionLib, a été spécialement décrite. Cette librairie s'inspire des algorithmes spécifiés dans [84]. Elle se veut expérimentale et pragmatique. Elle répond à un besoin particulier apparu au cours de l'expérimentation. Elle permet, lors d'une transformation ATL, de détecter des schémas de conception dans un modèle d'instance d'UML. A partir d'un modèle d'instance, d'un modèle de classes et d'une description des motifs à détecter, cette librairie retourne la liste des instances correspondant aux différents motifs, c'est-à-dire collaborant conformément aux différents motifs.

E.1 Présentation des modèles d'entrée

La figure E.1 page suivante décrit la synoptique d'utilisation de cette librairie. Les modèles d'entrée de la librairie sont principalement de trois types :

– le modèle de classes, c'est-à-dire le modèle des types de l'application. Ce modèle est composé de classes UML. Il représente l'architecture de l'application. La figure E.2 page ci-contre illustre par exemple un modèle composé de trois classes A, B et C. Les couples (A,B) et (B,C) sont liés par deux associations.
– le modèle des collaborations, c'est-à-dire le modèle des collaborations des types du modèle de classes. Ces collaborations concrétisent les motifs à détecter. La figure E.2 page suivante illustre un modèle où une collaboration, nommée AetBCollabore est décrite. Cette collaboration est composée de parties jouant les rôles a et b. Ces rôles sont typés par les classes du modèle précédent, respectivement par A et B. Ils sont liés par un connecteur. Ce connecteur est typé par l'association abAsso. Cette collaboration spécifie donc un motif dans lequel trois instances de A doivent être liées à une instance de B. Dans ce motif, le nombre d'instance de A et de B qu'il faut détecter sont identiques au nombre d'instance qui doivent être connectées (la multiplicité des parties et des extrémités des connecteurs sont identiques). Dans certain cas ces nombres peuvent être différents.
– le modèle des instances, c'est-à-dire le modèle des spécifications d'instance d'UML. C'est dans ce dernier modèle que la librairie tente de détecter des instances collaborant conformément aux motifs décrit dans le modèle des collaborations. La figure E.3 page ci-contre illustre un modèle d'instance dont les types sont spécifiés dans le modèle de classes.

Typiquement, la librairie retourne dans le cas des modèles des figures E.2 page suivante et E.3 page ci-contre , une première liste composée des instances [a1,b1,a2] puis une seconde liste composée des instances [a3,b2,a4]. Pour ces instances, le motif est détecté, c'est-à-dire que les instances collaborent conformément au motif AetBCollabore.

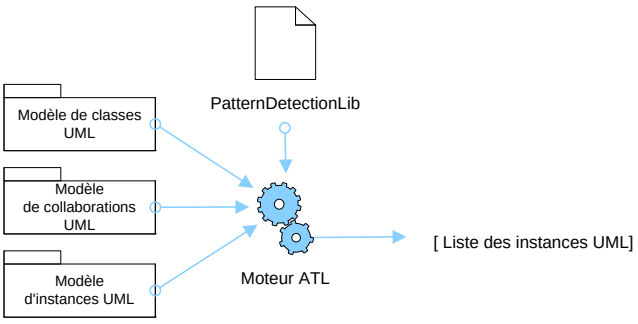

Fig. *E.1 – Synoptique d'utilisation de la librairie* UMLPatternDetectionLib

E.2 Présentation des contraintes

Dans cette librairie, une collaboration UML est le support de description des motifs recherchés. Par construction, cette collaboration décrit implici-

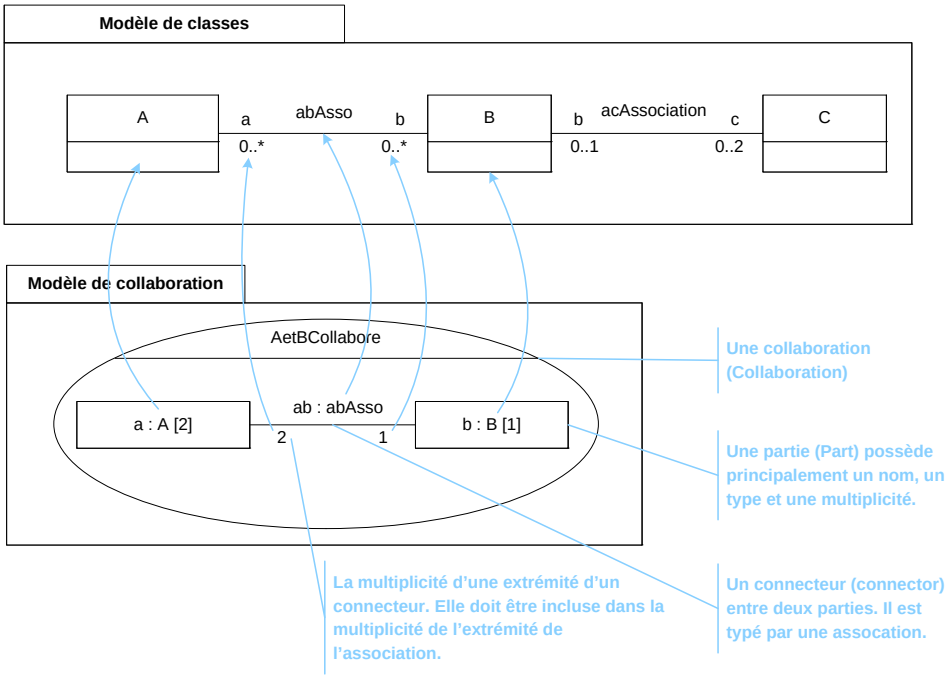

FIG. *E.2 – Exemple d'un modèle de classes et d'un modèle de collaboration*

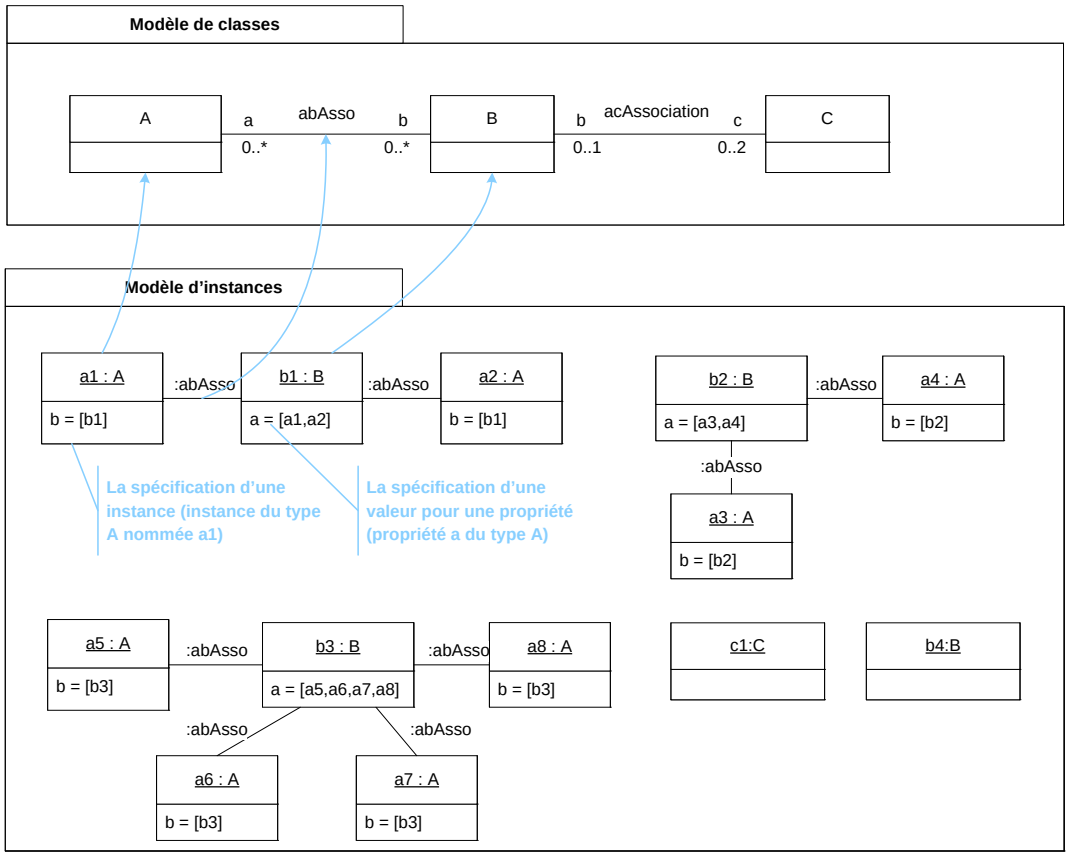

FIG. *E.3 – Exemple d'un modèle d'instances*

tement des contraintes que la libraire cherche à satisfaire sur l'ensemble des instances.

Ces contraintes sont classées en trois grandes familles :

1. les contraintes génériques au modèle UML, c'est-à-dire les contraintes invariantes qui ne sont pas dépendantes du motif lui-même mais générique au métamodèle UML. Ces contraintes sont décrites dans la librairie.
2. Les contraintes dynamiques, c'est-à-dire les contraintes spécifiques au motif. Ces contraintes sont induites par la librairie.
3. Les contraintes utilisateurs, c'est-à-dire les contraintes écrites explicitement par l'utilisateur sous la forme de contraintes OCL.

E.2.1 Les contraintes génériques

Ces contraintes sont liées aux métamodèles UML lui-même. Elles sont au nombre de trois :

- les instances doivent être typées par un type qui est référencé dans le motif, c'est-à-dire une classe qui type une partie (un rôle) de la collaboration. Par exemple, les instances (a1 ... a8) de type A sont susceptibles de collaborer car ce type A est référencé par le rôle a dans le motif AetBCollabore. En revanche, l'instance c1 ne peut pas être détectée pour le motif AetBCollabore car son type n'est pas référencé dans ce motif.
- les instances doivent spécifier une valeur pour une propriété dont le type est référencé dans le motif. Par exemple, une instance spécifiant une valeur pour la propriété a du type B est une instance susceptible de collaborer car le type B est spécifié dans le motif AetBCollabore.
- le nombre d'instance détectée pour un type donné doit être inférieur ou égal à la multiplicité de la partie partageant le même type. Par exemple, le nombre d'instance typée A et détectée pour la partie (le rôle) a doit être égale à trois.

E.2.2 Les contraintes spécifiques

Les contraintes spécifiques au motif sont induites dynamiquement par la librairie, c'est-à-dire qu'elles sont construites pendant l'exécution de l'algorithme. Elles concrétisent les connections entre les parties de la collaboration. Une contrainte est crée pour chaque extrémité du connecteur.

Par exemple, pour l'extrémité a du connecteur ab de la collaboration AetBCollabore, la contrainte suivante est construite :

- Pour les instances de type B, il faut une spécification des valeurs de la propriété a ou deux instances de type A sont référencées.

De même pour l'extrémité b,

- Pour les instances de type A, il faut une spécification des valeurs de la propriété b ou une instance de type B est référencée.

E.2.3 Les contraintes utilisateurs

Les contraintes utilisateurs sont exécutées pendant l'exécution de l'algorithme. Pour cela, le moteur de transformation ATL appelle le plugin Eclipse de validation de contrainte OCL. Dans la version actuelle ces contraintes sont spécifiées dans le contexte de la collaboration. L'écriture du corps de ces contraintes est à la charge du concepteur.

E.3 Présentation des algorithmes de la librairie

E.3.1 Introduction à l'algorithme général de la librairie

L'algorithme de la librairie se décompose en cinq étapes :

- Initialisation des variables et des instances du modèle
- Réduction de l'espace des solutions par vérification des contraintes invariantes, c'est-à-dire des contraintes sur les types et sur la spécification de valeur pour des propriétés dont les types collaborent.
- Séparation de l'espace des solutions en sous-espace, c'est-à-dire classification des instances en sous listes. Ces listes représentent les instances collaborant effectivement pour un motif donné.
- Réduction des sous-espaces valides par satisfaction des post-contraintes, c'est-à-dire le nombre d'élément attendu dans ces espaces.
- Réduction finale par satisfaction des contraintes utilisateurs

La figure E.4 page suivante illustre l'algorithme général. Dans cette figure, les informations en bleues illustrent une exécution de l'algorithme pour les instances de la figure E.3 page 171 .

E.3.2 Introduction à l'algorithme de classification des instances

Cet algorithme de classification des instances est divisé en deux grandes étapes :

- Création des contraintes spécifiques aux motifs, c'est-à-dire les référencements nécessaires entre les instances.
- Classification par appel récursif des instances candidates, c'est-à-dire que tant que toutes les instances ne sont pas classées l'algorithme est appelé récursivement sur les instances candidates.

La figure E.5 page 175 illustre l'algorithme de classification des instances. Dans cette figure, les informations en bleues illustrent une exécution de l'algorithme pour les instances de la figure E.3 page 171 .

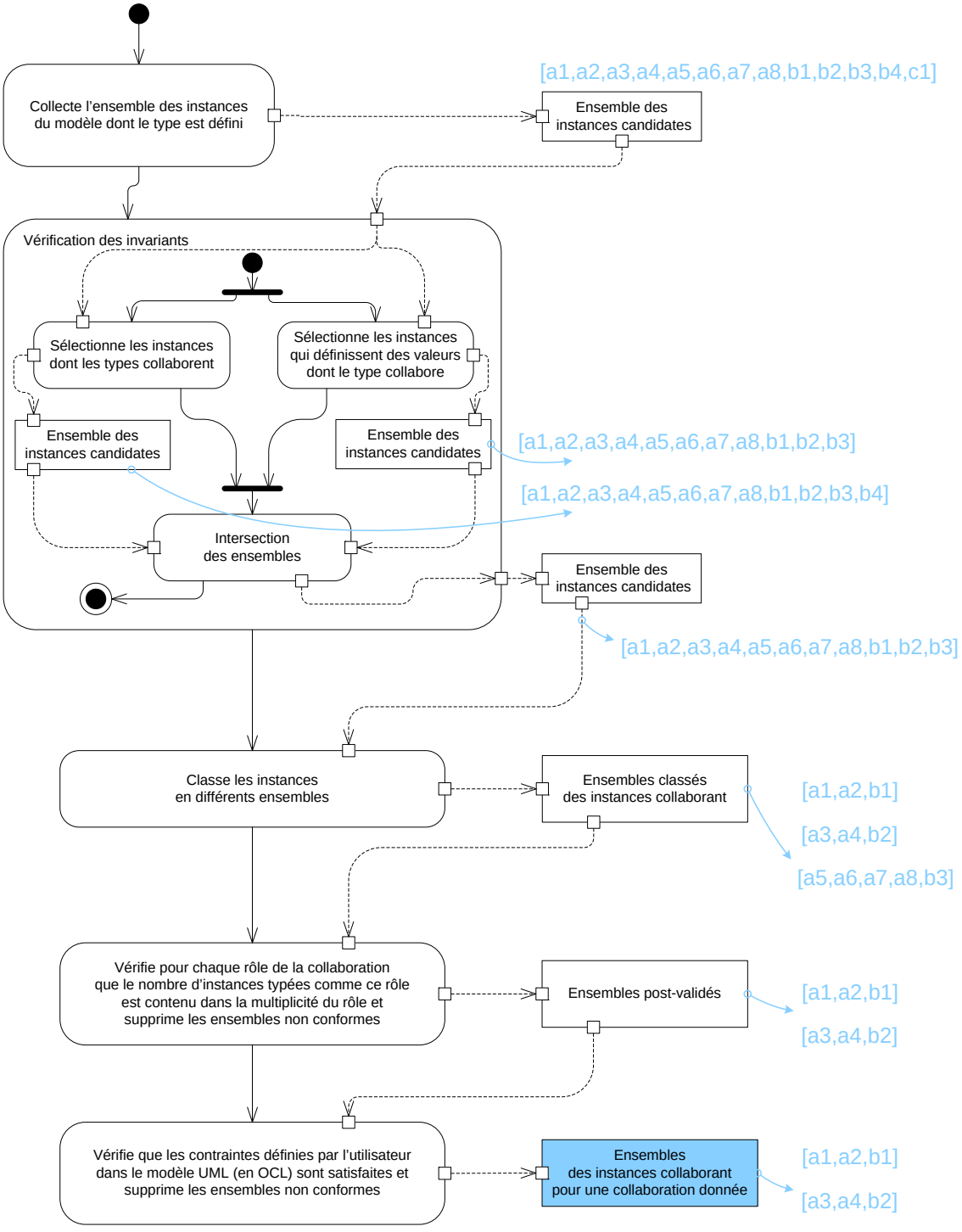

Fig. *E.4 – Algorithme général de la librairie*

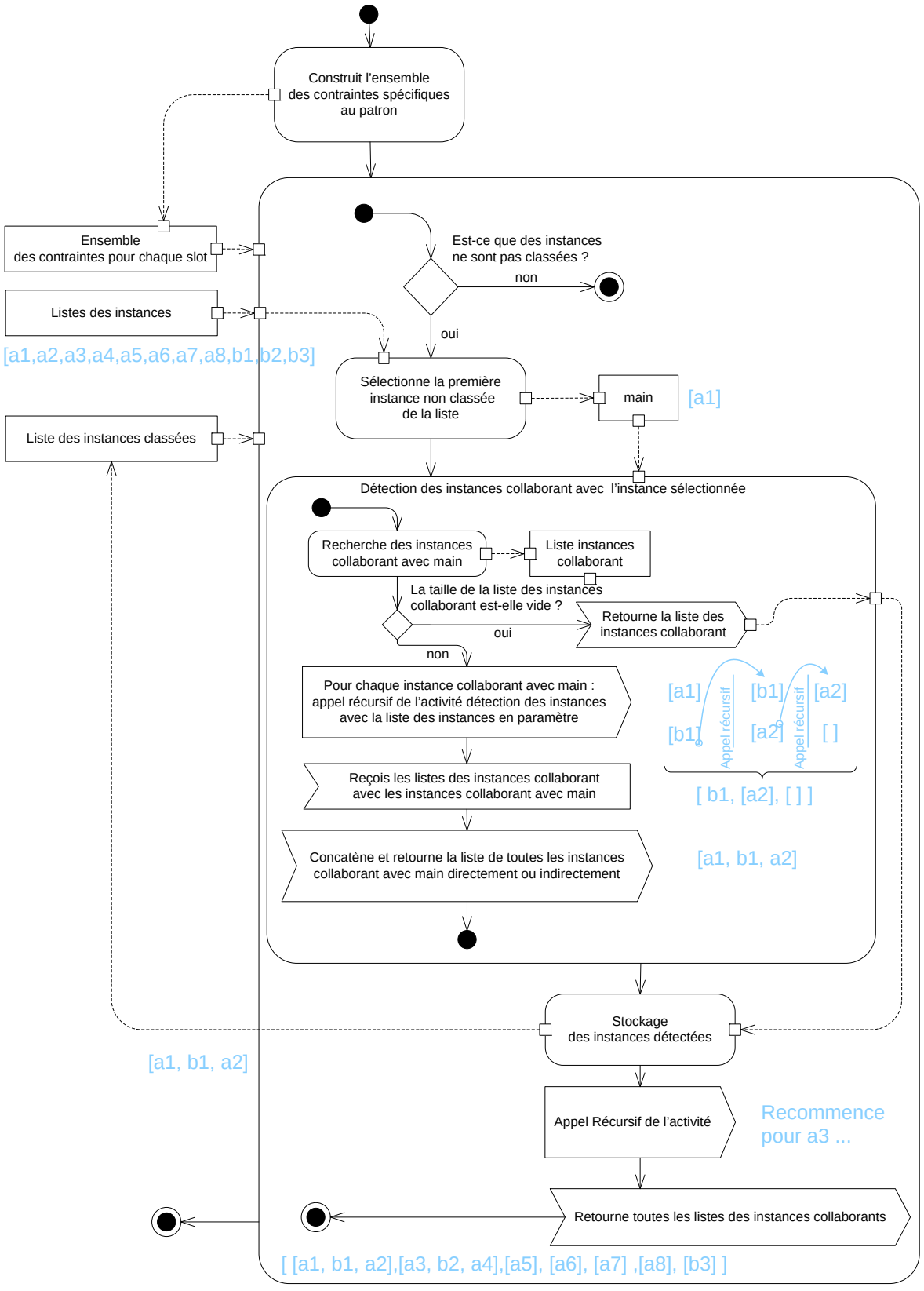

FIG. *E.5 – Algorithme de classification des instances*

Liste des figures

Liste des tableaux

Liste des Algorithmes

Liste des codes

Bibliographie

[1] Joseph Beretta : Les systèmes électroniques embarqués : un enjeu majeur pour l'automobile. *Dans Journée nationales de réflexion et de perspective sur les systèmes embarqués*, Paris, France, Juin 2003. (Cité page 1.)

[2] Manfred Broy : Challenges in automotive software engineering. *Dans Proceedings of the 28th international conference on Software engineering (ICSE '06)*, pages 33–42, New York, NY, USA, 2006. ACM. (Cité page 1.)

[3] Hermann Kopetz : The complexity challenge in embedded system design. *Dans Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 3–12, Los Alamitos, CA, USA, Mai 2008. IEEE Computer Society. (Cité page 1.)

[4] Pierre Maret et Jean-Marie Pinon : *Ingénierie des savoir-faire*. Numéro ISBN 2-86601-620-3. Hermès - Lavoisier, Paris, France, 1ère édition, Juillet 1997. (Cité page 1.)

[5] Michael L. Scott : *Programming language pragmatics*. Numéro ISBN 0-12-63951-1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000. (Cité page 1.)

[6] Jean-Marie Favre : Towards a basic theory to model model driven engineering. *Dans Workshop on Software Model Engineering (WISME 2004)*, Lisbon, Portugal, Octobre 2004. (Cité pages 1 et 14.)

[7] Tivadar Szemethy et Gabor Karsai : Platform modeling and model transformations for analysis. *Journal of Universal Computer Science*, 10 (10):1383–1407, octobre 2004. (Cité page 2.)

[8] Alberto Sangiovanni-Vincentelli et Grant Martin : Platform-based design and software design methodology for embedded systems. *IEEE Computer Science*, 18(6):23–33, Novembre 2001. (Cité pages 2, 25 et 30.)

[9] Object Management Group : Response to the omg rfp for schedulability, performance, and time. OMG Document ad/2001-06-14, Juin 2001. (Cité pages 2, 30, 44 et 155.)

[10] Petri Kukkala, Jouni Riihimâki, Marko Hämäläinen et Klaus Kronlöf : Uml 2.0 profile for embedded system design. *Dans DATE'05*, Vol.2, pages 710–715. Automation and Test in Europe Conference (DATE 2005), Mars 2005. (Cité pages 2, 29 et 53.)

[11] Object Management Group : Uml profile for modelling and analysis of real-time and embedded systems (marte). OMG Document ptc/07-08-04, Août 2007. (Cité pages 3, 104, 112 et 115.)

[12] Yvon Trinquet et Jean-Pierre Elloy : Systèmes d'exploitation temps réel. *Dans Techniques de l'ingénieur*. Maurice POSTELS S.A., Mars 1999. (Cité pages 9 et 10.)

[13] Damien Chabrol : *Etude, conception et mise en oeuvre d'un protocole de communication synchrone tolérant aux fautes et prédictible sur des composants réseaux standards*. Thèse de doctorat, Université Paris Sud, Juin 2006. (Cité pages 9 et 25.)

[14] Jean-Philippe Babau : *Formalisation et structuration des architectures opérationnelles pour les systèmes embarqués temps réel*. Mémoire d'habilitation à diriger des recherches, Université Institut National des Sciences Appliquées de Lyon et Université Claude Bernard de Lyon 1, Décembre 2005. (Cité page 9.)

[15] Ahmad Badreddin Alkhodre : *Développement formel de systèmes temps réel à l'aide de SDL et IF*. Thèse de doctorat, Institut National des Sciences Appliquées de Lyon, Septembre 2004. (Cité page 9.)

[16] Safouan Taha : *Modélisation conjointe logiciel / matériel des systèmes temps réel*. Thèse de doctorat, Université de Lille, Mai 2008. (Cité pages 10 et 53.)

[17] Paul Caspi, Grégoire Hamon et Marc Pouzet : *Systèmes temps réel 1 : techniques de description et de vérification*, volume 1, chapitre Lucid Synchrone, un langage de programmation des systèmes réactifs, pages 217–260. Hermes, 2006. (Cité page 11.)

[18] Vincent David, Christophe Aussaguès, Stéphane Louise, Philippe Hilsenkopf, Bertrand Ortolo et Christophe Hessler : The oasis based qualified display system. *Dans Fourth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls and Human-Machine Interface Technologies (NPIC&HMIT 2004)*, Columbus, Ohio, USA, Septembre 2004. (Cité page 11.)

[19] OSEK/VDX Group : Osek/vdx operating system specification, version 2.2.3. Rapport technique, OSEK/VDX Group, http ://www.osek-vdx.org/, Février 2005. (Cité pages 11, 93, 116 et 147.)

[20] Airlines Electronic Engineering Committee : Avionics application software standard interface, arinc specification 653-1. Rapport technique, Aeronautical radio INC., Annapolis, Maryland, USA, October 2003. (Cité pages 11 et 104.)

[21] WindRiver : Vxworks programmer's guide, 5.4. Edition 1, http ://www.windriver.com/, Mars 1999. (Cité pages 11, 93 et 147.)

[22] Andrew S. Tanenbaum : *Modern Operating Systems*. Numéro ISBN 0-13-092641-8. Pearson International Edition, Upper Saddle River, NJ, USA, seconde édition, 2001. (Cité page 12.)

[23] Vincent Hugel : Cours des systèmes temps réel. Institut des Sciences et des Techniques des Yvelines (ISTY), Septembre 2008. (Cité page 12.)

[24] Frédéric Jouault : *Contribution à l'étude des langages de transformation de modèles*. Thèse de doctorat, Université de Nantes, 2006. (Cité pages 14, 16, 129, 143, 164 et 169.)

[25] Colin Atkinson et Thomas Kuhne : Model-driven development : A metamodeling foundation. *IEEE Software*, 20(5):36–41, 2003. (Cité page 14.)

[26] Edwin Seidewitz : What models mean. *Dans IEEE software*, volume 20, pages 26–32, Los Alamitos, CA, USA, Octobre 2003. IEEE Computer Society. (Cité page 14.)

[27] Jean Bézivin : On the unification power of models. *Dans Software and Systems Modeling*, volume 4, pages 171–188. Springer, May 2005. (Cité page 14.)

[28] *MDA guide version 1.0.1*. OMG, Juin 2003. (Cité pages 14, 26 et 27.)

[29] Thomas Kühne : Matters of (meta-) modeling. *Software and Systems Modeling (SoSyM)*, 5(4):369–385, December 2006. (Cité page 15.)

[30] Jean Bézivin : La transformation de modèles. Ecole d'Eté d'Informatique CEA EDF INRIA, cours n°6, Juin 2003. (Cité page 16.)

[31] Object Management Group : Mof qvt final adopted specification. http ://www.omg.org/docs/ptc/05-11-01.pdf, Novembre 2005. (Cité page 16.)

[32] Bertrand Nicolas : Les modèles dans l'action à france télécom avec SmartQVT^{TM}. *Dans NEPTUNE workshop 2008*, Paris, Avril 2008. (Cité page 16.)

[33] Jonathan Musset, Etienne Juliot et Stéphane Lacrampe : Acceleo^{TM} 2.2 : Guide utilisateur. Obeo, http ://www.acceleo.org/pages/accueil/fr, Avril 2008. (Cité pages 17 et 129.)

[34] Markus Völter et Thomas Stahl : *Model-Driven Software Development : Technology, Engineering, Management*. Numéro 978-0-470-02570-3. John Wiley & Sons, Juin 2006. (Cité page 17.)

[35] Frédéric Jouault, Jean Bézivin et Ivan Kurtev : Tcs : a dsl for the specification of textual concrete syntaxes in model engineering. *Dans GPCE '06 : Proceedings of the 5th international conference on Generative programming and component engineering*, pages 249–254, New York, NY, USA, 2006. ACM. (Cité page 17.)

[36] Pierre-Alain Muller, Frédéric Fondement, Franck Fleurey, Michel Hassenforder, Rémi Schnekenburger, Sébastien Gérard et Jean-Marc Jézéquel : Model driven analysis and synthesis of textual concrete syntax. *Dans Journal of Software and Systems Modeling (SoSyM)*, Avril 2008. (Cité page 17.)

[37] Object Management Group : Unified modeling language (uml) : Superstructure, version 2.1.2. Document formal/2007-11-02, Novembre 2007. (Cité pages 18 et 62.)

[38] James Bruck et Kenn Hussey : Customizing uml : Which technique is right for you ? Rapport technique, International Business Machines Corporation, 2007. (Cité page 18.)

[39] Gilles Kahn : The semantics of a simple language for parallel programming. *Dans IFIP Congress 74*, Holland, 1974. (Cité page 19.)

[40] Sébastien Gérard : *Modélisation UML exécutable pour les systèmes embarqués de l'automobile*. Thèse de doctorat, Université d'Evry, 2000. (Cité pages 19, 73 et 140.)

[41] C. Broecker et all : D1.2 architecture and platform modelling profiles. Rapport technique, Zühlke Engineering GmbH, May 2006. (Cité page 19.)

[42] Raphaël Marvie, Laurence Duchien et Mireille Blay-Fornarino : *Les Plate-formes d'exécution et l'IDM*, chapitre 4, pages 71–86. Numéro ISBN : 2-7462-1213-7. Hermes, January 2006. (Cité pages 25 et 27.)

[43] Colin Atkinson et Thomas Kühne : *A Generalized Notion of Platforms for Model-Driven Development*, volume 2, pages 119–136. Springer Verlag, 2005. (Cité pages 25 et 27.)

[44] Bran Selic : On software platforms, their modeling with uml 2, and platform-independent design. *Dans Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 15–21, Washington, DC, USA, 2005. IEEE Computer Society. (Cité pages 25 et 44.)

[45] João Paulo Andrade Almeida : *Model-Driven Design of Distributed Applications*. Computer science, Telematica Instituut Fundamental Research Series, Enschede, The Netherlands, 2006. ISBN 90-75176-422. (Cité page 26.)

[46] Jérôme Delatour, Frédéric Thomas, Guillaume Savaton et Sébastien Faucou : Modèle de plate-forme pour l'embarqué : première expérimentation sur les noyaux temps réel. *Dans Actes des premières journées sur l'Ingénierie Dirigée par les Modèles (IDM 2005)*, Paris, France, juin 2005. (Cité page 26.)

[47] Society of Automotive Engineer : Architecture analysis & design language (aadl) as5506, 2006. (Cité page 28.)

[48] Tero Arpinen, Mikko Setälä, Erno Salminen, Marko Hännikäinen et Timo D. Hämäläinen : Modeling embedded software platform with a uml profile. *Dans* Engenio Villar, éditeur : *Forum on Specification and Design Languages (FDL'07)*, Barcelona, Spain, September 2007. ECSI. (Cité page 29.)

[49] Alessandro Pinto : Metropolis design guidelines. Rapport technique, University of California, Berkeley, November 2004. (Cité page 30.)

[50] Metropolis Project Team : The metropolis meta model version 0.4. Rapport technique UCB/ERL M04/38, University of California, Berkeley, September 2004. (Cité page 30.)

[51] Rong Chen, Marco Sgroi, Grant Martin, Luciano Lavagno, Alberto Sangiovanni-Vincentelli et Jan Rabaey : *UML and Platform-Based Design*, volume UML for Real : Design of Embedded Real-Time Systems, chapitre 5. Kluwer Academic Publishers, May 2003. (Cité page 30.)

[52] Abhijit Davare, Douglas Densmore, Trevor Meyerowitz, Alessandro Pinto, Alberto Sangiovanni-Vincentelli, Guang Yang, Haibo Zeng et Qi Zhu : A next-generation design framework for platform-based design. *Dans Conference on Using Hardware Design and Verification Languages (DVCon)*, February 2007. (Cité page 31.)

[53] Tivadar Szemethy : *Domain-Specific Models, Model Analysis, Model Transformations*. Thèse de doctorat, Université de Vanderbilt, Nashville, Tennessee, USA, May 2006. (Cité pages 31 et 32.)

[54] Bertrand Meyer : *Object-Oriented Software Construction*. Numéro ISBN 0-13-629155-4. Prentice Hall, NJ, USA, seconde édition, Mars 2000. (Cité page 41.)

[55] Erich Gamma, Richard Helm, Ralf Johnson et John Vlissides : *Design patterns : Elements of reusable object-oriented software*. Addison Wesley, Boston, MA, USA, 1994. (Cité pages 42 et 169.)

[56] Bran Selic : Domain-specific modeling languages and uml profiles - an introduction. *Dans Third Edtion of the summer school MDD4DRES*, Brest, France, Septembre 2006. (Cité page 42.)

[57] François Lagarde, Huáscar Espinoza, François Terrier, Charles André et Sébastien Gérard : Leveraging patterns on domain models to improve uml profile definition. *Dans Fundamental Approaches to Software Engineering (FASE 08)*, pages 116–130, Budapest, Hongrie, Avril 2008. LNCS. (Cité pages 42, 50 et 123.)

[58] Jim Steel et Jean-Marc Jézéquel : On model typing. *Journal of Software and Systems Modeling (SoSyM)*, 6(4):452–468, dec 2007. (Cité page 43.)

[59] Antoine Colin et Isabelle Puaut : Worst-case execution time analysis of the rtems real-time operating system. *Dans Proc. of the 13th Euromicro Conference on Real-Time Systems*, pages 191–198, Delft, The Netherlands, June 2001. (Cité page 44.)

[60] Jean-Luc Bechennec, Michael Briday, Sébastien Faucou et Yvon Trinquet : Trampoline an open source implementation of the osek/vdx rtos specification. *Emerging Technologies and Factory Automation (ETFA '06)*, pages 62–69, September 2006. (Cité pages 44, 118, 131 et 164.)

[61] Rebecca Wirfs-Brock, Brian Wilkerson et Lauren Wiener : Responsibility-driven design : Adding to your conceptual toolkit. *Dans Report on Object Analysis and Design*, numéro ISBN 9781884842269, pages 27–34, Juillet 1994. (Cité pages 49 et 54.)

[62] Julio Luis Medina Pasaje, Medina González Harbour et Jose M. Drake : Mast real-time view : A graphic uml tool for modeling object-oriented real-time systems. *Dans Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, page 245, Londre, UK, Decembre 2001. IEEE Computer Society. (Cité page 50.)

[63] Huascar Espinoza : *An Integrated Model-Driven Framework for Specifying and Analyzing Non-Functional Properties of Real-Time Systems*. Thèse de doctorat, Université d'Evry, Saclay, France, Septembre 2007. (Cité page 50.)

[64] Frédéric Thomas, Sébastien Gérard et Jérôme Delatour : Towards an uml 2.0 profile for real-time execution platform modeling. *Dans 18th Euromicro Conference on Real-Time Systems, Work in progress session*, Dresden, Allemagne, Juin 2006. (Cité page 56.)

[65] Frédéric Thomas, Jérôme Delatour, Sébastien Gérard, Matthias Brun et François Terrier : Contribution à la modélisation explicite des plates-formes d'exécution pour l'idm. *Dans Ingénierie dirigée par les modèles*, volume 13 de *l'Objet- logiciel, bases de données, réseaux*, pages 9–31, Paris, France, Septembre 2007. Hermes-Lavoisier. (Cité page 56.)

[66] Giancarlo Guizzardi : On ontology, ontologies, conceptualizations, modeling languages, and (meta)models. *Artificial Intelligence and Applications, Databases and Information Systems IV*, 2007. (Cité page 59.)

[67] Bruce Powell Douglass : *Real-Time Design Patterns : Robust Scalable Architecture for Real-Time Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. (Cité page 61.)

[68] Patrick Albert, Laurent Henocque et Mathias Kleiner : A constrained object model for configuration based workflow composition. *Dans Revised and Selected Papers, Business Process Management - BPM 2005 Workshops*, volume 3812 de *Lecture Notes in Computer Science*, pages 102–115. Springer, janvier 2006. (Cité page 73.)

[69] Krzysztof Czarnecki et Ulrich W. Eisenecker : *Generative programming : methods, tools, and applications*. Numéro ISBN 0-201-30977-7. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, Decembre 2002. (Cité page 77.)

[70] Object Management Group : Object constraint language (ocl), Mai 2006. (Cité page 87.)

[71] ATLAS group : Atl user manual - version 0.7. LINA - INRIA, http ://www.eclipse.org/m2m/atl/doc/, Février 2006. (Cité page 87.)

[72] The Open Group Base Specification : Portable operating system interface (posix). AINSI/IEEE Std 1003.1, http ://www.opengroup.org/, 2004. (Cité pages 93 et 147.)

[73] United Business Media : Embedded systems survey : Operating systems up for grabs. http ://www.embedded.com/, May 2005. (Cité pages 93 et 147.)

[74] DIAPM : Realtime application interface for linux. Dipartimento di Ingegneria Aerospaziale Politecnico di Milano, https ://www.rtai.org/, October 2006. (Cité pages 93 et 147.)

[75] Groupe de travail SCEPTRE 2 : Rapport sur les services offerts aux applications temps réel à contraintes strictes par les exécutifs temps réel. Rapport technique, Groupe de travail SCEPTRE 2, 1992. (Cité pages 93 et 147.)

[76] Jean-Jacques Schwarz : Lacatre : Langage d'aide à la conception d'application multitâche temps reel. *Dans Revue d'automatique, Productique et informatique industrielle*, volume 26, pages 355–385. APII AFCET, 1992. (Cité pages 93 et 147.)

[77] Frédéric Thomas : Modèle de plate-forme pour l'embarqué : Première expérimentation sur les noyaux temps réels. Mémoire de D.E.A., Institut de Recherche en Communications et Cybernétique de Nantes, Septembre 2005. (Cité pages 93 et 147.)

[78] Nicolas Pernet, Sylvie Cauvinand, Frederic Thomas, Chokri Mraidha, Michel Sall et Philippe Robin : Uml modeling of a real-time embedded industrial system : an engine tested. *Dans 4th European Congress ERTS EMBEDDED REAL TIME SOFTWARE*, Toulouse, France, Juin 2008. (Cité pages 104 et 112.)

[79] Frédéric Thomas, Sébastien Gérard, Jérôme Delatour et Francois Terrier : Software real-time resource modeling. *Dans Forum on Specification and Design Languages (FDL'07)*, Barcelona, Spain, September 2007. ECSI. (Cité page 112.)

[80] Frédéric Thomas, Sébastien Gérard, Jérôme Delatour et Francois Terrier : *Software Real-Time Resource Modeling*, volume Embedded Systems Specification and Design Languages de *FDL selected papers*, chapitre 12, pages 169–182. Eungenio Villar, Barcelona, Spain, springer science+business media b.v. édition, September 2008. (Cité page 112.)

[81] Frédéric Thomas, Jérôme Delatour, François Terrier et Sébastien Gérard : Towards a framework for explicit platform based transformations. *Dans 11th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2008)*, Orlando, FL, USA, Mai 2008. (Cité pages 112 et 136.)

[82] Sébastien Demathieu, Frédéric Thomas, Charles André, Sébastien Gérard et François Terrier : First experiments using the uml profile for marte. *Dans 11th IEEE International Symposium on Object-oriented*

Real-time distributed Computing (ISORC 2008), Orlando, FL, USA, Mai 2008. (Cité page 112.)

[83] Frank Singhoff, Jérôme Legrand, Laurent Nana et Lionel Marcé : Cheddar : a flexible real time scheduling framework. *Dans SIGAda 2004*, numéro 4, pages 1–8, Atlanta, GA, USA, 2004. ACM. (Cité pages 118, 131 et 159.)

[84] Edward Tsang : *Foundations of Constraint Satisfaction*. Numéro ISBN 0-12-701610-4. Academic Press, 1996. http://cswww.essex.ac.uk/Research/CSP/edward/FCS.html. (Cité pages 125 et 169.)

[85] Marco Didonet Del Fabro : *Metadata management using model weaving and model transformation*. Thèse de doctorat, University of Nantes, Septembre 2007. (Cité page 135.)

[86] Object Management Group : Systems modeling language, version 1.0. document formal/07-09-01, September 2007. (Cité page 155.)

[87] Object Management Group : Software process engineering metamodel, version 2.0. Document formal/2008-04-01, Avril 2008. (Cité page 155.)

[88] Hervé Albin-Amiot : *Idiomes et patterns Java : application à la synthèse de code et à la détection*. Thèse de doctorat, Université de Nantes, Février 2003. (Cité page 169.)

[89] Yann-Gaël Guéhéneuc : *Un cadre pour la traçabilité des motifs de conception*. Thèse de doctorat, Université de Nantes, Juin 2003. (Cité page 169.)

[90] Ilka Philippow, Detlef Streitferdt, Matthias Riebisch et Sebastian Naumann : An approach for reverse engineering of design patterns. *Software and System Modeling*, 4(1):55–70, 2005. (Cité page 169.)

[91] Dae K. Kim et Lunjin Lu : Inference of design pattern instances in uml models via logic programming. *Engineering of Complex Computer Systems, 2006. ICECCS 2006. 11th IEEE International Conference on*, 2006. (Cité page 169.)

Glossaire

API acronyme anglais de Application Programming Interface désignant l'interface de programmation des applications, c'est-à-dire l'ensemble des primitives permettant à un programme d'utiliser les ressources d'un système.

ARINC-653 acronyme anglais désignant un standard des systèmes d'exploitation temps réel embarqués destinés à l'industrie avionique. Ce standard est promu par la société ARINC.

ARINC acronyme anglais de la compagnie Aeronautical Radio, Incorporated détenue par des compagnies aériennes et des constructeurs aéronautiques américains. Elle a pour objectif de définir les principaux standards de communications à l'intérieur des aéronefs, entre les aéronefs et entre les aéronefs et le sol.

ASIC acronyme anglais de Application-Specific Integrated Circuit déignant un circuit intégré électronique à application spécifique, par opposition au circuit intégré standard. Un ASIC réalise une fonction bien précise, définie par le concepteur d'une carte électronique. Cette fonction est ensuite intégrée dans une puce de silicium par un fondeur.

ATL acronyme anglais de l'initiative Atlas Transformation Language désignant un langage de transformation de modèle.

Cheddar acronyme d'un outil éducatif ouvert dédié à l'analyse d'ordonnancement des systèmes temps réel. Cet outil est développé dans le Laboratoire d'Informatique des Systèmes Complexes (LISyC) rattachée à l'université de Brest.

EMF acronyme anglais de Eclipse Modeling Framework signifiant environnement de modélisation pour Eclipse.

Eclipse plate-forme logicielle ouverte pour le développement d'applications développée par la fondation du même nom.

FIFO acronyme de Fist In First Out désignant la politique d'une file d'attente dans laquelle le premier arrivé est le premier servi.

FPGA acronyme anglais de Field Programmable Gate Array désignant un réseau de portes (logiques) programmables in-situ, c'est-à-dire un circuit intégré programmable.

HLAM acronyme du sous profil High-Level Application Modeling de MARTE dédié à la description à un haut niveau d'abstraction des applications multitâches.

IDM acronyme français de Ingénierie Dirigée par les Modèles, synonyme de IDM et équivalent MDE : Model Driven Engineering en anglais.

LACATRE acronyme français de Langage d'Aide à la Conception d'Applications multitâches Temps REel désignant un langage facilitant la conception d'applications basées sur les exécutifs multitâches temps réel. Ce langage est développé par le laboratoire CITI de l'Institut National des Sciences Appliquées (INSA) de Lyon.

LIFO acronyme désignant Last In Last Out désignant la politique d'une file d'attente dans laquelle le dernier arrivé est le premier servi.

MARTE acronyme anglais de Modeling and Analysis of Real-time and Embedded system signifiant modélisation et analyse des systèmes temps réel embarqués. Il s'agit d'une recommandation de l'OMG.

MDA acronyme anglais de Model Driven Architecture®, signifiant architecture dirigée par les modèles. Le MDA est promu par l'OMG pour mettre en application une ingénierie dirigée par les modèles s'appuyant sur ses standards.

OASIS acronyme français de Objets Actifs Synchrones Isochrones pour la Sûreté désignant un projet de recherche du CEA List. Ce projet offre un cadre rigoureux permettant d'effectuer la conception multitâche temps-réel déterministe d'une application de sûreté.

OCL acronyme anglais de Object Constraint Language signifiant langage de contraintes sur les objets. Il s'agit d'une recommandation de l'OMG définissant un langage d'expression de requêtes et de contraintes sur des modèles.

OIL acronyme de OSEK Implementation Language désignant un langage de description dans lequel les objets du système d'exploitation Osek/Vdx-OS sont décrits (les tâches, les interruptions, les ressources et les alarmes par exemple).

OMG acronyme anglais de Object Management Group représentant un consortium d'acteurs industriels et académiques pour l'intégration de standards dans l'industrie. UML et OCL sont des standards de l'OMG par exemple.

Osek/Vdx-OS acronyme du système d'exploitation temps réel standardisé par le consortium OSEK/VDX et lancé en 1997. Il est principalement utilisé pour les systèmes embarqués de l'industrie automobile.

PCP acronyme de Priority Ceiling Protocol (Protocole de priorité plafond), protocole utilisé dans les systèmes multitâches pour la gestion des accès aux ressources partagées. Dans ce protocole la tâche accédant à la ressource prend la priorité plafond. Cette priorité est possédée par la ressource. Elle correspond à la priorité de la tâche de plus haut priorité pouvant accéder à la ressource.

PIP acronyme de Priority Inheritance Protocol (Protocole d'héritage de priorité), protocole utilisé dans les systèmes multitâches pour la gestion des accès au ressources partagées. Dans ce protocole, une tache B qui possède la ressource prend la priorité de la tache A en attente de la ressource si la priorité de A est supérieur à la priorité de B.

POSIX acronyme anglais de Portable Operating System Interface désignant le nom d'une famille de standards définie depuis 1988 par l'IEEE et formellement désignée IEEE 1003. Ces standards ont émergé d'un projet de standardisation des APIs des logiciels destinés à fonctionner sur des variantes du système d'exploitation UNIX. Le X de l'acronyme exprime l'héritage UNIX de l'API.

QVT acronyme anglais de Query/View/Transform signifiant requête/vue/ transformation. C'est le nom donné par l'OMG à sa recommandation MDA pour la transformation de modèles.

RTAI acronyme anglais de Real Time Application Interface désignant un correctif des systèmes d'exploitation Linux pour les utiliser dans des applications multitâches temps réel embarquées.

SCEPTRE acronyme français de Standardisation du Coeur des Exécutifs des Produits Temps Réel Européens désignant le résultat du projet de recherche de même nom. Ce projet visait à unifier les interfaces de programmation des systèmes d'exploitation temps réel embarqués.

UML acronyme anglais de Unified Modeling Language signifiant langage de modélisation unifié. Il s'agit d'une recommandation de l'OMG pour la modélisation des systèmes.

VxWorks acronyme du système d'exploitation temps réel commercial distribué par la société WindRiver.

XMI acronyme anglais de Xml Model Interchange signifiant échange de modèles en XML. C'est le nom donné par l'OMG à sa recommandation MDA pour la sérialisation des modèles dans les fichiers informatiques.

XML acronyme anglais de eXtensible Markup Language désignant est un format universel de stockage et d'échange de données.

XQuery acronyme anglais de XML Query Language désignant un langage de requête XML.

XSLT acronyme anglais de eXtentsible Style Language Transformations est un langage XML permettant de formuler des transformations sur un ou plusieurs documents XML sources pour produire un ou plusieurs documents XML cibles.

Index

– S –

– T –

– U –

– V –

Ce document a été préparé à l'aide de l'éditeur de texte Vim et du logiciel de composition typographique LaTeX 2ε. Des schémas ont été réalisés à l'aide de la bibliothèque LaTeX 2ε TikZ & PGF.

► **Titre –** Contribution à la prise en compte des plates-formes logicielles d'exécution dans une ingénierie générative dirigée par les modèles

► **Résumé –** Face à la complexité inhérente des applications logicielles multitâches, une approche prometteuse est l'automatisation des développements. En pratique, cette automatisation se concrétise par des générateurs capables de produire les implantations s'exécutant sur des plates-formes logicielles multitâches (par exemple des systèmes d'exploitation multitâches). Ces générateurs sont ainsi spécifiques aux plates-formes visées. Ils sont figés autour d'heuristiques d'implantations propres à chacune de ces plates-formes. En vue d'obtenir des solutions plus adaptables et plus flexibles, cette étude a pour objectif d'externaliser les formalismes propres aux plates-formes d'exécution dans des modèles explicites. Ces modèles sont alors utilisés en entrée des ingénieries de développement ce qui permet de capitaliser et de réutiliser les générateurs. Pour y parvenir, cette étude définit premièrement un motif générique dédié à la modélisation des plates-formes logicielles d'exécution, deuxièmement, une extension au langage UML destiné à la modélisation des plates-formes logicielles d'exécution multitâche (le profil Software Resource Modeling) et, troisièmement, une infrastructure de transformation intégrant des modèles de plates-formes explicites. Les deux premières contributions constituent le profil UML pour la modélisation et l'analyse des systèmes embarqués temps réel (MARTE).

► **Mots-clés –** Ingénierie Dirigée par les Modèles, Modèle de Plateforme, Transformation de modèle, Exécution multitâche, UML, MARTE

► **Title –** Contribution to the software execution platform integration in a generative model driven engineering

► **Abstract –** To minimize the inherent complexity of multitasking programs, a promising approach is to automate developments. In practice, automation is achieved by generators. Those generators produce applications which execute on software multitasking platforms (for example multitasking operating systems). Such generators are in fact specific to selected platforms. They are made of implementation rules which are specific to each platform. In order to cope with adaptable and flexible solutions, this study aims to explicitly describe executing platforms as parameters of generators. For that, it defines, firstly, a generic pattern dedicated to modelling software execution platforms, secondly, an extension to the UML language for modelling multitasking software execution platforms (the Software Software Resource Modeling profile) and, thirdly, a transformation framework based on explicit platform models. The two first contributions are part of the UML profile for Modelling and Analysis of Real-time and Embedded systems (MARTE).

► **Keywords –** Model Driven Engeneering, Platform model, Model transformation, Multitasking execution, UML, MARTE