Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

BADJI MOKHTAR-ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR-ANNABA

جامعة باجي مختار - عنابة

Faculté des Sciences de l'Ingénieur
Département d'Informatique

Année : 2009/2010

Mémoire

Présenté en vue de l'obtention du diplôme de

Magister en Informatique

Option : Informatique Industrielle

Un Modèle Logique pour les Architectures Logicielles dans l'Embarqué

Par
Yacine Djebar

Devant le Jury

Président	:	Rachid BOUDOUR	M. de Conf. Univ. Annaba
Rapporteur	:	Mohamed T. KIMOUR	M. de Conf. Univ. Annaba
Examinateurs	:	Salim GHANEMI	M. de Conf. Univ. Annaba
		Nacira GHOUALMI	M. de Conf. Univ. Annaba

الملخص

الإشكاليات الجديدة التي فرضها اختراق تكنولوجيا المعلومات لعالم الصناعة و على الخصوص المتعلقة منها بالأنظمة المحمولة تتركز حول النمو المذهل و المعقد للبرمجيات. من بين الحلول الأحسن تكيفا لعملية تصنيع البرمجيات برزت تقنية التصميم و التطوير ضمن إطار خطوط المنتجات.

هذه التقنية تأخذ بعين الاعتبار القواسم المشتركة – المشتركية – و القواسم المتغيرة – التباينية – المتعلقة بالبرمجيات التي تتيح تخفيض تكاليف هذه الأخيرة، و مدة إنجازها و المدة اللازمة لتسويقها.

تتموقع هذه الرسالة في سياق نمذجة خطوط المنتجات البرمجية بواسطة لغة UML و تقترح مناهج جديدة لاستعمالها من خلال النشاط الرئيسي لهذه التقنية ألا و هو التباينية.

من خلال هذه الرسالة، سوف نقوم بتقديم نظام نمذجة و تسيير للتباينية الهيكلية داخل خطوط المنتجات باستعمال مصطلح SGV (نظام تسيير التباينية).

يرتكز SGV على أربعة مساهمات مرتبطة بالبعد التبايني ألا و هي :

- توفير تمثيل أكثر دقة بالنسبة للتباينية **الإختيارية** (**أو الغير إلزامية**) حتى يتسنى الأخذ بعين الاعتبار قابلية التمديد للسمات المنمذجة .
- أيتاح تمثيل أفضل للتباينية المرتبطة بالخصائص و العمليات.
- أيتاح للمصمم امكانية تمثيل و تسيير تطور التباينية الخاصة بالمتغيرات على مستوى الأقسام من خلال استخدام مفهوم << الدور >>.
- توفير إمكانية وصف مختلف نماذج و إنجازات التباينات التصميمية من خلال التفصيلات الهيكلية لخطوط المنتجات باستعمال آليات نابعة كليا من لغة UML. تتيح هذه الآليات تسييرا أفضل لتباينية البرمجيات المنتجة و ذلك حسب واجهات تصميمية مختلفة.

RESUME

Les nouvelles problématiques introduites par la pénétration de l'informatique dans l'industrie notamment celles relative aux systèmes embarqués sont axées sur la croissance et la complexité des logiciels. L'une des solutions les plus prometteuses à cette industrialisation du logiciel est la conception et le développement dans le cadre de lignes de produits logiciels. Cette technique prend en compte les facteurs communs (commonalité) et variants (variabilité) des logiciels et permettent de minimiser les coûts, les temps de réalisation et les délais de mise sur le marché.

Ce travail se situe dans le contexte de la modélisation des lignes de produits logiciels en UML, et propose de nouvelles approches pour leur manipulation à travers l'activité primordiale de cette discipline, en l'occurrence, la variabilité.

Dans ce mémoire, nous présentons un système de modélisation et de gestion de la variabilité architecturale dans les lignes de produits. Ce système s'articule autour de quatre contributions associées à la dimension variabilité:

i. *Plus raffiné dans notre modèle, le concept d'optionalité prend en compte l'extensibilité de la feature modélisée.*
ii. *Le modèle permet une meilleure représentation de la variabilité des attributs et des méthodes.*
iii. *Le modèle à travers l'usage des roles permet de modéliser et de gérer l'évolution des variants au niveau des classes variables.*
iv. *Le modèle en offrant la possibilité de décrire les différentes représentations et réalisations des variabilités conceptuelles à travers les décompositions architecturales des lignes de produits par l'usage de mécanismes UML, permet une meilleure gestion de la variabilité des produits selon les différentes vues UML.*

ABSTRACT

The penetration of computer science into industry, in particular that relating to the embedded systems, has induced new problems, which are especially inherent to the growth and the complexity of the software. One of the most promising solutions to this software industrialization is the design and the development within the context of software product lines. This technique takes into account the common factors (commonality) and variable factors (variability) of the software and makes it possible to minimize costs, implementation time and time to market.

This work is conducted in the context of the software modeling product lines in UML, and proposes new approaches for their handling through the first activity of this discipline which is variability. In doing so, we present an architectural management and modeling variability approach for the product lines. Such an approach is articulated around four contributions associated with variability dimension:

- *More refined in our model, the optionality concept will take into account the extensibility of modeled feature.*
- *The model allows a better representation of the attributes and methods variability.*
- *The model on using of the role concept allows modeling and managing the variability evolution in the variable classes.*
- *The model offers the possibility to describe the different representations and achievements of conceptual variability through the architectural decompositions of the product lines by using of UML mechanisms .It allows a better product variability management according to UML views.*

…………..

A ma Mère,
Mon père,
Mes frères et sœurs,
Et en particulier, *à mon frère Réda et à son épouse*

A mon épouse,
Et Mes enfants,

Et à titre posthume
Nabila, Masria , Aida *et Mes grand- parents*

REMERCIEMENTS

Mes premiers remerciements vont naturellement à Mr Mohamed Tahar Kimour, Maître de Conférences à l'Université d'Annaba, mon directeur de Mémoire, qui, par son encadrement pendant ces deux années, m'a permis de découvrir l'antichambre du milieu de la recherche. Je tiens également à le remercier pour tout ce qu'il a bien voulu (et veut bien encore) me procurer comme soutien, aussi bien sur le plan scientifique que sur le plan moral. Qu'il sache que sa gentillesse, son entière disponibilité, ses précieux conseils et ses encouragements m'ont profondément marqué et m'ont permis de continuer à mener l'ensemble de ce travail à bien, y compris durant les inévitables périodes de doute. Qu'il sache également que je mesure bien la chance que j'ai eu de l'avoir pour directeur de Mémoire et quel formidable directeur il fût pour moi. Sa grande rigueur scientifique, son encadrement de qualité exceptionnelle ainsi que son perpétuel souci quant à mon avenir professionnel ont grandement contribué à l'accomplissement de cette page de ma vie. Pour tout cela, et pour tout le reste, je tiens à le remercier le plus chaleureusement possible.

Ma plus grande gratitude va également à tous mes enseignants et en particulier à Mr Rachid Boudour, Maitre de Conférences à l'Université d'Annaba et responsable de notre P.G qui m'a toujours encouragé et donné de précieux conseils pour poursuivre ce travail et aller le plus loin possible dans mes études en dépit de mes nombreux engagements .

Je tiens à remercier également et chaleureusement Mr Salim Ghanemi, Maitre de conférences à l'université d'Annaba, et Mme Nacira Ghoualmi, Maitre de conférences à l'université d'Annaba qui m'ont fait le grand honneur de bien vouloir évaluer mon mémoire de Magister et apporter leur précieux jugements. Je veux également les remercier pour leur disponibilité et leur immense gentillesse.

Mes remerciements vont également à Mes camarades de promotion et en particulier à Ahlem ; Ahcene , Mourad et tous les autres…

Je tiens aussi à remercier du plus profond de mon cœur et de mon âme tous les membres de ma famille, ***et notamment ma mère. Qu'elle sache que sa présence à mes côtés tout au long de ma vie a été des plus précieuses. J'espère sincèrement pouvoir lui rendre un peu de tout ce qu'elle a fait pour moi depuis mon plus jeune âge.***

Je remercie également Mon épouse et mes enfants qui ont partagé et supporté avec moi les charges et les engagements de ces dernières années d'études. Qu'ils sachent aussi que leurs présences à mes côtés tout au long de ces années a été des plus réconfortantes.

Je ne pourrais terminer sans oublier de remercier Mon directeur Mr Taabache, Mes collègues de travail Chahira , Nacira , Ahcene, les deux Samia, Souhila, Nassima, Moufida , Ami Moussa qui ont fait de leur mieux pour prendre en charge les différentes tâches du service lors de mes absences à l'université…...

Table des Matieres

Introduction.
Contributions.
Plan du document.
Table des Matieres.
Liste des figures.
Liste des tableaux.

I –Première partie. Etat de l'art.

Chapitre 1. Les systèmes embarqués.

1.1 Introduction …………………………………………………………………… 2

1.3 Les classes de systèmes embarqués ……………………………………………….. 4
1.3.1 Les systèmes embarqués dans l'automobile …………………………… 4
1.3.2 Les systèmes embarqués dans l'avionique …………………………… 6
1.3.2.1 L'avionique civile ………………………………………… 6
1.3.2.2 L'avionique militaire ……………………………………… 7
1.3.3 Les systèmes embarqués dans la robotique …………………………….. 8
1.3.4 Les systèmes embarqués dans la domotique …………………………… 8
1.3.4.1 Les domaines de la domotique ……………………………… 8
1.3.4.2 Techniques de la domotique ………………………………… 9
1.3.4.3 La domotique pour l'Assistance à la vie quotidienne ……… 9
1.3.4.4 La compensation des situations de handicap ou de dépendance ……………………………………………….. 10
1.3.5 Les systèmes embarqués dans les autres secteurs de la technologie …… 10
1.3.6 Autres domaines d'applications ……………………………………….. 11
1.4 Exigences et contraintes …………………………………………...................... 11
1.4.1 Contraintes de temps ………………………………………………….. 12
1.4.2 Consommation énergétique …………………………………………… 12
1.4.3 Mémoire ……………………………………………………………… 13
1.4.4 Tolérance aux fautes ………………………………………………….. 13
1.5 Architecture …………………………………………………………………... 13
1.6 Développement des systèmes embarqués ……………………………………….. 14

Chapitre 2. UML pour la modélisation des systèmes embarqués

2.1 Introduction …………………………………………………………………… 16
2.2 Les mécanismes d'extension d'UML pour les systèmes embarqués ………………. 16
2.3 Les profils UML pour les systèmes embarqués ……………………………………... 16
2.3.1 Définitions ……………………………………………………………… 16
2.3.2 Etat de l'art des profils UML pour l'embarqué ………………………… 17
2.3.2.1 SYSML ……………………………………........................ 17
2.3.2.2 MARTE ………………………………………………… 19
2.3.2.3 Le langage AADL ………………………………………... 23
2.3.2.4 UML et systemC ………………………………………... 27
2.4 Le développement des Systèmes embarqués dans le cadre des lignes de produits … 35
2.4.1 Introduction ……………………………………………………………. 35
2.4.2 Les Principales problématiques d'une approche LdP ………………… 36
2.4.3 Les Lignes de Produits Logiciels et la variabilité ..…………………….. 36
2.4.4 La variabilité logicielle ………………………………………………….. 38
2.5 UML pour la modélisation de la variabilité des LdP ……………………………… 40
2.5.1 Introduction ………………………………………………………… 40
2.5.2 État de l'art sur la modélisation des LdP en UML …………….. 40

II Deuxième partie. Contributions

Chapitre 3. Un sous-modèle pour l'optionalité et la variabilité des attributs et méthodes.

3.1 Introduction 44
3.2. Prise en compte de la variabilité dans notre sous-modèle 44
3.2.1 Première insuffisance 46
3.2.2 Deuxième insuffisance 47
3.2.3 Troisième insuffisance 47
3.2.4 Quatrième insuffisance 48
3.2.5 Solution complète 49
3.3 Discussions 50
3.4 Intégration à UML des mécanismes d'extension proposés 51
3.4.1 L'architecture d'UML 51
3.4.2 Les contraintes OCL 53
3.4.3 Les diagrammes 53
3.4.4 Les paquetages logiques du méta-modèle d'UML 54
3.4.5 L'extensibilité à UML 55
3.4.5.1 Intégration des stéréotypes « optional strict » et « optional accessory » 55
3.4.5.2 Intégration des stéréotypes « individual » et « collective » 56
3.5 Conclusion 57

Chapitre 4. Un sous-modèle d'expression de la variabilité par les roles.

4.1 Introduction 59
4.2 Eléments du sous-Modèle 59
4.2.1 Règles du sous-modèle 59
4.2.2 Exemple d'expression de la variabilité dans une classe variable 60
4.2.3 Prise en compte de l'évolution des variants par les roles 61
4.3 Intégration du sous-modèle à UML 63
4.3.1 Intégration de l'association role 63
4.3.1.1 Introduction du concept de role 63
4.3.1.2 Expression de la variabilité par le changement dynamique des variants 66
4.3.1.3 Expression Des contraintes de transitions à travers les prédicats 66
4.3.1.4 Intégration de l'association role dans les méta modèles UML 67
4.3.2- Intégration de la machine à états 70
4.4 Vers un profil UML pour les lignes de produits logiciels 71
4.5 Conclusion 72

Chapitre 5. Application au système automobile

5.1 Introduction 74
5.2 Application des solutions apportées par le sous-modèle 74
5.2.1 Pour l'optionalité 75
5.2.2 Pour les extensions par les stéréotypes au niveau des classes ... 75

Chapitre 6. Un sous-modèle pour la gestion de la variabilité architecturale selon les vues conceptuelles.

6.1 Introduction ………………………………………………………………………….. 80
6.2 Représentation de la variabilité architecturale dans notre sous-modèle 81
6.2.1 Le Processus global 83
6.2.2 Les éléments du sous-modèle ……………………………………….. 83
6.2.3 Actions sur les éléments du sous-modèle …………………….... 83
6.2.3.1 Compositions à granularité fine …………………….. 83
a. Aspect statique ……………………………………….. 83
b. Aspect dynamique …………………………………… 84
6.2.3.2 Compositions à gros grains ………………………… 85
6.2.4 Les Dépendances ……………………………………………… 87
6.3 Conclusion ………………………………………………………………………… 87

Chapitre 7. Etude du système « maison intelligente ».

7.1 Présentation du système de contrôle d'une maison intelligente …………………….. 89
7.2 Description des compositions à granularité fine 90
7.3 Prise en compte des compositions à granularité fine ……………………………… 92
7.4 Description et prise en compte des compositions à gros grains …………………….. 94

III Troisième partie. Conclusion et perspectives **…………………..** 96

References.

Table des figures

Figure	Titre	Page
Figure 1	Contributions de notre modèle de représentation de la variabilité dans les LdP.	préambule
Figure 2	Plan du document.	préambule
Figure 1.1	Exemple de systèmes embarqués dans l'automobile.	4
Figure 1.2	Systèmes embarqués dans un avion.	6
Figure 1.3	Systèmes enfouis dans un aéronef militaire	7
Figure 2.1	Comparaison de SysML 1.0 avec UML2.1.	18
Figure 2.2	Modèle de temps dans MARTE.	22
Figure 2.3	Flot de données UML / AADL	23
Figure 2.4	Les contraintes du profil UML	26
Figure 2.5	Exemple de profil UML pour AADL	26
Figure 2.6	Transformation à partir d'AADL vers le profil UML	27
Figure 2.7	Flot de conception systemC.	29
Figure 2.8	Organisation de SystemC	29
Figure 2.9	Un exemple de structure en SystemC	31
Figure 2.10	Flot de transformation UML vers SystemC.	32
Figure 2.11	Le mapping SystemC-UML.	32
Figure 2.12	Exemple de modélisation SystemC-UML de l'architecture matérielle d'un composant d'un simple bus.	33
Figure 2.13	Exemple de modélisation d'un simple bus	34
Figure 2.14	L'ingénierie des lignes de produits.	37
Figure 2.15	Exemple d'un diagramme de features FODA –système automobile-	37
Figure 2.16	Les dimensions temps et espace de la variabilité.	39
Figure 3.1	Notre Modèle d'expression de la variabilité dans le diagramme de classes	45
Figure 3.2	Modèle logique *–système d'allocation de ressources-*	46
Figure 3.3	Le Système *allocation de ressources* avec une nouvelle classe systemthread	46

Figure 3.4	Introduction des concepts : attributs collectifs et méthodes collectives / attributs individuels et méthodes individuelles.	48
Figure 3.5	Structure d'un dictionnaire de la variabilité	49
Figure 3.6	Variabilité dans la classe (entités prises en compte : la classe, attributs et méthodes)	49
Figure 3.7	Modèle logique final	50
Figure 3.8	L'architecture à quatre niveaux de L'OMG –exemple du système de surveillance d'un Moteur -	52
Figure 3.9	L'architecture détaillée à 4 niveaux de d'UML – exemple du système de surveillance d'un Moteur -	52
Figure 3.10	Les sous paquetages du paquetage Foundation du méta modèle M2	54
Figure 3.11	Extrait du méta modèle des classes d'UML	55
Figure 3.12	Les stéréotypes « Optional strict » et « Optional accessory»	56
Figure 3.13	Les stéréotypes « Individual », « Collective » et « Optional »	57
Figure 4.1	Un exemple de notre modèle de role.	60
Figure 4.2	Modèle de role amélioré -prise en compte des contraintes-.	60
Figure 4.3	Exemple d'interdépendance entre 2 roles	62
Figure 4.4	Sémantique d'une super classe optionnelle	63
Figure 4.5	Un exemple de l'association role.	65
Figure 4.6	Le méta-modèle de l'association role.	67
Figure 4.7	Exemple de roles.	68
Figure 4.8	Un exemple d'une machine à états qui référence une machine virtuelle CD pour la partition CD4.	70
Figure 4.9	Exemple de spécification d'un profil UML et son utilisation.	71
Figure 4.10	Profil « Software PL » pour les lignes de produits logiciels.	72
Figure 5.1	Une partie du diagramme de features pour le système automobile	74
Figure 5.2	Modèle logique diagramme de classes système automobile.	74
Figure 5.3	Modèle logique de la classe Moteur .	76
Figure 5.4	Dictionnaire de la variabilité du sous-système Moteur.	77
Figure 5.5	Le système moniteur de surveillance du moteur.	77
Figure 5.6	Diagramme états transitions montrant l'évolution des roles.	78
Figure 5.7	Le sous diagramme des états transitions correspondant à la classe variation	78

Figure 6.1	Le processus général du sous-modèle	81
Figure 6.2	Le sous-modèle conceptuel global.	82
Figure 6.3	Fonctionnement du sous-modèle	85
Figure 6.4	Aspect statique de notre sous-modèle.	86
Figure 6.5	Aspect dynamique du sous-modèle selon la vue séquence.	87
Figure 6.6	Vue conceptuelle de la sémantique de fusion de packages par l'association « merge » d'UML	87
Figure 7.1	Le diagramme de features pour le système contrôle maison intelligente	89
Figure 7.2	Diagramme des composants pour le système contrôle maison intelligente.	91
Figure 7.3	Exécution de la séquence « h ».	93
Figure 7.4	Système contrôle maison intelligente - Compositions à gros grains-	94

Liste des tableaux

Tableau	**Titre**	**Page**
Tableau 1.1	Les nouveaux services contrôlés par le logiciel dans l'automobile	5
Tableau 2.1	Eléments du profil UML	25
Tableau 2.2	Récapitulatif des principaux travaux sur la modélisation de la variabilité avec UML.	42
Tableau 3.1	Mapping applicable aux classes.	50
Tableau 3.2	Mapping applicable aux attributs et méthodes	50
Tableau 4.1	Récapitulatif des principaux cas modélisant l'évolution des Variant-roles.	64

Introduction

Les nouvelles problématiques introduites par la pénétration de l'informatique dans l'industrie sont axées sur la croissance et la complexité des logiciels. L'une des solutions les plus adaptées à cette industrialisation du logiciel est la conception et le développement dans le cadre de lignes ou de familles de produits logiciels. Cette technique prend en compte les facteurs communs et variants des logiciels et permet de minimiser les coûts, les temps de réalisation et les durées de mise sur le marché.

Les facteurs de variation des logiciels peuvent être techniques (utilisation d'une gamme de matériels associés aux logiciels selon certains paramètres), commerciaux (création de plusieurs versions du logiciel produit allant d'une version fonctionnant en monoposte par exemple à une version réseau), ou culturels (logiciels adaptés selon les pays). Pour illustration, on peut citer les logiciels intégrés aux téléphones mobiles qui doivent supporter plusieurs standards de communication et une variété de langues.

Cependant les lignes de produits n'ont émergé comme une approche à part entière que durant cette dernière décennie, lorsqu'ils ont commencé à regrouper une communauté à travers les différents projets européens tels que ESAPS [1], CAFE [2], FAMILIES [3]. Aux États Unis, Le Software Engineering Institute a également crée une section spéciale qui s'intéresse à l'ingénierie des lignes de produits [4].

Dans la littérature, il y a un consensus sur la définition d'une ligne de produits logiciels (LdP). Elle est définie comme un ensemble de systèmes partageant un ensemble de propriétés communes et satisfaisant des besoins spécifiques pour un domaine particulier [5,6]. La notion de variabilité est utilisée pour regrouper les caractéristiques qui différencient les produits de la même famille. Les langues supportées sont un exemple de variabilité dans une LdP du domaine des téléphones mobiles.
La gestion de cette variabilité est l'activité principale pour le développement des lignes de produits. La deuxième activité concerne la construction d'un produit particulier (on parle aussi de dérivation de produit) qui consiste en particulier à figer certains choix vis-à-vis de la variabilité définie dans la LdP. Une particularité des LdP est que certains choix sont incompatibles entre eux, d'autres sont liés. Un choix particulier lors de la dérivation d'un produit peut exclure ou exiger d'autres choix. La dernière activité est la gestion des contraintes permettant de faciliter les choix lors de la dérivation.

Plusieurs travaux se sont intéressés au développement des LdP notamment au niveau du code et ce, par l'usage de diverses techniques comme celle des objets [7,8, 9, 10] et la programmation générative [11,12,13,14]. Même si ces travaux se basent sur des techniques qui ont connu une grande réussite, la maîtrise du code devient de plus en plus difficile avec la croissance de la complexité des logiciels (les systèmes récents utilisent des millions de lignes de code).
Indépendamment du code, la modélisation semble une solution adéquate pour maîtriser cette complexité. Il ne s'agit plus de manipuler le code du système lui même mais plutôt de manipuler un ensemble de modèles qui le décrivent d'une manière plus abstraite. La modélisation est le fondement de plusieurs méthodes d'analyse et de conception tel le langage UML (Unified Modeling Language) [15, 16, 17] qui s'est imposé avec le temps comme un standard industriel pour la modélisation, et par la suite pour l'ingénierie des modèles [18] ou Model Driven Engineering (MDE).

Ce travail se situe dans le contexte de la modélisation de lignes de produits logiciels en UML, et propose de nouvelles approches pour leur manipulation à travers l'activité primordiale de cette discipline qu'est la variabilité.

Comme nous le verrons par la suite, il existe dans la littérature de très nombreux travaux autour de la manipulation des LdP en UML. La principale hypothèse de la plupart de ces travaux concerne l'utilisation des mécanismes standards d'extension d'UML pour permettre la modélisation des LdP et prendre en compte les nouvelles dimensions de la modélisation de la variabilité, de la dérivation de produits et de la gestion des contraintes. Mais en étudiant de plus près ces travaux, des constats d'insuffisances se dégagent et indiquent que la manipulation de LdP en UML manque de maturité :

- Le premier constat est que la majorité des travaux existants utilisent les vues statiques d'UML pour modéliser les LdP. Cependant, peu de travaux font référence à l'aspect dynamique des LdP.

- Le second constat est que la majorité des travaux existants ne prennent pas en compte l'extensibilité des features optionnelles (le fait de connaitre si dans la ligne de produits un point d'extension de la feature optionnelle a été prévu ou non) .Ce paramètre peut influencer sur l'évolution de la LdP .

- Le troisième constat est que l'étude de la variabilité au niveau des attributs et méthodes ne couvre pas pleinement tous les aspects dont les concepteurs et les développeurs de LdP ont besoin mais se limitent à identifier la variabilité des attributs et méthodes par rapport à celle des classes auxquelles elles appartiennent.

- Le contrôle de l'évolution de la variabilité des variants relevant des points de variation n'est pas pris en charge convenablement .Il est laissé à l'appréciation du concepteur qui doit lui-même trouver des solutions pour sa gestion.

- La gestion de la variabilité architecturale à travers sa représentation sous des formes hétérogènes dans différentes vues est prise en compte dans quelques travaux seulement. Ces travaux utilisent soit une documentation soit carrément un langage pour gérer cet aspect important des LdP. Il n y a pas de propositions de modélisation de cet aspect à l'aide de mécanismes UML capables de référencer des points de variation à travers des vues architecturales multiples qui supportent les compositions évolutives des variabilités de différentes granularités.

Contributions

Dans ce travail, nous présentons un système de modélisation et de gestion de la variabilité architecturale dans les lignes de produits logiciels. Nous l'avons désigné par SGV. Ce modèle s'articule autour de quatre contributions associées toutes à la dimension variabilité.
Ces contributions concernent la représentation de la variabilité dans les modèles UML. Nous étendons UML pour permettre la spécification de la variabilité dans les deux types d'aspects de modèles d'UML: L'aspect statique concerné par les diagrammes de classes, de composant et de déploiement et l'aspect dynamique concerné par les diagrammes de séquence et états-transitions.
Contrairement aux travaux existants, qui ne proposent que des améliorations ciblées sur certains modèles de la variabilité, notre travail touche plusieurs aspects de la variabilité dans les diagrammes statiques et dynamiques d'UML. Les contributions que nous avons apporté à la représentation de la variabilité dans les LdP sont axées sur les thèmes suivants - cf. figure 1 - :

1- *L'optionalité :* plus raffiné dans notre modèle, ce concept prendra en compte l'extensibilité de la feature modélisée. une feature optionnelle dont l'extension est prévue dans la LdP est une feature optionnelle accessoire. Dans le cas contraire (aucune extension n'est prévue pour la feature), la feature est d'une optionalité stricte. Ces nouveaux types de variabilité sont applicables au niveau des modélisations des points variables optionnels dans toutes les vues relatives à la LdP.

2- *L'extension des variabilités représentant les attributs et les méthodes à d'autres types.*

Les types existants pour décrire la variabilité dans les attributs et les méthodes sont les mêmes utilisés pour décrire les classes, ce qui est insuffisant pour décrire pleinement la variabilité de ces deux abstractions .Dans notre modèle nous avons proposé deux autres types de variabilités pour les attributs et méthodes à savoir le type individuel et le type collectif. Ces types se rapportent au comportement des méthodes et attributs vis-à-vis des instances des classes variables et variantes. Le comportement se traduit par la description pour les attributs et par le traitement pour les méthodes. Le type individuel est relatif à un comportement vis-à-vis d'une seule instance à la fois (il ne peut concerner qu'une instance de la classe). Le type collectif par contre se rapporte à un comportement vis-à-vis de toute la population (il concerne toute la collectivité). Cette perception de la variabilité nécessite la connaissance de la portée en instances des superclasses. Cette information n'étant pas fournie par les concepts actuels du diagramme de classes UML, nous avons proposé pour solutionner ce problème d'ajouter dans la structure de la classe un nouvel attribut que nous avons appelé *dimension* qui contient l'information relative au nombre d'instances de la classe. Cet attribut bivalué nous renseignera à l'aide d'un cadrage à deux valeurs sur le nombre minimal et maximal d'instances de la classe.

3- *L'usage des rôles pour modéliser les variants.*

Dans notre modèle, les variants sont perçus dans le diagramme de classes comme des rôles pouvant être joués par les points de variation. Chaque point de variation est représenté par une superclasse ou classe mère avec un stéréotype pour décrire son type (optionnel, alternative ou paramétré). Les variants sont représentés par des sous classes

rôles de la super classe (ou des classes filles rôles de la classe mère) .On les a désigné par variant-rôles).
La dimension : nouveau concept intronisé précédemment au modèle qui permet de connaitre les nombres minimal et maximal d'instances de toute classe dont la classe variable peut être utilisé pour contrôler les cardinalités des alternatives de variants (aspect important pour les LdP).

De plus, le modèle permet de contrôler l'évolution de la variabilité au niveau des variants-rôles à travers l'évolution de ces derniers .Cet aspect dynamique des rôles est pris en compte par une machine à état-transitions qui est définie pour chaque point de variation.
4- Le modèle permet la gestion et la composition des variabilités architecturales en offrant la possibilité de décrire comment les variabilités conceptuelles sont représentées et réalisées à travers les décompositions architecturales des lignes de produits .Cet aspect de la variabilité solutionne le problème que les variabilités architecturales décrites en termes de décomposition et de choix de conception n'ont pas souvent de correspondance biunivoque avec les décompositions des modèles de features.

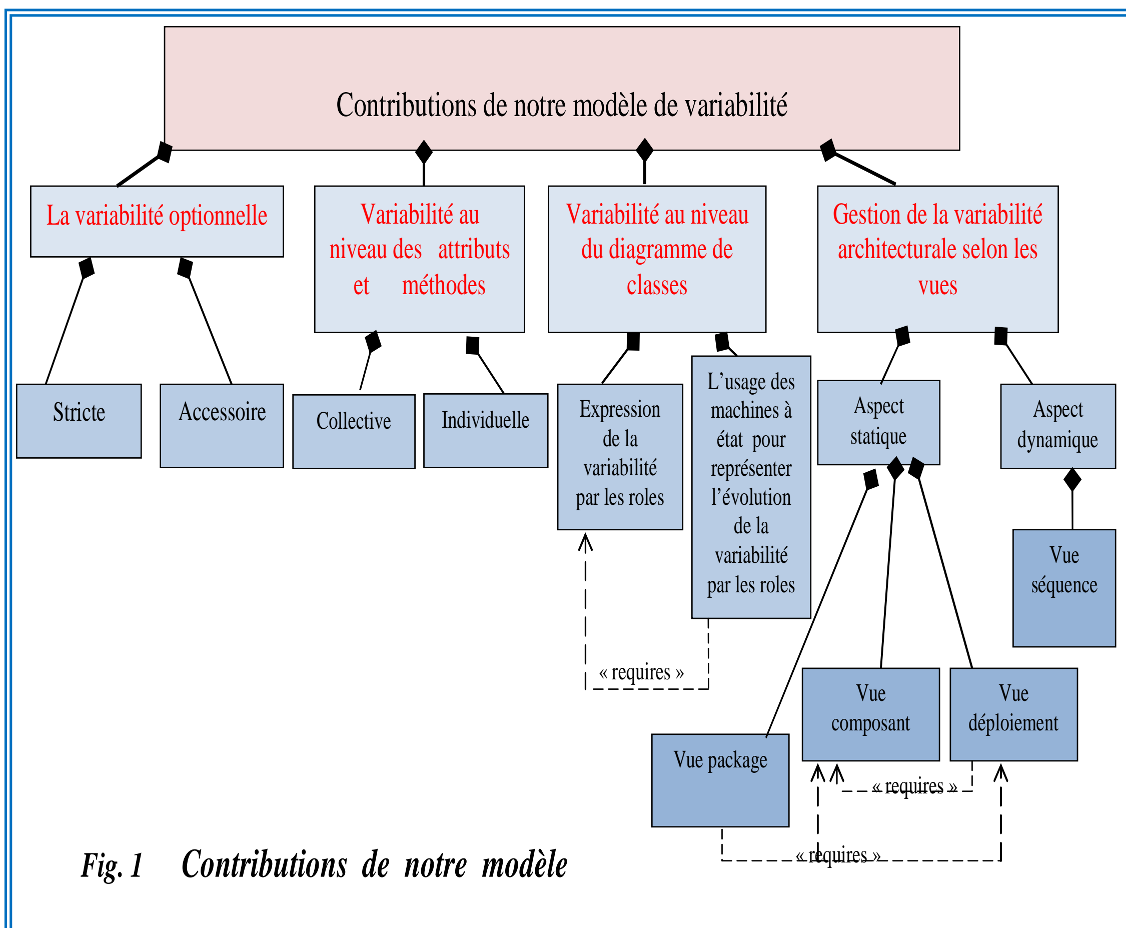


Fig. 1 Contributions de notre modèle

Cette partie de notre modèle permet la gestion de la variabilité architecturale en lui conférant une représentation sous des formes hétérogènes dans différentes vues. Le modèle utilise des mécanismes UML capables de référencer des points de variation à travers des vues architecturales multiples tout en supportant les compositions évolutives des variabilités de différentes granularités. Le modèle complète les approches existantes

de modélisation des variabilités architecturales mais s'en distingue par l'usage exclusif de mécanismes UML (associations et stéréotypes organisés dans un modèle conceptuel).

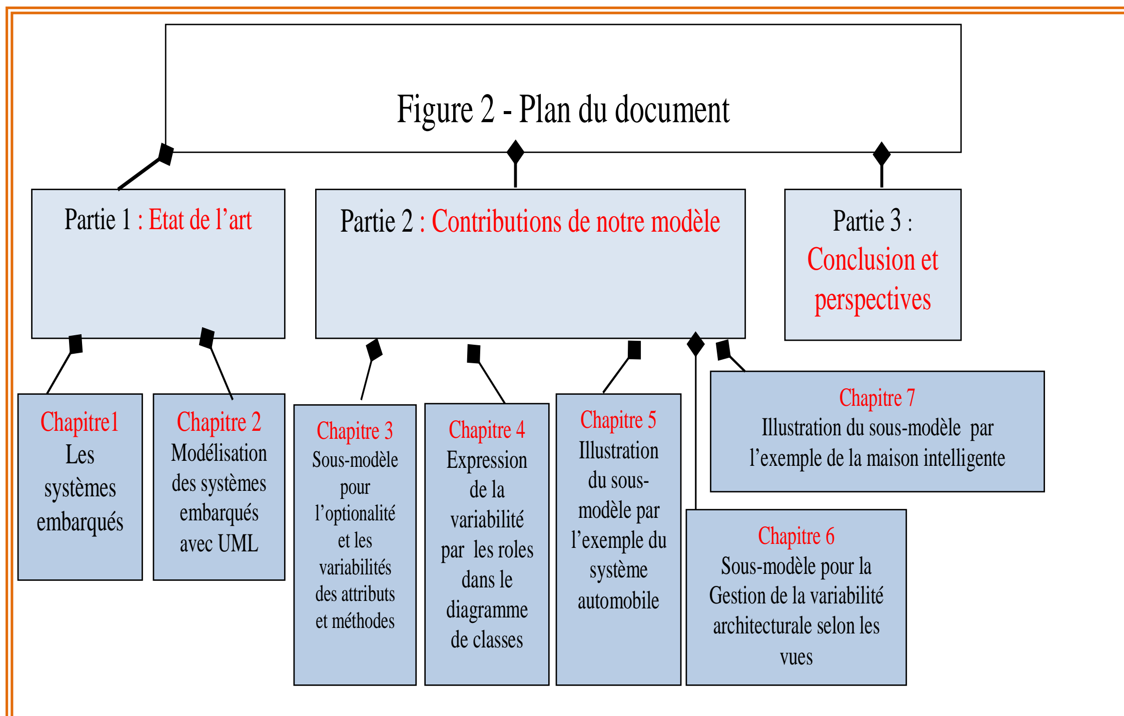
Les travaux présentés dans ce mémoire ont fait l'objet de deux publications suivies de deux expositions lors de la conférence internationale ICAI09 de l'université de Bordj Bou Arreridj en 2009. Ils sont intitulés :

-Article 1 : « *Un modèle de variabilité pour les LdP logiciels* » et

-Article 2 : « *Un modèle conceptuel pour la gestion et la représentation de la variabilité orthogonale selon les vues* ».

Plan du document.

Ce document est organisé en trois parties -cf. Figure 2 - :



- La première partie présente un état de l'art ; elle est divisée en deux chapitres :

Le chapitre 1 présente les systèmes embarqués et leurs concepts fondamentaux compte tenu du fait que ce sont principalement les logiciels enfouis dans les systèmes embarqués qui font l'objet de notre travail dans le cadre des lignes de produits .Le chapitre 2 rappelle les principes et les concepts d'UML qui seront abordés dans le reste de ce document. Il présente aussi un état de l'art des travaux existants relatifs à la modélisation des systèmes embarqués avec les profils UML. Une présentation des LdP comme approche de base dans la conception et le développement des Systèmes embarqués est aussi abordée.

- Les contributions de ce travail et les mécanismes d'intégration de notre modèle à UML sont détaillés dans la deuxième partie de ce document.

Cette seconde partie compte cinq chapitres ; les chapitres 3 et 4, présentent nos trois premières contributions à la modélisation de la variabilité des LdP avec UML : le chapitre 3 illustre les extensions de la notion d'optionalité et le raffinement des types de variabilité des attributs et méthodes, quant au chapitre 4, il illustre l'expression de la variabilité dans les diagrammes de classes à travers les rôles. Ces contributions sont illustrées dans un exemple relatif au système automobile dans le chapitre 5 .Le chapitre 6 aborde notre quatrième contribution à travers laquelle des mécanismes UML sont proposés pour gérer la variabilité orthogonale selon les différentes vues architecturales. Le chapitre 7 illustre les mécanismes de gestion et de modélisation de la variabilité orthogonale à travers l'exemple du système maison intelligente.

- La troisième et dernière partie est consacrée à la conclusion du mémoire et présente des perspectives pour un certain nombre de directions de recherche dans ce domaine.

Première partie
État de l'art

Chapitre 1.

Les systèmes embarqués

1.1 Introduction

Un système embarqué est une combinaison de matériels et logiciels informatiques, et d'autres composants additionnels mécaniques ou autres, conçus pour accomplir une fonction bien spécifique [19,20].

Les systèmes embarqués, s'ils ne constituent pas un problème nouveau, prennent néanmoins une importance économique sans cesse accrue : dans les systèmes traditionnels de contrôle-commande avioniques, spatiaux, industriels, ferroviaires, etc. et dans les systèmes classiques de télécommunications ; s'ajoute maintenant l'explosion de la téléphonie mobile, des systèmes de paiement, et de l'électronique de consommation courante.

L'avionique, l'automobile, les systèmes automatisés de fabrication, les appareils médicaux, les appareils photo/vidéo/hifi, les produits électroménagers, les jouets sont des applications types des systèmes embarqués.

Les systèmes embarqués sont rarement un but en eux-mêmes mais, au contraire, peuvent être vus comme des sous-systèmes de systèmes plus importants qu'ils sont chargés de commander et/ou de surveiller. Ils sont d'origines très diverses, mécaniques, automatiques, aéronautiques, ferroviaires, télécommunications, etc. Les propriétés fonctionnelles (sûreté de fonctionnement, performances, ressource mémoire, ou consommation d'énergie) ont une grande importance pour ce type de systèmes [19 , 21].

Ces systèmes se distinguent de l'informatique d'entreprise (PC, Serveur) sur plusieurs points [22] :

- Contraintes matérielles et physiques des terminaux: encombrement, périphérique d'entrée, autonomie énergétique, puissance de calcul, …
- Contraintes réseaux: fiabilité, débit et coût, fonctionnement alterné online/offline, configuration éphémère des éléments réseaux, …

Ces contraintes ont influencé la conception et l'architecture de leurs systèmes d'exploitation et le développement des applications.

Etant donné que les systèmes embarqués sont dans leur grande majorité des systèmes temps réel, ils doivent respecter des contraintes temporelles fortes et l'on y trouve enfoui un système d'exploitation ou noyau Temps Réel (RTOS, Real Time Operating System) [23].

Un système temps réel est un système réactif devant fournir des sorties logiquement correctes tout en respectant strictement des contraintes temporelles explicites [23]. Il est considéré comme défaillant s'il ne respecte pas au moins une de ses spécifications logiques ou temporelles.

Un système embarqué temps réel fait intervenir :

1) Un ordinateur (calculateur), dont les activités sont gouvernées par un système d'exploitation spécialement conçu pour les activités temps réel.

2) Un processus physique à commander, depuis lequel l'ordinateur reçoit de l'information et sur qui il exécute ses commandes.

3) Des appareils d'interconnexion entre les deux composants ci-dessus tels que: Transducteurs, Actionneurs.

L'ordinateur lit les données depuis les transducteurs et répond avec des commandes exécutées par les actionneurs toujours dans des limites temporelles bien précises.

Un système embarqué temps réel interagit avec un environnement externe qui lui-même évolue avec le temps. Il réalise certaines fonctionnalités en relation avec cet environnement, et exploite des ressources limitées [24]. Deux principales contraintes sont à vérifier pour qu'un système temps réel embarqué soit fonctionnel [25] :

- *Exactitude logique (logical correctness):* sorties adéquates en fonction des entrées, assurant le comportement désiré pour le système suite à des événements et aux données communiquées;

- *Exactitude temporelle (timeliness):* respect des contraintes temporelles;

La défaillance des systèmes embarqués temps réel peut avoir des conséquences importantes et leur niveau de criticité est souvent élevé;

Dans ce chapitre, nous allons présenter un aperçu sur les systèmes embarqués, leur importance dans notre vie quotidienne à tous les niveaux, leurs différentes classes ainsi que les contraintes et les exigences qui leurs sont imposées et pour terminer les aspects spécifiques nécessaires à leurs différentes approches de développement.

1 2. Importance des systèmes embarqués

Les systèmes embarqués caractérisés par un aspect omniscient, mobile et embarqué (téléphonie mobile, assistant personnel, …) sont bien présents depuis plusieurs années pour l'usager, à son domicile, à son travail ... Ces systèmes informatiques se dotent maintenant de moyens de communication reliant entre eux l'ensemble des machines de l'usager au sein d'un réseau personnel (PAN: Personal Area Network) au moyen de réseaux de proximité. De plus, ce réseau personnel est lui même relié à un réseau global de services via des réseaux mobiles (GSM par exemple).

Les systèmes embarqués sont d'une importance stratégique pour l'économie. Ils constituent un facteur d'innovation et de différentiation: nouvelles fonctionnalités et nouveaux services dans les produits existants, nouveaux produits et services. Ils constituent aussi une source principale de valeur ajoutée: surtout les logiciels embarqués. Ils jouent un rôle important dans l'amélioration de la compétitivité [26].

L'importance des systèmes embarqués se manifeste par la multiplication des produits utilisant ces systèmes mais aussi par les efforts des organisations gouvernementales ou industrielles pour développer ce secteur. Le développement de l'électronique embarquée devrait augmenter en particulier dans le domaine des transports (automobile, avionique, transport ferroviaire) et des télécommunications. Récemment, d'autres domaines tel l'électronique à faible, coût semblent émerger : la domotique, les applications multimédia (photo, vidéo, télévision interactive….). Même s'il est difficile de prévoir précisément quelles applications se développent plus rapidement que d'autres, il semble raisonnable de conclure que l'usage des systèmes embarqués est en constante augmentation.

1.3 Les classes de systèmes embarqués

1.3.1 <u>Les systèmes embarqués dans l'automobile.</u>

Le domaine de l'automobile est très représentatif de la situation dans laquelle les systèmes embarqués se développent [27]. Il comprend de très riches fonctionnalités tant complexes que diversifiées. C'est là où on a les systèmes embarqués les plus nombreux. C'est un domaine où la compétition est sans ralenti et où les ingénieurs doivent être constamment créatifs. De nouvelles fonctions du véhicule sont de plus en plus basées sur le logiciel [28]. Dans la plupart des cas, les besoins des systèmes embarqués dans l'automobile sont des besoins communs aux autres domaines. La figure 1.1 illustre les différentes fonctionnalités dans l'automobile, contrôlées par le logiciel.

La conception de nouveaux modèles automobiles est rendue complexe par la demande croissante de nouveaux services faisant appel à de nouvelles technologies. Les nouveaux services attendus concernent aussi bien la sécurité du véhicule, que l'assistance à la conduite ou la communication permanente du véhicule avec son environnement (cf.Tableau 1.1).

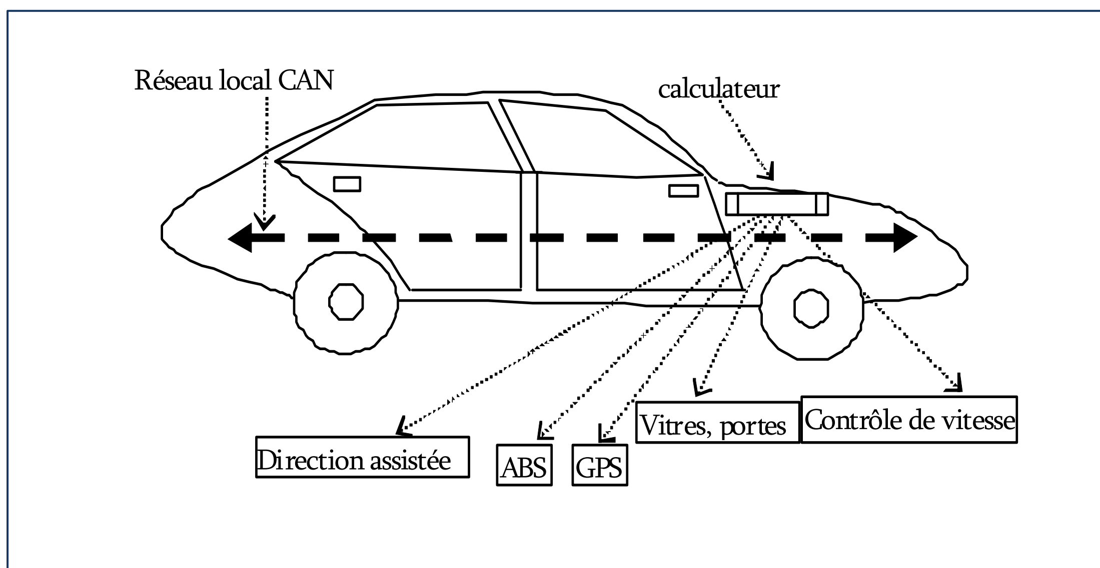


Figure 1.1: Exemple de systèmes embarqués dans l'automobile

La complexité croît également par la possibilité donnée au conducteur de personnaliser son véhicule, ainsi que par les moyens à mettre en œuvre pour assurer sa maintenance, et anticiper les pannes.

Catégorie	Nouveau service
Sécurité	Assistance au freinage (ABS) ; Airbags intelligents ; Assistance anticollision ; Freinage tout électronique (brake-by-wire)
Moteur	Injection électronique
Navigation	Suivi de parcours par GPS
	Repérage et guidage sur voie
	Contrôle de la vitesse
Productivité	Gestion de flotte: repérage, aide à la navigation, informations professionnelles
Information	Connexion Internet (UMTS)
Confort	Gestion intégrée de l'habitacle (climatisation, portes, éclairage, visibilité) ;CD Radio , Direction tout électronique (steer-by-wire) , Assistance au stationnement
Confort-Sécurité	Commandes vocales ; Ecran de projection

Tab 1.1. Les nouveaux services dans l'automobile contrôlés par le logiciel.

D'un point de vue technologique, les évolutions dans l'automobile sont proches dans leurs principes, à ce qui se passe dans l'aéronautique: la technologie «by-wire» s'impose, les systèmes mécaniques de transmission sont remplacés par des bus électroniques connectant des unités de calcul (en anglais: Electronic Control Unit: ECU), capteurs et actionneurs. D'autres évolutions technologiques accompagnent l'introduction de l'électronique et du logiciel, comme la banalisation des calculateurs et le téléchargement de logiciel pour réparation ou évolution (Mise à jour).

De nouveaux standards sont en cours d'adoption ou de généralisation dans le monde des constructeurs et des équipementiers pour l'exécution du logiciel et la communication à bord. Le standard le plus significatif à ce titre est OSEK/VDX (Open Systems and the Corresponding Interfaces for Automotive Electronics) [29] dont L'utilisation augmente la portabilité et l'intégrabilité des logiciels développés, ce qui constitue un avantage aussi bien pour les constructeurs que pour les équipementiers.

Dans ces nouvelles architectures distribuées, les standards de communication les plus notables, en complément à OSEK/VDX Com, sont le bus CAN (*Controller Area Network*, pour du faible débit) et le bus MOST (pour les réseaux multimédia haut débit).

L'emploi d'un bus IEEE 1394 pour du haut débit ou de connexions radio «Bluetooth» pour de la très courte distance sans fil suscitent aussi beaucoup d'intérêt.

Différents réseaux sont déployés sur le véhicule, basés sur ces technologies, en fonction du débit, par exemple un réseau CAN faible débit pour l'habitacle connectant les calculateurs de gestion des portes, l'éclairage et la visibilité, un autre réseau MOST haut débit connectant les applications de communication, multimédia et internet, ou en fonction de la sensibilité des informations, comme un réseau haut débit indépendant pour le contrôle moteur.

1.3.2 Les systèmes embarqués dans l'avionique.

L'avionique représente un domaine industriel important. Le Système avionique est un ensemble de moyens informatiques embarqués à bord d'un avion, d'un lanceur, d'un satellite… .Les logiciels d'application et les logiciels exécutifs sont nombreux -cf.Figure 1.2-:

Le pilote automatique, les calculateurs, les bus, les passerelles…Cependant le cœur

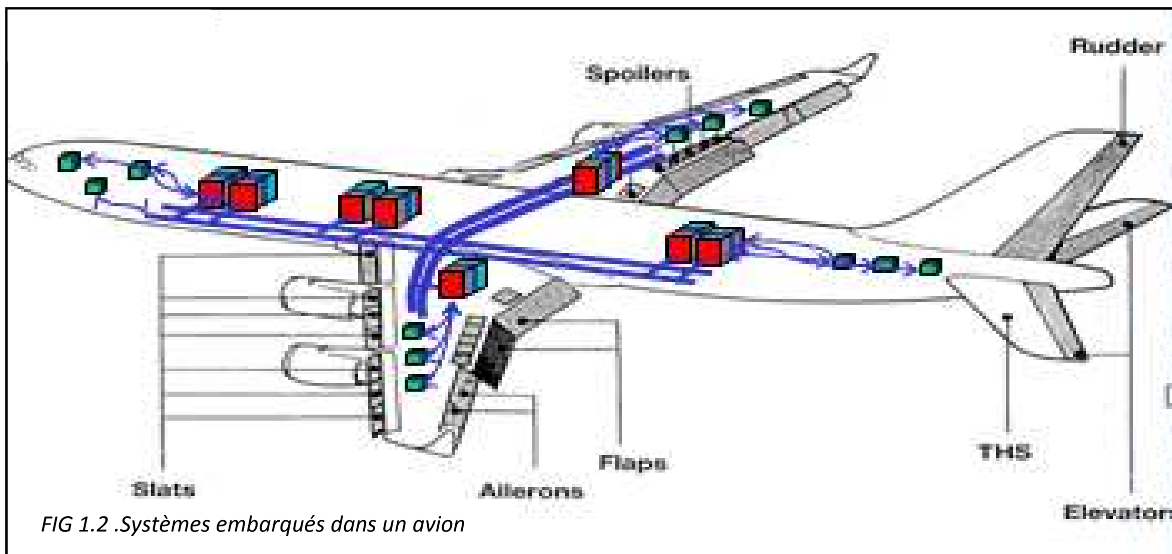


FIG 1.2 .Systèmes embarqués dans un avion

avionique doit assurer essentiellement la gestion du vol (navigation, pilotage) ou la gestion des missions (surtout pour le domaine militaire).

1.3.2.1 L'avionique civile :

a- Fonctionnalités : dans le cœur avionique, on trouve des systèmes de :

Guidage : systèmes FMS (Flight Mangement System) et FGS (Flight Guidance System) dont le role principal est la planification et le contrôle de la trajectoire de l'avion (long terme) (suivi de plan de vol, tenue de cap, d'altitude, suivi axe radioguidage…)

Pilotage : pour le contrôle des mouvements avion autour de son centre de gravité (contrôle assiette, facteur de charge, roulis…)

Asservissement : pour l'asservissement des organes de pilotage (gouvernes et moteurs).

Protection : notamment pour le domaine de vol.

Interface **:** avion / équipage pour la gestion des écrans cockpit (EIS), la gestion des alarmes (FWS)…

Divers : pour la gestion du carburant, la gestion de l'alimentation électrique, anticollision…

b- Les techniques de développement des systèmes avioniques :

Les principales techniques nécessaires au développement de systèmes avioniques sont :

- l'ingénierie système et ingénierie logiciel
- l'automatique (discrète et continue)
- la spécification et la vérification formelle
- la sûreté de fonctionnement
- la garantie de la performance temps réel
- les réseaux industriels embarqués

1.3.2.2 L'avionique militaire : l'avionique militaire a trait principalement aux avions de chasse .Elle concerne les systèmes et les équipements informatiques ainsi que les fonctions, les calculateurs, les bus de communication, les coupleurs bus, les unités graphiques (générateurs de symboles et écrans), les coupleurs radars... embarqués à bord de l'aéronef (voir figure 1.3) .

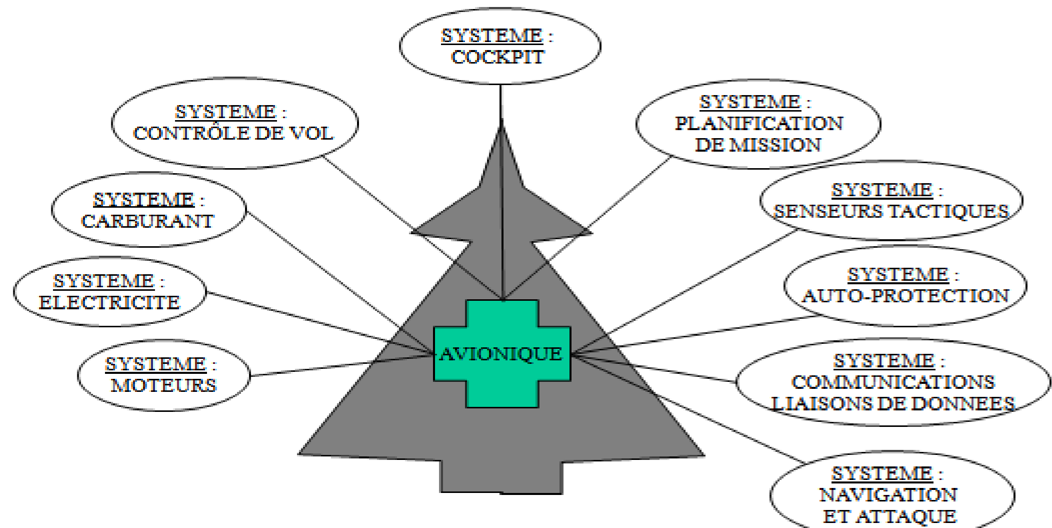


Fig. 1.3 .Systèmes enfouis dans un aéronef militaire

Les systèmes que l'on trouve dans un avion militaire sont :

- Pour les Emports : on y distingue :
  - Les armes
  - Les réservoirs
  - Les capteurs amovibles (POD désignation laser, …)
- Pour le Système principal, on y trouve :
  - Le Système cellule équipée
  - Le Système propulsif
  - Le Système de commande de vol
  - La Plate-forme cellule
  - Le Moyen d'emport
  - Le Système de Navigation et d'attaque qui est le cerveau de l'avion

L'Avionique est un domaine industriel en plein essor dont l'extension se fait ressentir rapidement au secteur de l'automobile. Il est caractérisé par des systèmes très critiques et de temps réel avec une forte exigence de garantie de qualité de service et une exigence de certification.

1.3.3 Les systèmes embarqués dans la robotique :

Les robots sont composés d'un ou de plusieurs systèmes embarqués.
Exemple : le robot Peintre dans une chaîne de montage de voitures (Renault), le Robot chien (Sony) ; Aspirateur (Electrolux) ; les Robots de combats (robots destructeurs) et les Robots chirurgiens…..
Les différents types de taches assignées à des robots sont :

-Tâches périlleuses : déminage, alerte à la bombe, gaz toxiques
-Tâches de précision : construction d'autres robots plus petits ou déplacement précis d'un bras.
-Tâches répétitives : mise en palette de petits objets
-Tâches délicates : vissage dans les endroits difficiles
-Tâches pénibles : déplacement de charges très lourdes

C'est selon la nature des taches accomplies que les robots doivent disposer de tels ou tels systèmes embarqués.
Pour les Derniers développements de la robotique, on parle de:

-Robot poisson avec de vrais muscles (MIT)
-Cerveaux de souris avec neurones humains (Stem Cells)
-Une machine apprend le langage humain (AIE)
-Pour un projet européen (Ambiance), l'ENST a développé un robot routeur qui est capable de communiquer en Ethernet radio (Wifi), en Bluetooth et par infrarouge .il se déplace (navigue) seul là où il faut sur le réseau fait des ponts entre les protocoles va se recharger en évitant les obstacles ,collabore avec les autres robots pour assurer une meilleure communication.

1.3.4 Les systèmes embarqués dans la domotique

La domotique est l'ensemble des technologies de l'électronique, de l'informatique et des télécommunications utilisées dans les habitations. La domotique vise à assurer des fonctions de sécurité (comme les alarmes), de confort (comme les volets roulants), de gestion d'énergie (comme la programmation du chauffage) et de communication (comme les commandes à distance) que l'on peut retrouver dans la maison (smart home ou maison intelligente). Il s'agit donc d'automatiser des tâches en les programmant ou les coordonnant entre elles.

1.3.4.1 Les domaines de la domotique

Les principaux domaines dans lesquels s'appliquent les techniques de la domotique sont :

- La programmation des appareils électrodomestiques, électroménagers.
- La gestion de l'énergie, du chauffage (par exemple, de la climatisation , de la ventilation, de l'éclairage, de l'ouverture et de la fermeture des volets (par exemple en fonction de l'ensoleillement), de l'eau (les robinets de lavabos peuvent ouvrir l'eau à l'approche de mains, etc.).
- La sécurité des biens et des personnes (alarmes, détecteurs de mouvement, interphone et digicode) .

- Le « confort acoustique ». Il peut provenir de l'installation d'un ensemble de haut-parleurs permettant de répartir le son et de réguler l'intensité sonore.
- La gestion des ambiances lumineuses.
- La compensation des situations de handicap.

1.3.4.2 Techniques de la domotique

La domotique est basée sur la mise en réseau des différents appareils électriques de la maison, contrôlés par une « intelligence » centralisée. L'intelligence (systèmes embarqués) qui gère ces commandes est une centrale programmable, des modules (intelligence décentralisée), ou bien une interface micro-informatique (un serveur). Les outils de pilotage peuvent être un ordinateur de poche, un téléphone portable ou smart phone , une télécommande (universelle ou non), un écran + une souris ou un écran tactile, pour n'en citer que les principaux.

Ainsi, il est possible de programmer soi-même sa maison grâce à une interface, (généralement informatique), reliant les appareils connectés .De plus il est possible, par une programmation plus complexe, d'adapter le système à son propre rythme de vie.

Les interfaces de contrôle permettant de contrôler le serveur utilisent :

- Des surfaces de contrôles possédant de nombreux boutons, télécommandes, souris, clavier avec ou sans fils
- Des interfaces tactiles : écrans tactiles (PDA ou UMPC) associés à un logiciel ou une interface web
- Des microphones (téléphones GSM ou VIP) associés à des logiciels de reconnaissance vocale.

Les modes de transmission tels que l'infrarouge, le courant porteur, les ondes radio sont utilisés, mais il est évident que les liaisons filaires (Ethernet, BUS…) jouent en faveur de la stabilité du système.

- Par ondes Radio Hertzienne
- Par infrarouge
- Par courant porteur secteur
- Par signatures numériques ou magnétiques
- Par réseau câblé Ethernet ou BUS

1.3.4.3 La domotique pour l'Assistance à la vie quotidienne

Des robots d'assistance à la vie quotidienne sont à l'étude dans de nombreux laboratoires de recherche. Ces robots domestiques prennent de plus en plus la forme humaine et sont parfaitement capables de se déplacer en évitant tous les obstacles. Ces machines sont actuellement limitées par des exigences de sécurité. Aujourd'hui, les robots possèdent des dispositifs de sécurité leur ôtant toute force mécanique dès qu'ils entrent en contact avec une personne par exemple.

Des groupes d'études travaillent sur des systèmes de capteurs de vision, programmés pour reconnaître les visages, ainsi que sur des programmes permettant au robot d'acquérir les mêmes possibilités d'apprentissage que les humains.

Quelques robots d'assistance à domicile sont déjà commercialisés, à l'image de Wakamaru, lancé en 2005 et révisé plusieurs fois notamment en 2008 par Mitsubishi Heavy industries. Wakamaru peut remplir plusieurs offices, comme rappeler un rendez-vous important, se connecter à Internet par le Wifi pour aller y chercher de l'information et la retransmettre grâce à sa voix ou surveiller la maison.

À long terme et en fonction de l'avancée des études sur la sécurité, les robots d'assistance devraient représenter un marché de 18 milliards d'euros d'ici à la fin de l'année 2009. Les robots pourraient être chargés des tâches suivantes :

- ménage,
- aide aux personnes handicapées,
- Internet/Mail/Messagerie instantanée,
- surveillance,
- guide de visiteurs. …

1.3.4.4 La compensation des situations de handicap ou de dépendance

La réalisation d'un environnement contrôlable à distance, demande l'installation d'un dispositif complexe, composé de trois parties :

1. Une interface entre l'utilisateur et le système domotique : appelée plus communément contrôle d'environnement, elle permet à la personne de contrôler son environnement, en sélectionnant et activant les éléments à contrôler. Cette interface peut être fixe, mobile ou mise sur un fauteuil roulant (Pour accéder à une liste de contrôles d'environnement). Suivant les capacités résiduelles de la personne, on peut proposer d'installer différentes commandes spécifiques (ou contacteurs). Les contacteurs peuvent être mécaniques, pneumatiques, musculaires).
2. Le système domotique **:** il permet de centraliser les informations émises par l'interface, afin de les organiser et de les adresser à des effecteurs.
3. Les effecteurs : regroupe tous les appareils et moteurs à contrôler à distance, tels que, par exemple : les fonctionnalités d'un lit (têtière, plicature de jambes, hauteur), le téléphone, la télévision, les portes, les lumières, les volets, le chauffage, les prises…

1.3.5 Les systèmes embarqués dans les autres secteurs de la technologie

L'immotique, c'est la domotique à l'échelle d'un grand bâtiment, immeuble ou grand site industriel ou tertiaire, etc. Ce qui implique des solutions techniques de domotique visant à gérer des quantités de modules de systèmes embarqués plus importantes que pour un simple domicile de particulier. Le terme *immotique* a été déposé à l'INPI par la société ISIOM, qui est à la fois un cabinet de conseil en immobilier d'entreprise et un éditeur/intégrateur de solution progiciel de gestion globale du patrimoine immobilier.

Deux grandes familles de solutions immotiques :

- Gestion Technique de Bâtiment (GTB)
- Gestion Technique Centralisée (GTC)

1.3.6 Autres domaines d'applications .

Les autres domaines dans lesquels on trouve des systèmes embarqués sont de plus en plus nombreux :

- Astronautique : fusée, satellite artificiel, sonde spatiale, etc..
- Militaire : missile
- Telecom : téléphonie, set-top box , pare-feu , routeur ; téléphone portable serveur de temps etc. …
- Impression : imprimante multifonctions, photocopieur, etc.
- Informatique : disque dur, lecteur de CD, flash disque .. etc.
- MultiMedia : console de jeu vidéo, assistant personnel …
- Guichet automatique bancaire (BAG)
- Equipement médical (scanner ….)
- Automate programmable industriel, contrôle de commandes …
- Jeux : consoles de jeux …
- Métrologie.

1.4 Exigences et contraintes

Les systèmes embarqués fonctionnent souvent dans des environnements physiques caractérisés le plus souvent par des ressources limitées, que ce soit au niveau de la mémoire, de l'énergie (utilisation de batteries et/ou, de panneaux solaires voir de piles à combustible) ou des capacités de traitement. De plus, compte tenues des fortes interactions des logiciels embarqués avec leur environnement, ces derniers sont confrontés à des contraintes de temps réel ou de sûreté de fonctionnement plus ou moins fortes selon le type d'utilisation du système embarqué.

Plus précisément, les systèmes embarqués doivent [20, 30] :

- Satisfaire des contraintes de temps réel provenant des applications, qui peuvent être aussi bien des contraintes de temps réel souple (par exemple dans le cadre d'applications multimédia) que des contraintes temps réel strict (par exemple dans le cadre de protocoles de communication).
- S'adapter à la nature périssable de certaines ressources (énergie), au fonctionnement aux limites du matériel (surchauffe), à des capacités de stockage limitées, et doivent intégrer l'instabilité des systèmes de communication utilisés (déconnexion, bande passante).

- Offrir une puissance de traitement suffisante pour satisfaire les contraintes de temps d'exécution des applications. Les processeurs utilisés dans les systèmes embarqués sont 2 à 3 décades moins puissants qu'un processeur d'un ordinateur PC.
- Revenir au moindre cout possible surtout s'il est produit en grande série.
- De sécurité et de sûreté de fonctionnement. Car s'il arrive que certains de ces systèmes embarqués subissent une défaillance, ils peuvent mettre des vies humaines en danger ou mettent en périls des investissements importants. Ils sont alors dits « critiques » et ne doivent jamais faillir. Par « jamais faillir », il faut comprendre toujours donner des résultats justes, pertinents et ce dans les délais attendus par les utilisateurs (machines et/ou humains) .

1.4.1 Contraintes de temps

Beaucoup de systèmes embarqués interagissent directement avec leur environnement via des capteurs/actionneurs ou un réseau de communication sans fil. Ces interactions contraignent les temps de réponse du système embarqué de manière plus ou moins forte selon le domaine d'applications visé. On parle alors de système temps réel [31] ou plus précisément de système embarqué temps réel [32], dans le sens où le temps de livraison des résultats d'un calcul fait partie intégrante de la spécification de ce dernier, au même titre que le résultat lui-même.

Temps réel strict. Dans les systèmes temps réel strict, ou dur, le non respect des contraintes de temps du système, le plus souvent exprimées sous la forme d'échéances de terminaison, constitue une défaillance de l'application. Dans le cadre d'applications critiques, telles que par exemple certaines applications dans l'automobile ou l'avionique, une telle défaillance peut avoir des conséquences catastrophiques, telles que la mise en danger de vies humaines ou des pertes financières importantes [33].

Etant donné les conséquences importantes du non respect d'une échéance dans les systèmes temps réel strict, il est nécessaire (ou tout du moins fortement recommandé) pour de tels systèmes de pouvoir vérifier avant leur exécution que toutes les échéances seront toujours respectées. Pour cela, des méthodes d'analyse d'ordonnançabilité [34] doivent être utilisées afin de procéder à cette vérification.

Système temps réel souple. Dans les systèmes temps réel souple, bien que l'instant d'obtention du résultat soit important, la violation des contraintes de temps du système est tolérée si elle reste rare. Cette tolérance est due au fait que les applications concernées ne relèvent pas du domaine des applications critiques. Les exemples typiques d'applications ayant des contraintes temps réel souples sont les applications multimédias à flux continus.

Le système vise au respect des contraintes de temps dans la délivrance des flux de données afin de garantir la qualité des images et du son; toutefois, les contraintes de temps peuvent être adaptées puisque une dégradation de la qualité des données ne sera que faiblement perçue par l'utilisateur.

Performance temporelle. A cause de la demande toujours croissante de puissance de traitement des applications, la performance globale du système embarqué est également un critère de conception important. Aussi, dans la conception d'un système embarqué la performance temporelle des mécanismes est une préoccupation constante. Au niveau logiciel ce travail porte à la fois sur les algorithmes afin de diminuer leur complexité, mais aussi sur l'optimisation des séquences fréquentes de code d'un mécanisme particulier

1.4.2 Consommation énergétique

Une grande majorité des systèmes embarqués (téléphones mobiles, ordinateurs de poche,…) sont confrontés au problème de l'autonomie. Aussi, afin d'étendre l'autonomie de fonctionnement de tels systèmes, deux approches sont actuellement possibles: augmenter la capacité de stockage des batteries ou réaliser un système embarqué à faible consommation énergétique. Dans le cadre de cette dernière approche, plusieurs méthodes sont alors envisagées qui touchent à la fois le domaine de l'électronique et du logiciel:

i- La conception de composants électroniques consommant le minimum d'énergie,

ii- L'optimisation du logiciel afin de diminuer le coût énergétique de son exécution,

iii- La conception de stratégies logicielles exploitant les fonctionnalités du matériel.

L'optimisation du logiciel consiste à privilégier les instructions moins gourmandes en énergie afin de diminuer la consommation énergétique de l'exécution du programme. Pour l'optimisation d'applications de type protocole réseau sans fil, elle porte sur la réduction du volume des communications afin de diminuer la consommation d'énergie occasionnées par l'utilisation du réseau. De plus, dans un contexte d'ordinateur de poche sans fil, l'exécution distante de traitement permettra de diminuer la consommation locale de l'exécution du traitement pour peu que le mécanisme de réalisation de l'exécution distante consomme moins que le traitement proprement dit.

La conception d'une stratégie logicielle exploitant des fonctionnalités du matériel afin de diminuer la consommation énergétique de l'ordinateur touche principalement à l'ordonnancement.

Tout d'abord, certains processeurs offrent différents modes d'exécution. Outre le mode d'exécution nominal, un mode veille permet à la fois d'endormir à faible coût le processeur et également de le réveiller de manière rapide.

1 4.3 Mémoire

Dans un grand nombre de systèmes embarqués, la mémoire est une source limitée (dans un téléphone portable, elle est de quelques Kilo-octets dans une carte à puce à quelques Mégaoctets), et par conséquent une bonne utilisation de la ressource mémoire est cruciale pour ces systèmes. Une difficulté supplémentaire dans les systèmes embarqués est que la gestion de la mémoire soit compatible avec les contraintes temps réel des applications, qu'elles soient souples ou strictes.

1.4.4 Tolérance aux fautes .

Certains systèmes embarqués doivent pouvoir remplir leurs fonctions malgré la présence de fautes, qu'elles soient d'origine physique ou humaine. Pour le premier type de faute par exemple, il est nécessaire de détecter les erreurs, par l'utilisation de méthodes telles que les codes détecteurs d'erreurs, les contrôles de vraisemblance ou encore le diagnostic en ligne, puis d'effectuer un recouvrement d'erreur permettant au système de continuer à remplir ses fonctions malgré l'erreur, que ce soit par reprise de son exécution à partir d'un état sauvegardé au préalable (point de reprise) ou par compensation exploitant la redondance présente dans le système (par exemple la duplication active de tâches).

Les difficultés issues du contexte embarqué sont relatives aux contraintes de temps de logiciels embarqués, ainsi qu'aux ressources limitées de l'architecture. Ceci contraint les types de méthodes de tolérance aux fautes utilisables dans un contexte embarqué temps réel. En particulier, dans les systèmes temps réel strict, il est nécessaire d'intégrer les mécanismes de tolérance aux fautes dans l'analyse d'ordonnançabilité du système.

1. 5 Architecture.

Les systèmes embarqués utilisent généralement des microprocesseurs à basse consommation d'énergie ou des microcontrôleurs dont la partie logicielle est en partie ou entièrement programmée dans le matériel, généralement en mémoire dans une mémoire morte ROM , EPROM , EEPROM , FLASH ....(on parle alors pour cette partie logicielle de Firmware).Les langages sont fréquemment utilisés pour manipuler les architectures (voir les ADL dans ch.2).

1.6 Développement des systèmes embarqués

Les systèmes embarqués présentent des aspects spécifiques nécessitant des approches adéquates de développement [35]. Les aspects les plus importants sont la disponibilité limitée des ressources (mémoire, processeur, moyen de communication) et les contraintes de temps qui sont souvent imposées. De telles contraintes ne peuvent être satisfaites que si le logiciel se comporte d'une façon suffisamment prédictible [33]. Les contraintes temporelles strictes et les processus de perturbations fortement corrélés qui caractérisent souvent les systèmes embarqués, demandent l'utilisation de nouvelles techniques de développement, mais aussi d'évaluation et d'optimisation.

De ce fait, le développement de tels systèmes, dans lesquels figurent des éléments informatiques et électroniques toujours plus importants, demande des méthodes puissantes et fiables pour s'adapter à des contraintes en temps et coût de développement toujours plus critiques [35].

Ce développement prend actuellement une ampleur considérable. Il est aussi largement influencé par l'énorme palette de systèmes mobiles, tels que les téléphones mobiles ou les PDA. De plus, la mise en réseau et la complexité augmentent alors que le temps de développement à disposition se réduit.

Toutefois et comme autrefois une grande partie de logiciel embarqué est écrite en assembleur ou en C/C++. Des critères, tels que la modularité, la compatibilité, la facilité de maintenance, la fiabilité et la documentation, ont pris une importance capitale dans les systèmes embarqués souvent développés pour durer.

Pour les avantages qu'il apporte (réutilisation, encapsulation, modularité), le modèle orienté objet est de plus en plus utilisé dans les systèmes embarqués. Basées sur UML comme langage standard de facto, de modélisation de systèmes informatiques, des méthodes ont été proposées et sont même déjà utilisées [23], Octopus [36], Room [36], Ropes [30]).

Elles permettent d'établir la spécification d'ensemble, la conception générale, et la conception détaillée en sous-modèles décrivant chaque aspect du système (fonctionnel: structure, comportement, traitements; matériel: architecture, performance, consommation, temps de réponse, taille mémoire…), indépendamment de sa programmation finale en C, C++ ou Java. Autre ligne directrice de ces méthodes, l'automatisation: aux termes de la spécification détaillée, par exemple, les sous-modèles doivent pouvoir être assemblés et affinés pour l'étape suivante par des outils logiciels automatisés [27]. Même logique pour la génération et l'exécution des tests de validation, qui doivent se définir et s'exécuter en automatique.

Les techniques modernes de développement de logiciel, le standard UML [UML 2.x], les lignes de produits –LdP- ainsi que l'approche MDA [MDA2004] sont fortement demandées par les concepteurs de systèmes embarqués car il devient primordial d'assurer la conception de tels systèmes vue leur complexité (system on chip ou SoC).

Les nouveaux problèmes posés par la mobilité et les environnements temps réel nécessitent des compétences spécifiques présentes dans les différentes structures de recherche.

Première partie
État de l'art

Chapitre 2

UML pour la Modélisation des systèmes embarqués.

2.1 Introduction.

UML (Unified Modeling Language) est le standard de l'OMG (Object Groupe Management) pour la modélisation orientée objet des systèmes logiciels. Il propose un ensemble de notations, sous forme de diagrammes, pour la documentation et la spécification des systèmes.
Les diagrammes d'UML modélisent le logiciel selon différents points de vue : vue fonctionnelle, vue statique, vue dynamique et vues d'implantation et de déploiement.

Dans ce chapitre, nous allons évoquer les mécanismes d'extension d'UML et nous détaillerons en particulier les profils orientés systèmes embarqués tels que les profils destinés aux systèmes complexes englobant plusieurs disciplines de l'industrie, les profils de gestion du temps, les profils dédiés au temps réel et temps réel critique et enfin ceux destinés essentiellement à améliorer la conception de SOCs.

2.2 Les mécanismes d'extension d'UML .

A priori la notation UML est destinée à la modélisation de tout type de systèmes. Cependant certains systèmes dans des domaines spécifiques sont caractérisés par des propriétés qui nécessitent l'ajout d'informations supplémentaires. Prenons par exemple le domaine des systèmes embarqués temps réel pour lesquels il est nécessaire de spécifier des propriétés de qualité de service. La notation standard d'UML ne permet pas d'ajouter une telle information d'où le besoin d'une extension de cette notation.
En plus des domaines spécifiques, il y a aussi le besoin d'adapter UML à une plate-forme technique particulière. Par exemple, pour modéliser un système qui sera implanté sur une plate-forme de type EJB (Entreprise Java Beans), on aura besoin de dire qu'une classe particulière joue le rôle d'un composant EJB (Bean).

Pour répondre à ces besoins particuliers, l'OMG a introduit depuis la version UML1.3 des mécanismes dits d'extension permettant de spécialiser et d'adapter UML à des domaines, des plate-formes ou des méthodes spécifiques. Les mécanismes d'extension introduits depuis la version UML1.3 sont les *Tagged Values, les Stéréotypes, les Contraintes* et *les profils.*

2.3 Les profils UML pour les systèmes embarqués :

2.3.1 Définitions.

Un profil, par définition, étend un méta-modèle de référence ou un autre profil [17]. Le méta-modèle de référence peut être le méta-modèle d'UML ou un autre méta-modèle basé sur la librairie d'infrastructure d'UML. L'appellation profil UML est utilisée pour désigner un profil dont le méta-modèle de référence est le méta-modèle UML. Les profils UML peuvent être considérés comme des dialectes du langage UML [17]. Un profil UML regroupe un ensemble de stéréotypes et tagged values. Les profils dans UML2.1 sont notés comme des paquetages UML avec le stéréotype <<profile>>.
La notion de Profil est introduite comme un moyen pour structurer et regrouper les trois mécanismes d'extension précédents (les *Tagged Values, les Stéréotypes, les Contraintes).* L'OMG a standardisé des profils dans certains domaines particuliers. On trouve par exemple : un profil UML pour les systèmes temps réel, un profil UML pour les applications

d'entreprise distribuées (EDOC) et aussi des profils UML pour les plateformes EJB et CORBA.

En plus de ces profils standardisés, les utilisateurs peuvent définir leur propre profil et certains ateliers comme Objecteering de Softeam permettent de les implanter et les intégrer dans l'environnement de modélisation. En plus des stéréotypes et des tagged values, un profil UML peut contenir aussi des contraintes et des règles. Les contraintes spécialisent la sémantique des éléments du méta-modèle de référence sous le profil. Les règles dans un profil UML décrivent en particulier comment le profil est utilisé. Les règles dans un profil sont souvent définies en utilisant un langage de transformation de modèle. L'atelier Objecteering par exemple propose un langage appelé J pour la définition des règles d'un profil.

Les Profils d'utilisation d'UML se traduisent par :

- des concepts spécifiques non décrits par UML ;
- des concepts métier, spécifique à un domaine
- des contraintes d'utilisation du langage (des diagrammes)
- des processus métier, guide de style, schémas type

L'intégration de certains langages et approches à UML se fait via un Profil.
Exemple d'approches : le Profil MARTE pour le temps réel.
Exemple de Langages spécifiques : AADL : langage de description d'architecture
SDL : langage spécifique pour les Télécom.

2.3.2 État de l'art sur la modélisation des systèmes embarqués avec les profils UML.

Plusieurs profils UML ont été adoptés en vue d'adapter et/ou de compléter une approche voire un langage spécifique à un domaine particulier .Les plus notables pour les systèmes embarqués sont :

2.3.2.1 SYSML.

Le langage de modélisation SysML(System Modeling Langage)a été développé en Mars 2003 Il s'inscrit dans une approche IDM (ingénierie dirigée par les modèles ou MDE -model driven ingénierie) .Il s'adresse plus précisément aux systèmes complexes .Ces systèmes peuvent inclure du matériel et du logiciel. Les domaines technologiques concernés sont l'informatique, l'électronique, la mécanique, la physique, la chimie ….

SYSML est un profil UML donc une spécialisation d'UML dédiée aux systèmes complexes que l'on rencontre en particulier dans le domaine automobile avionique automatique et communication.

Comparé à UML, SysML permet une meilleure modélisation de la traçabilité et des exigences (diagrammes des exigences) .SysML est mieux adapté pour l'expression des contraintes sophistiquées liées à des lois physiques (diagrammes paramétriques) [37,38].
La Figure 2.1 traduit une Comparaison succincte de SysML 1.0 avec UML2.1. Le texte dans la figure résume les différents diagrammes disponibles dans SysML.
Les diagrammes des Exigences, des Paramétriques et des allocations sont disponibles uniquement dans SysML. Les diagrammes de blocks et celui des Activités sont réutilisés à partir d'UML2.1 et étendus à SysML.

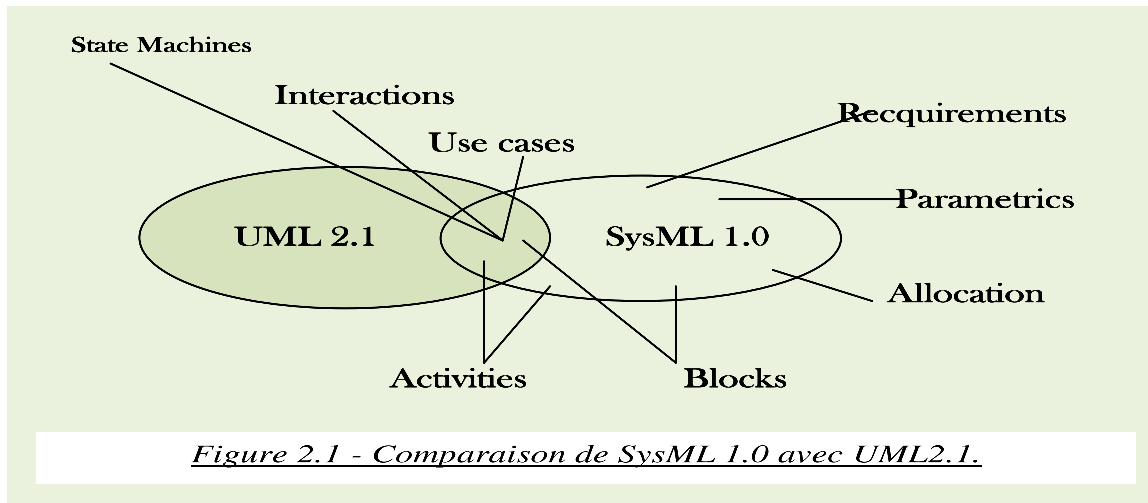


Figure 2.1 - Comparaison de SysML 1.0 avec UML2.1.

Enfin, les états Machine, les interactions et les cas d'utilisation sont réutilisés à partir d'UML2.1 sans modification.

SysML est un langage de modélisation pour représenter les systèmes et les architectures de produits, aussi bien que leurs comportements et leurs fonctionnalités. Il s'appuie sur l'expérience acquise dans la discipline du génie logiciel dans la construction d'architectures logicielles en UML (un diagramme classique des classes.) .L'architecture représente la réalisation des éléments réalisant l'aspect fonctionnel de leur produit. L'aspect physique est parfois représenté aussi, par exemple lorsque l'architecture représente la manière dont le logiciel est déployé sur un ensemble de ressources à traitement.

SysML est destinée à supporter l'étape de la conception du cycle de vie du produit. Cette étape est précédée par la décomposition des besoins de l'utilisateur en features du produit. SysML permet la représentation de ces features comme des exigences du modèle.
À leur tour, ces exigences ne peuvent être attribuées aux cas d'utilisation, aux sous-systèmes et aux composants (qu'ils soient techniques ou Physiques) identifiés pour le produit [37].

La phase conceptuelle nécessite la spécification des différents sous-systèmes et la nécessité de détailler davantage dépend de leur niveau d'intégration.
SysML offre un ensemble de mécanismes pour supporter la description de la structure du produit. Les blocs sont utilisés pour modéliser des sous-systèmes et des composants, et les ports supportent la description de leurs interfaces.
Les dépendances entre les propriétés structurelles sont exprimées à l'aide des contraintes et représentées à l'aide du diagramme paramétrique.

En plus de la structure, l'étape de la conception doit préciser la manière dont le comportement du produit s'exprime à travers l'interaction de ses composants. Par exemple, la modélisation du comportement donne une description détaillée des cas d'utilisation du produit.
SysML offre trois moyens pour expliquer le comportement du produit, à savoir les interactions, les machines d'état et d'activités. Ces trois mécanismes sont construits comme un concept unifié de comportement et peuvent par conséquent être orchestrés dans un seul, uniforme et complexe modèle de comportement pour l'ensemble du produit.
Un produit complexe est un modèle sous forme de plusieurs sous-modèles de nature différente (par exemple les exigences, les blocs, les contraintes, les activités, etc.) .SysML

offre un mécanisme reliant ces différents aspects du modèle tout en faisant respecter la traçabilité à travers ce même modèle [38].

La phase conceptuelle précède l'élaboration détaillée des éléments à l'intérieur des différentes disciplines de l'ingénierie. La conception joue donc des roles centraux dans le cycle de vie du produit et nous le soulignons ci-dessous dans quelques-uns des plus importants roles :

-L'allocation des exigences aux éléments du modèle à une étape précoce veille à ce que ses besoins soient couverts et fournit une justification pour l'ingénieur en charge de la satisfaction de ces exigences.

-La rationalisation de la conception, un outil de communication couvrant l'organisation, les niveaux et les étapes du cycle de vie. Elle améliore la communication à l'intérieur des équipes, entre les équipes (de différentes disciplines de l'ingénierie) et entre les équipes et les décideurs. Elle utilise un langage générique (dans le sens où il n'est spécifique à aucune des disciplines de l'ingénierie) qui tient compte de l'augmentation du détail de la représentation du produit. Ce dernier aspect permet de faire face à la contrainte des niveaux d'organisation. A noter également que cette description formelle est bien adaptée aux méthodes.

-Le modèle SysML fournit une représentation graphique du produit, considérée comme un levier pour les outils de décision. Les études Trade-off sont effectuées par l'évaluation des fonctions du modèle (fonction de coût, l'estimation de l'effort d'intégration.) [37,38].
A un stade précoce du cycle de vie, souvent des estimations brutes sont utilisées, c'est pourquoi le modèle ne doit pas nécessairement avoir une grande quantité de détails afin qu'il soit utilisé de manière efficace. Lorsque les détails sont ajoutés, ou les artefacts (par exemple des simulations de sous-systèmes) sont produits par l'ingénierie détaillée, le modèle est utilisé pour orchestrer les différentes simulations et effectuer la vérification des exigences.
Par conséquent le modèle SysML est un outil de décision évolutif dans l'ensemble du cycle de vie du produit, et non au niveau de l'étape conceptuelle seulement.

-Le modèle de produit représente une abstraction d'artefacts qui sont progressivement élaborés tout au long du cycle de vie. Ces artefacts sont distribués dans l'ensemble des disciplines de l'ingénierie qui participent à la conception.

Ainsi, les modèles de formulaires de traçabilité fournissent un moyen de mesurer le progrès du développement, réaliser l'analyse des incidences des changements, et gérer les dépendances entre les processus et les artefacts produits. Le modèle SysML est donc un outil de gestion et d'intégration pour toutes les parties prenantes d'un projet informatique.

2.3.2.2 MARTE :

MARTE est un profil UML pour la modélisation et l'analyse des systèmes temps réel et embarqués .Elle a été adopté en juin 2007.
Une partie (un sous-profil) importante de MARTE concerne la modélisation du temps
En février 2005, l'OMG a voté la RFP (Request for Proposals) MARTE(14) . Cette demande de propositions portait sur un profil UML pour la modélisation et l'analyse des systèmes temps réel et embarqués (UML profile for Modeling and analysis of réal-time and Embedded Systems) .Un consortium appelé ProMARTE a soumis une proposition qui a été adopté le 29 juin 2007.Cette proposition est dés lors entrée dans sa phase de finalisation :

Une FTF (finalisation Task force) a été nommée. La FTF doit assurer le suivi des demandes de corrections soumises à l'OMG et fournir les implémentations du profil.[39]

Le profil « UML pour MARTE » a pour objectif d'étendre UML pour l'utiliser dans une approche de développement dirigée par les modèles de systèmes temps réel et embarqués. MARTE fournit des supports pour les étapes de spécification, de conception et de vérification/validation.

Ce profil remplace le profil UML SPT (c. section suivante) (UML Profile for schedulability , Performance and time) qui devait être alignée sur UML 2 et étendu . Les retombées attendues de l'usage de ce profil sont de :

- Fournir une modélisation unifiée pour les parties matérielles et logicielles du système ;
- Permettre l'interopérabilité entre les outils de développement utilisés en spécification, en conception, en vérification et en génération de code ;
- Faciliter la construction de modèles sur lesquels on peut faire des prévisions quantitatives tenant compte des caractéristiques du matériel et de logiciel.

Nous nous limiterons dans ce qui suit à présenter brièvement une partie de MARTE relative au temps, à travers des concepts -liés au temps- et leurs utilisations.

a- **Le profil SPT**. Le profil UML 1.4 « Schedulability Performance and Time" avait pour objectif de combler les lacunes d'UML 1.4 pour les concepteurs et développeurs d'applications temps réel. Ce profil permet d'annoter les éléments de modèle par des informations quantitatives relatives au temps .Ces informations sont ensuite utilisées pour des analyses de performance d'ordonnançabilité ou de vérification du respect de contraintes temps réel.SPT ne considère qu'un temps métrique qui fait référence au temps physique .Il a introduit les concepts d'instants, de durée, d'événements et de stimulus liés au temps .Il modélise les mécanismes temporels (clock et timer) et les services associés (démarrage ,arrêt ,suspension reprise). L'alignement de SPT sur UML 2 est l'un des objectifs du profil MARTE.

b- **UML 2** a ajouté des méta-classes pour prendre en compte sous forme simpliste le temps. Ces mécanismes sont regroupés dans le paquetage *simple-time* qui fait partie du paquetage *commonbehaviors* d'UML métastructure .Malheureusement et malgré ces mécanismes d'UML 2, le temps reste extérieur à UML .Ceci est du principalement aux raisons suivantes :

-Il n y a aucune indication précise sur la signification des valeurs temporelles .Elles sont exprimées en une unité de temps inconnue par UML.
-Ce modèle ne prend pas en compte les effets relativistes rencontrés dans les systèmes répartis
-Il ignore les imperfections des horloges.
-UML 2 lui-même recommande dans son chapitre *commonbehavior* des modèles de temps plus élaborés fournis par un profil approprié à cause des phénomènes suscités. Et c'est la aussi une des missions du profil MARTE.

c- ***MARTE et le temps*** **:**

Il y a des sous-profils qui gèrent le temps à travers des versions de MARTE améliorées, qui étendent largement le modèle simple Time de la première version de MARTE. Ces modèles permettent de faire référence explicitement à plusieurs référentiels de temps, par l'intermédiaire d'horloges (gérées par le stéréotype Clock).

Le temps peut être de nature dense ou discrète. MARTE dans sa dernière version modélise aussi bien le temps physique que les temps logiques. Les horloges donnant accès au temps physique sont appelées des horloges chronométriques.
MARTE permet d'exprimer des différences dans la perception du temps, différences dues par exemple à la distribution spatiale du système.
On peut également modéliser des perceptions imparfaites caractérisées par des propriétés attachées aux horloges : stabilité, décalage, dérive, etc. .

Les temps logiques, auxquels on accède par des horloges logiques, s'avèrent très utiles en phase de conception. Les horloges peuvent être associées aussi bien à des éléments de modèles matériels (cycles d'un processeur, par exemple) qu'à des éléments applicatifs (itérations contrôlées).
Une autre caractéristique du modèle de temps de MARTE est de pouvoir lier directement et explicitement des éléments comportementaux (TimedEvent et TimedProcessing) au temps. Il en est de même avec les contraintes (TimedConstraint) et les observations (Timed Observation).

UML ayant des objectifs pragmatiques, il a fallu offrir des facilités d'utilisations des concepts temporels. Cette aide est fournie sous forme de bibliothèques et de langages spécialisés.
Le langage d'expression de contraintes d'horloge permet en particulier d'exprimer de façon concise des dépendances, parfois complexes, entre instants d'horloges différents. Une représentation purement graphique de ces contraintes n'est pas réaliste.
D'un point de vue pratique, il y a des plug-ins Eclipse qui implémentent le nouveau profil MARTE et font l'analyse syntaxique des langages spécialisés.

Nous allons présenter dans ce qui suit l*e méta-modèle de temps de MARTE.*
La vue domaine appelée aussi méta-modèle a pour objectif de spécifier les concepts qui vont être utilisés dans le profil, les relations existant entre ces concepts, ainsi que les contraintes imposées sur leur composition.

Ce méta-modèle doit être défini pour répondre aux besoins du domaine, sans se soucier d'UML et des limitations qu'il est susceptible d'imposer.

Dans la vue domaine de MARTE, les concepts de bases sont définis dans le paquetage *CoreElements* qui lui-même est divisé en deux paquetages : *Foundations et Causality.*

Le premier introduit les concepts d'éléments de modèle (ModelElement), de classeur (Classifier) et instance (Instance).

Le second traite des concepts comportementaux (CommonBehavior) dans lequel sont introduits les événements, les exécutions et les contextes d'exécution.

Le domaine de temps de MARTE réutilise certains de ces concepts. Il a été décomposé en quatre grands sous-domaines :

1- TimeStructure qui rassemble les concepts définissant le modèle de temps constitué d'ensembles d'instants, les instants étant partiellement ordonnés.
2- TimeAccess qui regroupe les moyens d'accès à la structure du temps. On y trouve en particulier le concept de Clock.
3- TimeValueSpecification qui permet de dénoter instants et durées.
4- Time-Usage qui introduit les éléments de modélisation couramment utilisés : événements temporels, comportements temporels et contraintes temporelles.

- *Structure associée au temps.*

Le modèle de temps retenu dans MARTE (Figure 2.2) [39] est un ensemble de bases de temps soumis à des contraintes.

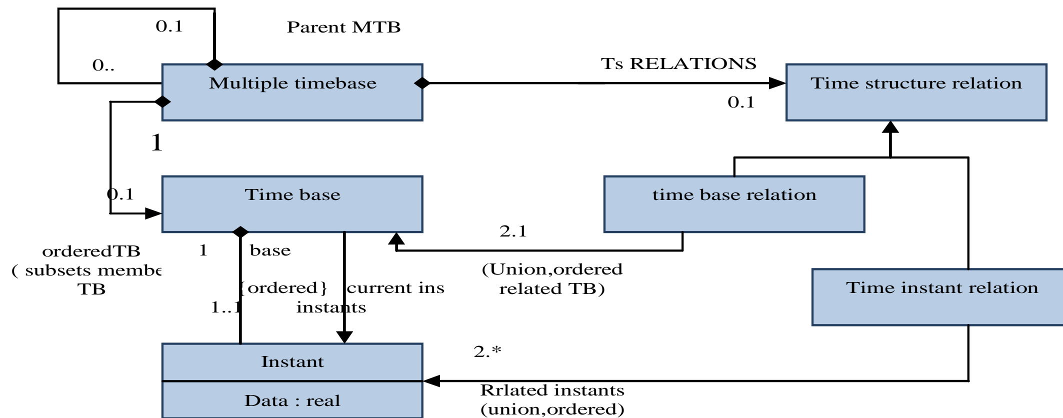


Fig. 2.2 .Modèle de temps dans MARTE

Une base de temps (TimeBase) est un ensemble totalement ordonné d'instants.

A chaque instant est associée une information numérique (date). L'ensemble des instants d'une base de temps peut être discret ou dense.

La vision linéaire du temps apportée par les bases de temps est insuffisante pour la plupart des applications, en particulier pour les applications multithreads ou pour les applications réparties. Il faut alors utiliser des bases de temps multiples (MultipleTimeBase) constituées de plusieurs bases de temps.

La structure de temps d'une application peut être une hiérarchie (arbre) de bases de temps multiples. Les bases de temps sont à priori indépendantes. Elles deviennent dépendantes quand leurs instants se trouvent liés par des relations (relation de coïncidence ou relation de causalité).

La classe abstraite TimeInstantRelation a pour sous-classes concrètes les classes CoincidenceRelation et PrecedenceRelation. Pour deux instants appartenant à deux bases de temps distinctes, la première relation indique que ces instants sont coïncidents, alors que la seconde relation indique qu'un instant précède nécessairement l'autre.

Le modèle de temps choisi peut être utilisé lors de la conception du système. La coïncidence n'est donc pas nécessairement interprétée comme un point de l'espace-temps (interprétation relativiste) ; elle peut représenter des points de synchronisations ou de simple choix de conception.

Au lieu d'imposer des dépenses locales entre instants, le profil impose directement des dépenses entre bases de temps. Une TimeBaseRelation (ou plus exactement une de ses sous-classes concrètes) spécifie de nombreuses (souvent une infinité de) relations entre instants. Une annexe de MARTE à été élaborée pour décrire les principales relations entre les bases de temps et propose un langage d'expression de ces relations (ClockconstraintSpecificationLangage).

2. 3. 2. 3 Le langage AADL

AADL (Architecture analysis & design langage) pour analyse de l'architecture et langage de conception : est un langage émergeant développé sous l'autorité de la SAE (Society of Automotive Engineers) pour répondre aux besoins spéciaux des systèmes embarqués temps-réel critiques tels que les systèmes avioniques.

Elle est conçue pour permettre la génération de systèmes exécutables et l'expression des caractéristiques des éléments du système ainsi que sa traduction en langage de programmation. Elle est dédiée spécialement pour la description des éléments matériels et logiciels des systèmes embarqués sous forme de plusieurs représentations :

- texte
- XML
- notation graphique
- profil UML 2, et représentation en UML 1.4.

La description d'une architecture en AADL consiste essentiellement à la représentation d'une architecture dynamique du logiciel associée et liée à une plateforme d'exécution [40,41].
En pratique, elle se décrit à l'aide de composants hiérarchisés (ils peuvent contenir d'autres composants) et interconnectés. Les composants communiquant entre eux par des liens fonctionnels (flots de données et de contrôle) et par des liens physiques (bus, mémoire, réseau,…) (cf. figure 2.3).
Des informations telles que les exigences temporelles, les modèles de fautes et d'erreurs, le partitionnement temporel et spatial ou les besoins de sûreté et de certification peuvent être modélisés et associés aux composants à l'aide de propriétés.

a. Les concepts AADL:

-Concept de composant.

Les composants sont les éléments de base d'une architecture AADL. Ils sont hiérarchisés, composés et interconnectés [43].

Ils appartiennent à l'une des catégories prédéfinies :

-catégorie composite
-catégories logicielles
-catégories plate-forme

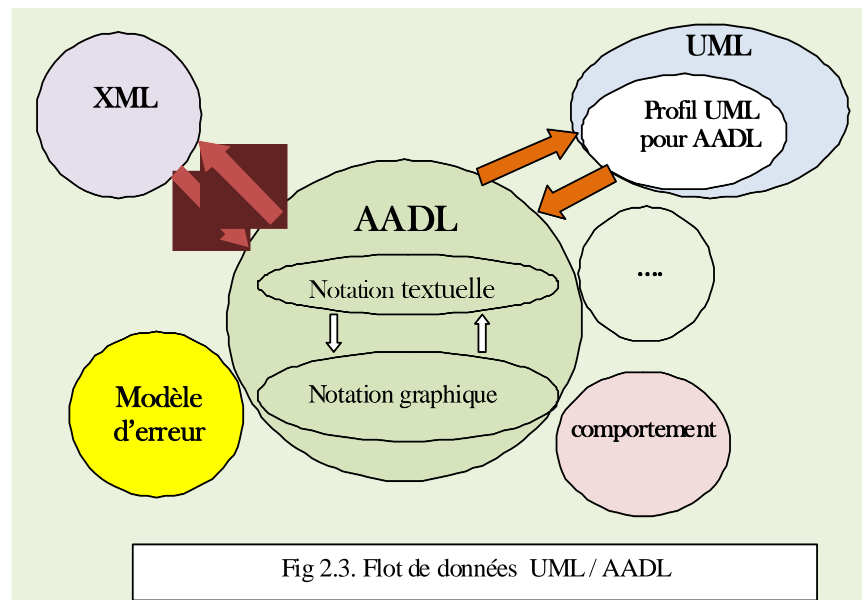


Fig 2.3. Flot de données UML / AADL

Catégories de composants. dans laquelle on trouve : Les threads ; les processus ,les sous-programmes ; les bus ; les mémoires ; les groupes de threads ; les systèmes , les données ; les périphériques et les processeurs .
On peut trouver aussi des Catégories prédéfinies dans lesquelles on doit spécifier :
Le Type, la spécification de l'interface externe l'Implémentation. La spécification du contenu se fait par une instance. L'instanciation d'un type ou d'une implémentation se fait par : type ;

implémentation et instance. Un type peut hériter d'un autre type et une implémentation peut hériter d'une autre implémentation.

-Concept de caractéristique.

Une « caractéristique » (*feature*) est un élément de type AADL de composant qui spécifie comment ce composant s'interface avec les autres composants du système.
Il y a quatre catégories de caractéristiques :

- Les ports ;
- Les accès à un sous-composant ;
- Les sous-programmes ;
- Les paramètres.

Port

Un « port » représente une interface de communication pour échanger des données ou des événements entre composants.
Trois directions sont possibles :

- port entrant (*in*) ;
- port sortant (*out*) ;
- port bidirectionnel (*in out*).

Trois types de ports :

- port donnée (*data port*) ;
- port événement (*event port*) ;
- port événement-donnée (*event data port*).

Groupe de ports

Les groupes de ports (*port group*) peuvent regrouper des ports et d'autres groupes de ports.

Sous-programme et paramètres

Une caractéristique « sous-programme » (*subprogram*) représente soit un point d'entrée d'exécution du code source qui opère sur un composant donnée (*data subprogram*) ; soit un point d'entrée pour un appel de procédure à distance (*server subprogram*).
Une caractéristique sous-programme fait référence à un composant sous-programme.
Un « paramètre » représente un argument de sous-programme.

-Concept de propriété.

Une « propriété » (*property*) fournit une information sur un composant, une caractéristique, une connexion, un mode ou un appel de sous programme.Une propriété est définie par : un nom ; un type et une valeur.
Le type de propriété définit l'ensemble des valeurs acceptables. On peut définir de nouvelles propriétés dans un *property set*.
Les Propriétés représentent des ensembles de propriétés pré déclarés. Deux ensembles de propriétés sont pré déclarés :
AADL_Properties définit des propriétés communes à toutes les spécifications AADL;
AADL_Project définit des types et des constantes énumérés, dont les valeurs peuvent différer d'un projet à l'autre.
Ces deux ensembles de propriétés font implicitement partie de toute spécification AADL.
Les types et constantes de ***AADL_Project*** peuvent être modifiées.

-Concept de Flux.

Pouvoir spécifier des flux de bout en bout permet des analyses temporelles, de fiabilité, de propagation d'erreur, de qualité de service, etc.
La description de flux complets est composée de plusieurs sortes de déclarations :

Spécification de flux ;
Implémentation de flux ;
Déclaration de bout en bout.

-Concept de Mode.

Un mode représente un état opérationnel d'un système
Les Modes existants sont :subsys 1 ; subsys 2 ; subsys 3 ; mode 1 ; mode 2 et mode 3.

-Concept de Paquet.

Un paquet (*package*) fournit le moyen d'organiser les descriptions en introduisant des espaces de noms. Un paquet peut contenir des types de composants ; des implémentations de composants ou des librairies annexes.
Le contenu de la section *public* est visible hors du paquet et le contenu de la section *private* n'est pas visible hors du paquet.

- Les Annexes.

Une annexe permet d'utiliser des déclarations exprimées dans un autre langage qu'AADL. L'usage principal est le support de nouvelles méthodes d'analyse ou de description de nouveaux aspects. Une clause annexe peut être utilisée dans un type ou une implémentation de composants, par contre une bibliothèque d'annexes peut être déclarée dans un paquet.
Exemple : introduction de contraintes avec OCL : {** nom annexe **}.

b. AADL comme profil UML :

AADL comme profil UML se compose d'un ensemble de (voir tableau 2.1) [40]:

-différents types de stéréotypes : stéréotypes abstraits (entités conceptuelles : espace de nommage, type et implémentations de composants, features, etc.) et stéréotypes concrets (Entités AADL, implémentation et type système, port de données connections de données…).
-Contraintes : qui assurent la consistance et la cohérence et limitent les options de composition - cf. figure 2.4- [40].
-Propriétés relatives aux stéréotypes : ce sont les propriétés standard AADL.

AADL Concept	UML Profile
Component Type	Stereotype aadl/Thread Type,... Extend UML Class
Feature	Stereotype aadl/dataPort,... Extend UML Property
Component Implementation	Stereotype aadl/Thread/mplementation,... Extend UML Class
Subcomponent	Stereotype aadl/Thread ,... Extend UML Property
Package, Property Set	Stereotype aadl/Package, aadl/PropertySet Extend UML Package
Connection	Stereotype aadl/dataConnection,... Extend UML Association

- TABLEAU 2 .1 - *Eléments du profil UML*

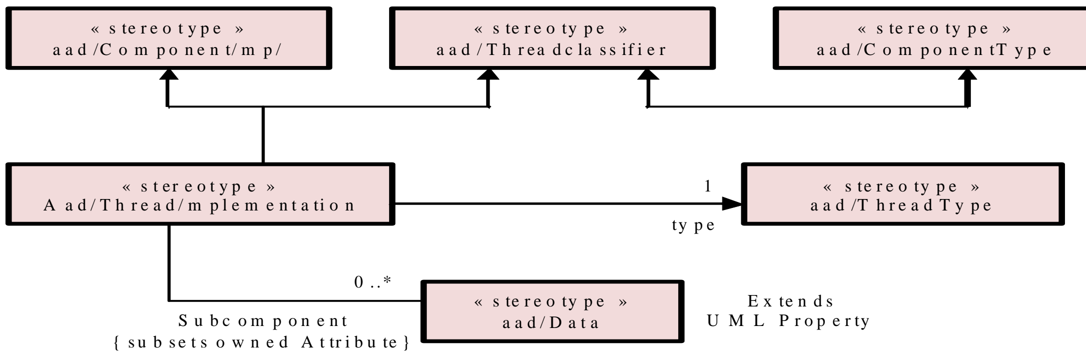


Fig 2.4- Les contraintes du profil UML

- Exemple de profil UML :
La figure 2.5 illustre un exemple de profil UML pour AADL [40].

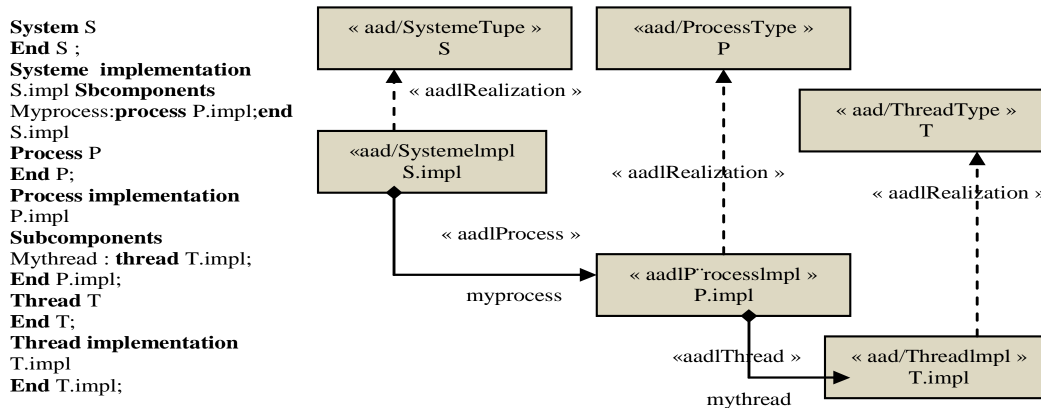


Fig 2.5 - Exemple de profil UML pour AADL .

c. Translation DSL – profil UML .

Un pont entre les 2 approches doit passer par une translation entre les modèles AADL conformément au méta-modèle AADL méta-model et par des modèles AADL conformément à AADL comme profil UML.
Il Permet notamment l'utilisation d'outils matures d'UML pour le développent de modèles, et l'utilisation des techniques spécifiques d'analyse d'AADL basées sur le méta-modèle AADL.
Il a comme objectifs principaux d'activer rapidement un outil commercial qui supporte AADL et d'accélérer la transition d'AADL dans l'industrie.
Le Processus de transformation suit les étapes suivantes :

1. créer le modèle AADL en utilisant un outil UML
2. traduire au méta modèle AADL
3. exécuter des analyses (exemple dans OSATE)
4. traduire au profil UML
5. Raffiner le modèle UML

d. Apport de L'outil UML.

L'outil UML ajoute :

- le contrôle de la gestion des Versions / configurations
- l'intégration avec d'autres outils, ex. la gestion des exigences.

```
rule SystemType {
from s : AADL!SystemType
to t : UML!Class (
name <- s.name
)
do {thisModule.applyStereotype(t,"aadlSystemType");
}
}
```

Fig. 2.6 - Exemple - Transformation à partir d'AADL vers le profil UML

e. Conclusion

Les ADL sont un élément de solution pour répondre aux besoins de l'industrie.
Les industriels sont engagés dans des projets de recherche pour les mettre en œuvre. Les ADL sont utilisés dans le cadre d'un processus défini.
AADL est un des ADL les plus prometteurs pour ces domaines. Il offre une grande souplesse de modélisation par son degré de modélisation selon les besoins, sa simple description, sa génération de code et sa simulation et analyse .Il peut être utilisé comme langage fédérateur qui permet l'exploitation par plusieurs outils différents : générateur, analyseur, visualiseur.

Il offre une modélisation concrète et intervient dans la dernière phase avant la génération / déploiement du système et peut être utilisé dans une chaîne de modélisation. C'est un langage qui permet une conception avec des formalismes plus abstraits (UML, … etc.) et avec lequel une génération d'une modélisation intermédiaire est possible .Il permet également une analyse puis une génération et un déploiement de l'application.

2 .3. 2. 4 UML et SystemC

a. Survol de SystemC :

SystemC est comme Verilog et VHDL, un langage de description de matériel. Les spécifications de SystemC ont été étendues en 2001 à la modélisation de systèmes abstraits (de très haut niveau, avant le partitionnement matériel / logiciel), aboutissant à la version 2.0.

SystemC est une bibliothèque C++, qui fournit des classes C++ adaptées à la modélisation de matériel ainsi qu'un moteur de simulation événementiel rapide, le tout sous License libre.
Il permet de garder un même langage d'un bout à l'autre du flot de conception, excepté à sa fin (à ce niveau on dispose de moyens de s'assurer de la cohérence des modèles).

Un modèle SystemC est écrit en C++.
Les modules sont les briques structurelles de base. Ils instancient d'autres modules, ainsi que des canaux et des processus. Ils communiquent avec l'extérieur par des ports. La

communication proprement dite se fait au travers de canaux reliés au port. Pour qu'un canal soit relié à un port, les deux doivent avoir la même interface (déclaration de fonctions). L'implémentation de ces fonctions est faite dans le canal.

b. Objectifs de SystemC.

SystemC a pour objectif de modéliser des systèmes numériques matériels et logiciels à l'aide de C++. Il modélise non seulement des systèmes matériels, mais aussi des systèmes logiciels, mixtes ou non-partitionnés. [44 ,45,46] .La modélisation d'un système complet passe par des niveaux comportementaux plus détaillés que ceux abordés précédemment. La description de ces niveaux se fait à l'aide des termes suivants :

-BCA (Bus Cycle Accurate) : s'applique à l'interface d'un modèle.
Il signifie que la modélisation des transactions sur l'interface est correcte au cycle prés.
Un modèle BCA n'apporte aucune information sur les bits (signaux) de l'interface.

-BA (Bit Accurate) : s'applique à l'interface d'un modèle et à la fonctionnalité d'un modèle.
Il signifie que la modélisation des transactions sur l'interface est BCA, et porte aussi sur les Signaux de l'interface. La modélisation est précise au bit (fil) prés.

-UTF (UnTimed Functional) : s'applique à l'interface et à la fonctionnalité d'un modèle.
Le modèle ne comporte aucune notion de durée d'exécution, mais seulement un ordre éventuel dans l'exécution des événements.
Chaque événement s'exécute en un temps nul. Seul compte l'ordonnancement des événements.

-TF (Time Functional) : s'applique à l'interface et à la fonctionnalité d'un modèle.
Le modèle comporte des notions de durée (temps d'exécution des processus, latence, temps de Propagation, ...)

-RTL (Register Transfert Level) : s'applique à l'interface et à la fonctionnalité d'un modèle matériel.
Chaque bit, chaque cycle, chaque registre du système est modélisé.

SystemC étant une extension de C++ (donc du C), un seul et même langage est utilisé de la première étape du flot de conception jusqu'a l'étape TLM (BCA), que ce soit pour la partie logicielle que pour la partie matérielle. Arrivé au stade BCA, ou à un niveau suffisamment bas et raffiné, on peut :

-Soit utiliser des synthétiseurs comportementaux, qui transforment le SystemC directement en Portes.

-Soit transcoder les blocs matériels en SystemC / Verilog / VHDL RTL puis utiliser un flot de synthèse habituel. Lors de ce transcodage, les blocs sont suffisamment simples pour que la probabilité d'erreur soit minime, ou qu'on puisse utiliser des outils de preuve d'équivalence, si le transcodage est manuel,ou qu'on puisse le faire de façon automatique.

A chaque étape, le même environnement de test (écrit en SystemC) peut être utilisé, garantissant la fonctionnalité du système (matériel + logiciel). Si on arrive à l'utilisation de Verilog ou VHDL, un cosimulateur se charge de simuler les deux langages en même temps.

c. Organisation de SystemC.

SystemC est une extension du langage C++, plus précisément une bibliothèque de classes C++ contenant des modèles de matériel, ainsi que toutes les briques de base pour modéliser un système matériel qui n'aurait pas été inclus dans la bibliothèque de base [44].
Cette bibliothèque C++ comprend aussi en standard un moteur de simulation événementiel rapide, permettant de se passer d'un simulateur externe.
La figure 2.7 ci-dessous résume l'architecture de la bibliothèque. Elle se base sur le langage C++ et Intègre en standard un moteur événementiel. La figure 2.8 ci-dessous montre l'organisation de SystemC.

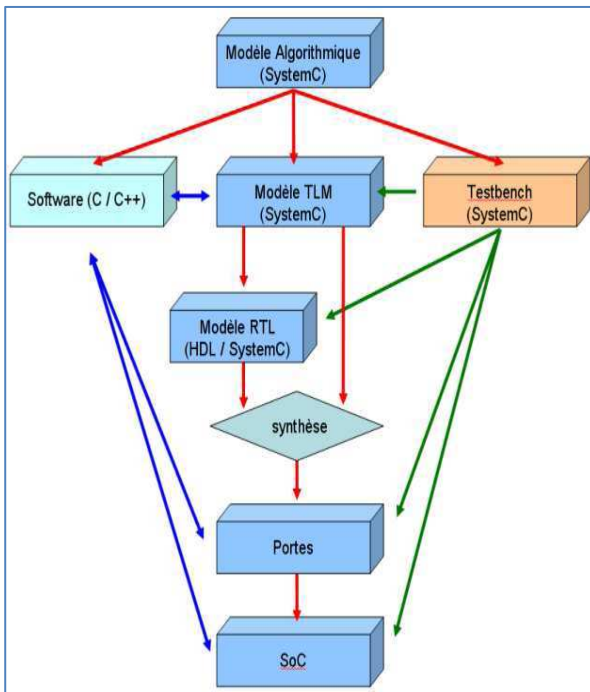


Fig. 2.7 Flot de conception SystemC.

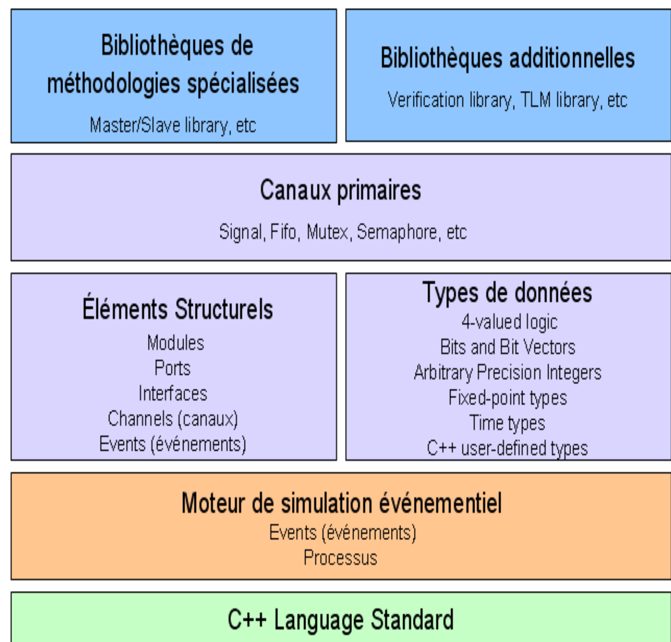


Fig. 2.8 Organisation de SystemC

Les types de données usuels C++ ont été enrichis par des types de données adaptées au matériel (logique multivaluée, décimaux ,virgule fixe, ...) [44]. Les autres éléments se répartissent en deux ensembles: les éléments structurels qui représentent la structure des systèmes matériels (modules, ports), et les éléments de communication entre processus et modules (événements, canaux).

Certains des éléments de communication, les canaux, qui sont fréquemment utilisés, font partie de SystemC. On les appelle les canaux atomiques ou primaires (car ce sont des briques de base), et regroupent les signaux, les FIFOs, ... Certains de ces types de données servent à représenter du matériel (modules, ports, signaux, FIFO), d'autres à représenter du logiciel (sémaphores, Mutex), et certains sont suffisamment abstraits pour pouvoir représenter les deux (canaux, événements, interfaces, ...).
Un modèle écrit en SystemC est donc un programme C++ utilisant les classes de la bibliothèque SystemC. Il doit donc être compilé.

Le processus de développement est le même que pour tout programme C/C++ :

-Compilation des librairies SystemC .
-Compilation des parties C++ avec g++
-Compilation des parties C avec gcc.
-Liaisons des objets pour produire un exécutable, en utilisant éventuellement toutes les bibliothèques désirées (libpng, libm, libssl, ...)

d. Structure d'un modèle SystemC.

Nous allons présenter un aperçu rapide de la façon dont est construit un modèle SystemC d'un système numérique (Les différents composants structurels de SystemC -cf. figure 2.7 -) :

Hiérarchie
Comme en Verilog ou VHDL, un système est une hiérarchie d'objets. Généralement ce sont des modules imbriqués et/ou des processus. Les modules communiquent entre eux par des canaux. Le plus haut de la hiérarchie d'un système complet (module à tester + testbench) n'est pas un module top (comme avec Verilog/VHDL), mais la fonction SystemC *sc_main* (l'équivalent de main des programmes en C).

Modules
Un module en SystemC est composé d'autres modules, de canaux de communication entre ces modules (signaux, ou canaux plus abstraits), et éventuellement de processus.

Ports :
Un module possède un ou plusieurs ports. Les ports sont juste des points d'entrée ou de sortie, qui ne font rien de particulier. Par contre, les ports doivent déclarer les fonctions qui seront utilisées pour communiquer à travers eux.
Exemples :
-Un port en entrée destiné à être relié à un signal normal déclare qu'il utilise la fonction *read* des signaux.
-Un port similaire mais bidirectionnel déclare qu'il utilisera les fonctions *read* et *write.*
-Un port destiné à être relié a une FIFO (un canal de communication abstrait, de haut niveau), déclarera selon le coté de la FIFO où il est sensé se trouver, qu'il utilisera les fonctions *read, nb_read* (read non bloquant*), num_available,write, nb_write, num_free...* . La déclaration des fonctions qu'il va utiliser est appelée *interface.*

Interfaces :
Une interface est une déclaration des fonctions (ou méthodes, dans la terminologie C++) qui pourront être utilisées à travers les ports d'un module. Une interface ne contient pas de code, c'est seulement une déclaration de fonctions.
Elles permettent au compilateur de détecter très tôt le branchement d'un port à un canal qui ne lui est pas adapté.

Canaux :
Les canaux sont les moyens de communication entre les modules. Ils peuvent être basiques et concrets (signaux), ou plus évolués / plus abstraits (FIFO, réseau Ethernet,...). Ils peuvent aussi contenir d'autres canaux, voire même des modules si ce sont des canaux de très haut niveau.
Les canaux contiennent aussi l'implémentation du code C++ déclarée dans les interfaces. Ils implémentent ainsi une interface.

On branche un canal à un module par un port, ce port devant déclarer l'interface implémentée par ce canal. Le port est alors lié au canal par une interface, et le contenu du module peut accéder au canal par l'intermédiaire des fonctions déclarées dans l'interface, ce qui est symbolisé dans le schéma ci-dessous qui combine port et interface.
Tout ce dont a besoin un module pour utiliser un port est l'interface à laquelle il est lié. Autrement dit, on peut modifier le contenu d'un canal, le raffiner progressivement, sans avoir à toucher quoi que ce soit d'autre. Il suffit que l'interface reste la même.

Processus :
Les processus en SystemC sont similaires à ceux de Verilog et VHDL. Ils décrivent une fonctionnalité, un comportement. Un processus ne doit pas être appelé directement; c'est le moteur de simulation SystemC qui se charge de l'appeler (le déclencheur) sur certains événements particuliers : ceux qui sont dans sa liste des sensibilités.
Les processus peuvent éventuellement communiquer directement avec les canaux. Ils n'ont pas besoin de ports pour cela, ils appellent directement les méthodes du canal.
Ainsi, on peut résumer l'architecture (ou la structure) d'un module en SystemC au modèle suivant [45]:

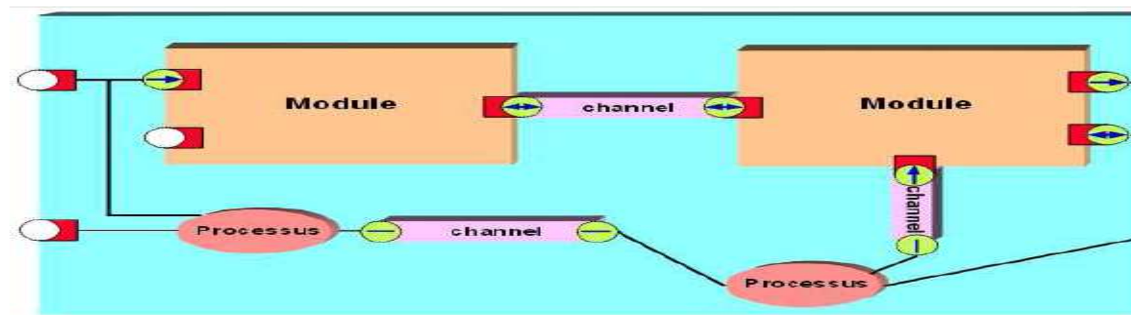


FIG 2.9 –Un exemple de structure en SystemC

e. SystemC comme profil UML.

e-1 - L'intérêt de combiner SystemC à UML :

L'industrie recherche des méthodes efficaces qui jouent le role de pont pour la conception d'application (produit) et ses méthodes d'exécution (conception basée sur les plateformes). Le lien entre ces deux mondes est établi par échange ou combinaison des modèles. L'évolution récente d'UML promet de faire de cette approche une des plus prometteuses.
Une combinaison SystemC-UML a été testée efficacement dans la conception de SOCs. Pour la spécification des systèmes la communauté des concepteurs a opté pour la version 2.x d'UML et pour l'implémentation, un flot d'implémentation basé sur SystemC. La modélisation par la structure UML fournit une représentation graphique directe des concepts SystemC. C'est Par une série successive de Mapping UML-SystemC que le concepteur peut aboutir rapidement à des programmes en SystemC très performants [46]. Le processus de conception d'une SOC basé sur une combinaison d'UML et de SystemC est avantageux compte tenu qu'il utilise des liens en se basant sur des standards, qu'il a fait ses preuves technologiques et qu'il est supporté par des outils opérationnels.

e.2 Le profil SystemC :

Le modèle SystemC est construit à partir du modèle UML.Le profil SystemC-UML est un profil UML dédié à SystemC. UML2 supporte SystemC comme structure de modélisation (cf.figure 2.10) .Les concepts de support sont disponibles dans l'outil Rational ROSE-RT. Des stéréotypes standards sont utilisés pour les concepts SystemC orientés hardware. (Exemple: les canaux élémentaires, les types de signaux hardware …)[45].

Fig 2.11 . Le mapping SystemC - UML.

SystemC	correspondance UML
Module=:	classe structurée (capsule dans ROSE-RT) .
Interface =	interface.
Port =	port
Primitive-Channel =	connecteur stéréotypé.
Hierarchical channel =	classe structurée.
Event =	événement stéréotypé.
Sensitivity :	trigger+état combinatoire.
Process=	comportement de la classe structurée ; statechart.
Wait() =	statechart receive.

Les Similarités UML/SystemC :

Les entités

package/name space
class/module
port
interface

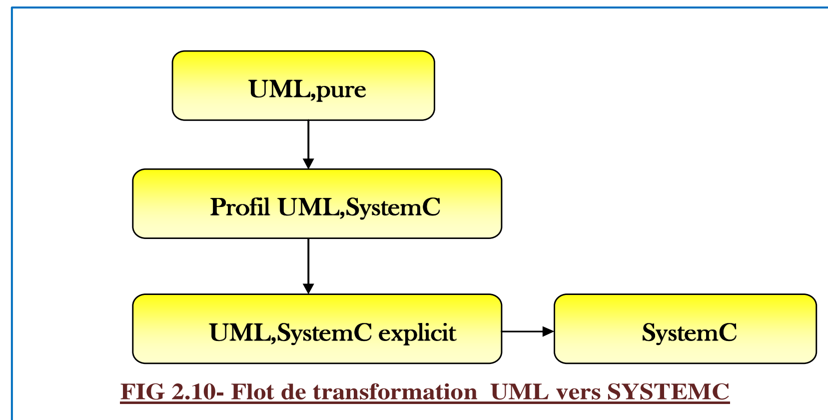


FIG 2.10- Flot de transformation UML vers SYSTEMC

Les classes structurées : sont des classes qui contiennent une structure de collaboration de ports et de parties interconnectés.
Les parties peuvent être des instances de classes structurées ou non structurées basées sur les concepts de modélisation dans les langages populaires de description d'architecture UML-RT. Les classes structurées peuvent avoir une structure interne des parties (les classes structurées) et des connecteurs. Les classes structurées peuvent être utilisées tout au long du cycle de développement et pour l'architecture système à son plus bas niveau de conception des éléments.

La communication dans UML:

-nécessite une interface
-réalise une interface
-est réalisée Sous forme de Liste d'interfaces
-nécessite des connecteurs et utilise des queues de message implicites.

La communication dans SystemC nécessite :

-des ports
-des exports
-une interface par port/export
-des canaux implémentant des interfaces
-des queues de message communes dans le canal.

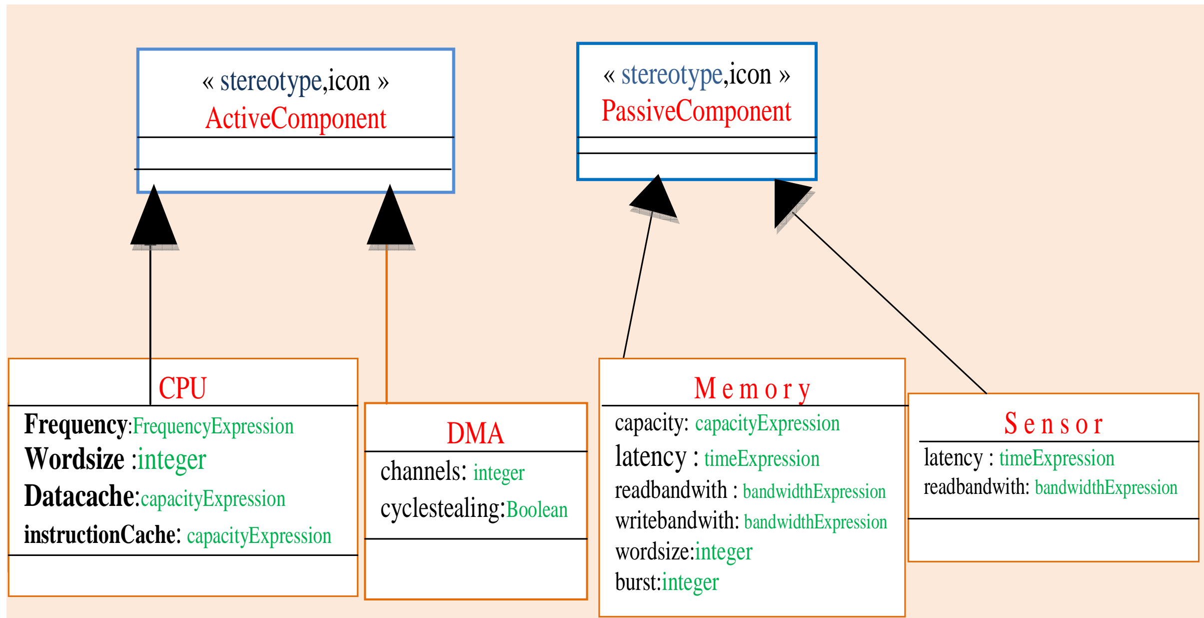


Fig 2.12 - Exemple d'une modélisation SystemC-UML de l'architecture matérielle d'un composant.

La Modélisation de la communication diffère entre SystemC et UML. C'est pour cette raison que la Structure de la communication doit être prédéfinie.
Nous allons présenter des exemples de résultats intermédiaires des quelques étapes du flot de transformations UML-SystemC (cf. figures 2.12 et 2.13) .

class IX: public scinterface {
publ i c :
 	
v i r t u a l void x (intvalu e) = 0 ;
c l a s s X s i g n a l : publ i c
UML s ignal {
publ i c :
i n t v a l u e ;
i n l i n e x s i g n a l (i n t v a l u e) :
v a l u e (v a l u e){ }
} ;
} ;
	SCMODULE(x){
	public :
	
	sc export t<IX> p 1 e x p o r t ;
	sc port <IY> p 1 **p o r t ;**
	c l a s s Ch a n n e l c l a s s
	: publ i c sc channel ,
	public i c IX {
	pr i va t e :
	s t d :: deque<UML signal !>
	queue ;
	scevent e ; public :
		Channel class(sc module name name) ;
		
		UML signal ! read ( ) ;
		void X (i n t v a l u e) ;
		} ;
		Ch a n n e l c l a s s X ch a n n e l ;
		void X t h r e a d ( ) ;
		void X : : Ch a n n e l c l a s s : : X (i n t v a l u e){
		queue . push ba ck (new X s i g n a l (va lu e)) ;
		. n o t i f y ( ) ;
		}
		X (sc module name name) : sc module (name) ,
		X ch a n n e l (" X ch a n n e l "){
		p 1 e x p o r t (X ch a n n e l) ;
		SC HAS PROCESS(X) ;
		SC THREAD(X t h r e a d) ;}

Exemple 3 **:** programmes avec des composants structurels en SystemC qui prend en compte le mapping SystemC-UML

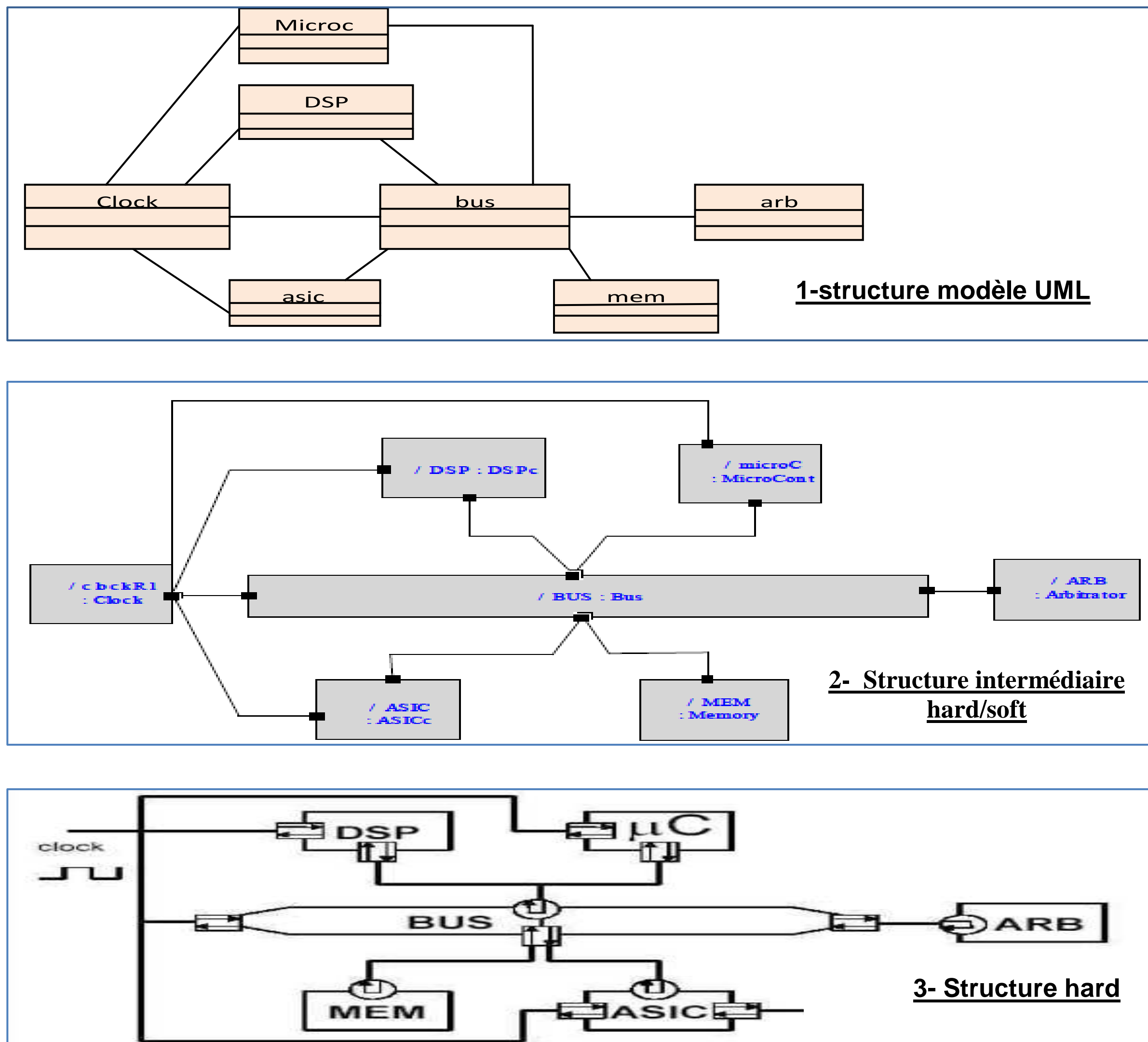


Fig. 2.13 -Exemple des 3 étapes de modélisation d'un simple bus.

e.3 Mise en œuvre :

Il existe une traduction qui génère du code SystemC à partir de modèles UML développés en utilisant l'outil Rational Rose RT. Le code est généré en SystemC synthétisable dans la mesure où il est accepté par le Synopsis Cocentric compilateur.

Le flux de conception commence par la description du système en utilisant une notation UML et produit l'implémentation des composants matériels et logiciels ainsi que leurs interfaces de communication.

L'importance du rôle de la notation UML (par exemple, le diagramme de déploiement), pourrait constituer une combinaison cohérente de la notation UML dans un flot de conception.

UML est un standard en évolution, notamment dans la direction de la conception des systèmes embarqués temps réel .Toutefois, d'importantes parties de UML, se sont stabilisées (telles que les diagrammes de classes, les diagrammes d'états les diagrammes de structure) et les concepteurs de logiciels, notamment les développeurs utilisent de plus en plus ces notations dans leur spécifications initiales.

Depuis, les descriptions basées sur UML sont très loin d'être implémentées, elles doivent être affinées et plus détaillées pour un certain nombre d'entre elles en vue d'obtenir des spécifications au niveau de l'implémentation. SystemC semble être un excellent choix pour servir de couche pour le plus haut niveau initial de conception basée sur UML.

SystemC est un langage de conception qui permet aux modèles d'être exprimés et vérifiés à des niveaux d'abstraction suffisamment hauts , tout en permettant la liaison avec l' implémentation et la vérification des matériels . En outre, SystemC, vue d'une perspective de programmation, est un ensemble de bibliothèques de classes construites au-dessus de C + + et il est donc naturellement compatible avec les méthodologies de conception orientée objet. Un mécanisme de traduction flexible tel que l'outil Rational Rose RT et entièrement automatisé et basé sur le langage de conception UML au niveau de la couche supérieure de SystemC est une stratégie prometteuse.
Cependant Rose RT peut produire du code C + + pour seulement les modèles qui impliquent des diagrammes de classes, des diagrammes d'états et des diagrammes de structure.

2.4 Le développement des Systèmes embarqués dans le cadre des lignes de produits.

2.4.1 Introduction.

La problématique durant la dernière décennie n'est plus de trouver des solutions pour développer un seul produit logiciel à la fois mais plutôt à concevoir et développer une ligne de produits logiciels -LdP- qui prend en compte les différents facteurs de variation des produits et permet de minimiser les coûts et les temps de réalisation[47].Les facteurs de variation pris en compte par une LdP de logiciels peuvent être techniques (utilisation d'une variété de matériels associés aux logiciels) commerciaux (création de plusieurs versions allant d'une version limitée à une version complète) ou culturels (logiciels destinés à plusieurs pays).
Pour illustration, on peut citer les logiciels intégrés aux téléphones mobiles qui doivent supporter plusieurs standards de communication et une variété de langues .le paradigme LdP n'a émergé comme une approche à part du génie logiciel seulement ces dernières années. Dans la littérature, une LdP est définie comme un ensemble de systèmes partageant un ensemble de propriétés communes et satisfaisant des besoins spécifiques pour un domaine particulier. La notion de variabilité est utilisée pour regrouper les caractéristiques qui différencient les produits de la même famille. Les langues supportées sont un exemple de variabilité dans une LdP du domaine des téléphones mobiles. [47].

Malheureusement UML est essentiellement destiné à modéliser un seul produit logiciel à la fois. Ceci a motivé Plusieurs chercheurs à s'intéresser à l'extension d'UML pour qu'elle prenne en charge la modélisation des lignes de produits. Ceci s'explique par deux raisons principales :

- UML est un standard largement adopté dans l'industrie et il existe plusieurs outils qui le supportent.
- UML définit des mécanismes standards d'extensions permettant d'étendre et d'adapter sa notation et sa sémantique à un domaine particulier. Les stéréotypes, les tagged values et les profils sont des exemples de ces mécanismes.

Notre travail se concentre sur l'approche lignes de produits et notamment celles dédiées aux produits logiciels destinés à être utilisés dans les systèmes embarqués.

2.4.2 Les Principales problématiques d'une approche LdP :

Les principales problématiques d'une LdP sont :

- L'identification et la gestion de la variabilité.

- La construction d'un produit particulier (ou dérivation d'un produit) qui consiste en particulier à figer certains choix vis-à-vis de la variabilité dans la LdP. Une particularité des LdP est que certains choix sont incompatibles entre eux, d'autres sont liés. Un choix particulier lors de la dérivation d'un produit peut exclure ou exiger d'autres choix.

- La gestion des contraintes permettant de faciliter les choix lors de la dérivation de produits.

2.4.3 Les Lignes de Produits Logiciels et la variabilité

- Quelques définitions :

- Un domaine est un secteur de métier ou de technologies ou de connaissances caractérisé par un ensemble de concepts et de terminologies compréhensibles par ses utilisateurs.

-Une ligne de produits logiciels est un ensemble de systèmes partageant un ensemble de propriétés communes et satisfaisant des besoins spécifiques pour un domaine particulier.
La définition ci-dessus de la ligne de produits caractérise les produits, membres de la ligne de produits, par un ensemble de propriétés communes (commonalité), mais aussi par leurs différences (variabilité).

-La variabilité regroupe l'ensemble des hypothèses montrant comment les produits, membres de la ligne de produits, diffèrent.

-La commonalité regroupe l'ensemble des hypothèses qui sont vraies pour tous les produits, membres de la ligne de produits.

b- L'ingénierie de domaine et l'ingénierie de l'application: l'ingénierie des lignes de produits logiciels, distingue deux niveaux, illustrés par la Figure 2.14 : l'ingénierie du domaine et l'ingénierie de l'application.

L'ingénierie de domaine. L'ingénierie du domaine consiste à développer et construire des assets (un asset est un élément qui permet de développer un logiciel, par exemple un document de spécification, un modèle, un code,..etc.) qui seront réutilisés pour la construction de produits ; Il s'agit d'un développement pour la réutilisation.

Dans ce premier niveau, nous distinguons trois activités : l'analyse, la conception et l'implantation du domaine. Le but de l'analyse du domaine est d'étudier le domaine de la ligne de produits et d'identifier les commonalités et les variabilités entre les produits [47].

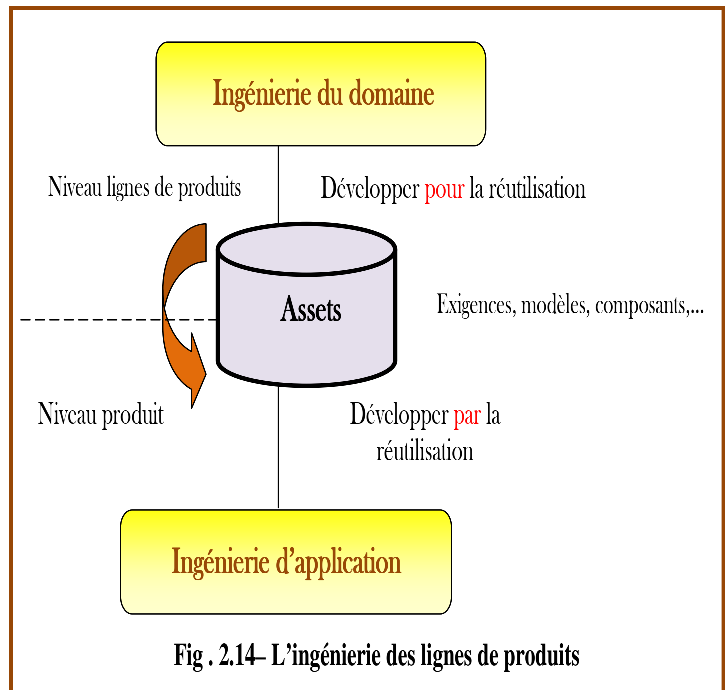


Fig . 2.14– L'ingénierie des lignes de produits

Il existe plusieurs méthodes pour l'analyse de domaine, la plus connue est FODA (FEATURE ORIENTED DOMAIN ANALYSE). Le domaine dans FODA est décrit dans un modèle de features, spécifié sous forme d'arbre dont les nœuds représentent les features ou caractéristiques du domaine et les arcs spécifient des liens de composition entre les caractéristiques.
FODA distingue trois catégories de features :
Les features obligatoires pour tous les produits membres de la LdP, les features optionnelles qui sont présentes seulement dans certains produits et les features alternatives. La Figure 2.15 montre un exemple de modèle de caractéristique FODA d'une ligne de produits de voitures. Chaque feature dans le diagramme correspond à un concept du domaine.

Le but de la conception de domaine est d'établir une architecture logicielle générique de la ligne de produits. Il n'y a pas de consensus sur la définition d'une architecture logicielle et par conséquent pour l'architecture de ligne de produits. *Une architecture logicielle de ligne de produits peut être représentée comme une architecture standard qui comporte un ensemble de composants, de connecteurs et de contraintes.* Pour les lignes de produits, l'architecture devrait être telle une architecture de référence à partir de laquelle l'architecture de chaque produit est dérivée.

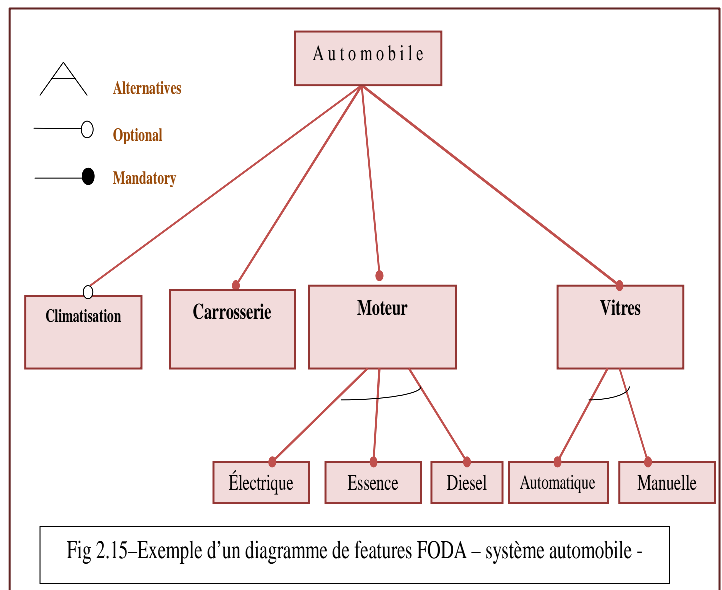


Fig 2.15–Exemple d'un diagramme de features FODA – système automobile -

La variabilité identifiée pendant l'analyse de domaine doit être spécifiée explicitement dans l'architecture de ligne de produits. L'implantation de domaine consiste à implanter l'architecture générique définie dans la conception de domaine sous forme de composants qui vont être réutilisés dans l'ingénierie d'application pour la construction de chaque produit.

L'ingénierie de l'application : l'ingénierie de l'application consiste à utiliser les résultats de l'ingénierie de domaine pour la construction, appelée aussi dérivation, d'un produit particulier. Il s'agit d'un développement par la réutilisation (cf. figure 2.14).

Les résultats de l'ingénierie de domaine (les modèles de features, l'architecture générique, et les composants) contiennent de la variabilité, la dérivation d'un produit particulier a donc besoin de décisions (ou de choix) associées à ces points de variation. La notion de modèle de décision est utilisée pour capturer et enregistrer les décisions nécessaires à la dérivation de produits.

2.4.4 La variabilité logicielle :

La commonalité et la variabilité sont les concepts centraux dans les lignes de produits logiciels. La gestion de la variabilité demande plus d'efforts que pour la commonalité. En effet, les propriétés communes dans la LdP sont identifiées et utilisées telles quelles pour la construction de tous les produits.
Cependant *la variabilité demande, en plus de son identification, des mécanismes pour sa gestion* (on parle aussi de résolution de la variabilité). La gestion de la variabilité logicielle n'est pas un nouveau problème et plusieurs techniques de conception et de programmation permettent de la gérer.
Cependant en dehors du contexte de lignes de produits, la variabilité concerne un seul produit, c.à.d. la variabilité fait partie du produit et elle est résolue après que le produit soit délivré et installé dans son environnement d'exécution. Dans le contexte des lignes de produits, la variabilité doit être explicitement spécifiée et elle fait partie de la ligne de produits et au contraire de la variabilité d'un seul produit, la variabilité dans les lignes de produits est résolue avant que le produit ne soit délivré et installé dans son environnement d'exécution.

Atkinson [52] appelle la variabilité contenue dans un seul produit la variabilité de temps d'exécution et la variabilité contenue dans la ligne de produits la variabilité de temps de développement. Nous considérons dans ce document la variabilité de LdP, c.à.d. la variabilité résolue avant la délivrance et l'installation des produits.

a- Les dimensions de la variabilité :

La variabilité logicielle apparaît en deux dimensions : le temps et l'espace (cf.figure 2.16) [52 ,47]:
La dimension du temps concerne la variation dans le temps d'un seul produit logiciel. La Figure 2.16 montre l'évolution dans le temps des produits d'une version à une autre. La dimension de l'espace concerne la variation entre plusieurs produits de la même famille. Les même éléments logiciels sont utilisés dans plusieurs produits et la variation concerne principalement des variations de fonctionnalités, c.à.d. les produits diffèrent dans les fonctionnalités qu'ils supportent. Dans ce document nous considérons les deux dimensions espace, et temps de la variation.

b. Les points de variation :

Dans les lignes de produits, la variabilité est identifiée durant l'ingénierie de domaine et elle est introduite par ce qui est appelé les points de variation. Un point de variation peut être défini comme identifiant un ou plusieurs emplacements auxquels la variation peut se produire.
Un point de variation peut être vu comme un point de décision avec plusieurs choix possibles appelés variants. Le nœud Moteur dans le diagramme FODA dans la Figure 2.15 est un exemple de point de variation avec trois features variantes : Essence, Électrique et Diesel.
L'optionalité est un cas particulier d'un point de variation où le seul choix possible est la présence ou non de la feature. Le nœud Climatisation représente ce cas.

En plus, au niveau du modèle de features, les points de variation doivent apparaître à tous les niveaux d'abstraction (exigences, architectures, implantation, tests, etc.). Au niveau des architectures, les travaux ont étudié l'extension des langages de description d'architecture pour la spécification de la variabilité dans les architectures de LdP. Nous présentons les principaux travaux existants autour de la modélisation des architectures de LdP en UML dans la suite du document.

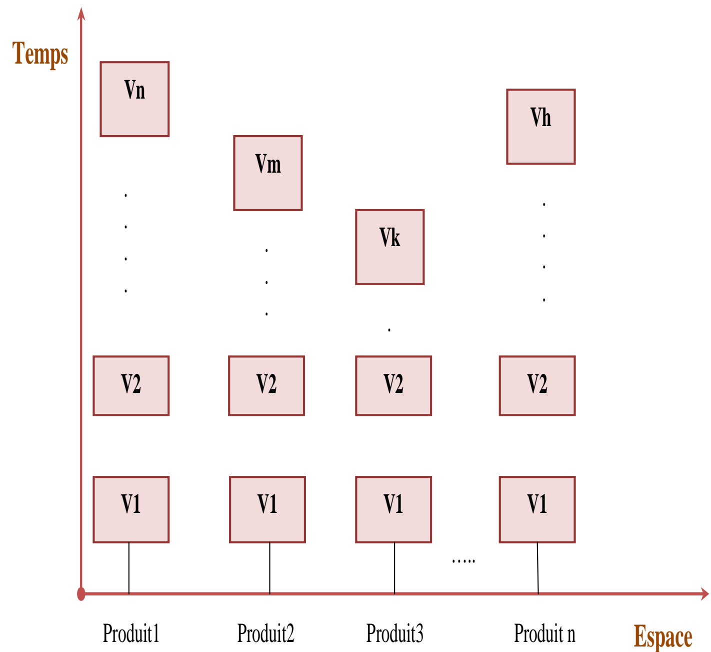


Fig 2.16– Les dimensions temps et espace dans la variabilité

<u>c. Gestion de La variabilité au niveau de l'implantation.</u>

Plusieurs techniques au niveau de l'implantation permettent la gestion de la variabilité. Parmi les approches utilisées pour la gestion de la variabilité au niveau de l'implantation, on peut retenir :

Les techniques de compilation. Elles permettent la dérivation d'un produit pendant la phase de compilation. La compilation conditionnelle et le chargement de bibliothèques sont des exemples de ces techniques. Elles sont utiles si la variabilité concerne les parties de code à inclure ou à exclure par rapport aux bibliothèques qu'elles utilisent.

Les techniques liées aux langages de programmation .Les langages à objets (LAO) ont apporté quelques techniques utiles pour implanter la variabilité. Citons l'abstraction à travers la notion d'héritage, du polymorphisme ; la surcharge et la liaison dynamique. Les points de variation peuvent être définis comme abstraits dans la LdP et redéfinis par chaque variant du produit d'une manière spécifique. Certains LAO permettent de définir des classes paramétrées, appelées classes templates. La variabilité peut être implantée en utilisant les classes templates lorsque les variantes ne diffèrent que sur un ensemble de types de paramètres (taille mémoire ; type d'un paramètre…).Les diagrammes de classes UML permettent de définir des classes Template et donc de gérer ce type de variabilité.

Les patrons de conception. Les patrons de conception [47] fournissent des solutions réutilisables pour certains types de problèmes. Dans [49] le patron de conception *fabrique abstraite* (Abstract Factory) est utilisé pour la réification des variants. La fabrique abstraite permet de définir une interface pour la création des produits concrets et propose un ensemble de patrons pour modéliser la variabilité [50].Ces patrons sont basés sur des types de dépendances entre un ensemble de propriétés.

Des approches de programmation. Des approches récentes de génie logiciel peuvent être utilisées pour implanter et gérer la variabilité dans les systèmes.

<u>La séparation des aspects</u> [51]. est une approche permettant de réduire la complexité des systèmes. Le principe est de décomposer le problème en un ensemble de composants

fonctionnels et d'aspects transversaux. Quelques travaux [52] proposent d'utiliser cette approche pour la gestion de la variabilité dans les LdP. L'idée est que pour implanter la variabilité, les aspects peuvent être vus comme des points de variation et chaque produit, membre de la LdP, est différencié par l'ensemble des aspects qu'il utilise. Le travail de Lopez-Herrejon [12] présente une étude de cas sur l'implantation d'une ligne de produit en AspectJ.

La programmation générative [53]. est une approche qui s'intéresse au développement des familles de systèmes, elle est basée sur la notion de « générateur ».
La variabilité dans les LdP peut être implantée en développant des générateurs comme des artefacts génériques, leur instanciation permet de générer l'implantation d'un produit. **Batory** [53] propose une approche générative pour la gestion des variants. Une approche pour gérer la variabilité basée sur l'évaluation partielle des programmes C.

d. Constat :

Les techniques présentées ci-dessus sont généralement liées aux langages de programmation, et relèguent l'activité de dérivation d'un produit à une activité de programmation. Même si ces travaux au niveau d'implantation se basent sur des technologies et des techniques qui ont connu une réussite remarquable, la maîtrise du code devient de plus en plus difficile à gérer avec la croissance exponentielle de la complexité des logiciels.
Pour notre étude, il est important de montrer et de spécifier la variabilité à un niveau plus abstrait, en particulier au niveau des modèles, en s'appuyant sur le standard UML. En effet, UML propose un ensemble de notations, sous forme de diagrammes, pour l'analyse et la conception orientée objet. Mais UML est dédié à la modélisation d'un seul système à la fois, et ne supporte donc pas explicitement la modélisation de la variabilité essentielle aux lignes de produits. Cependant, UML est un langage extensible, grâce à certains mécanismes tels que les *stéréotypes*, *les tagged values,* les *contraintes* (OMG, 2001)… .

2.5 UML pour la modélisation de la variabilité des LdP .

2.5.1- Introduction.

Un nombre important de travaux se sont intéressés aux diagrammes statiques d'UML et en particulier aux diagrammes de classes pour resoudre la variabilité dans les LdP. Ceci à notre avis peut être expliqué par deux raisons principales :

i– La majorité des travaux s'inspirent des modèles de features qui introduisent la variabilité au premier niveau d'abstraction (niveau de l'analyse du domaine). En effet, le modèle de features, comme celui de FODA, décrit une structure statique du domaine en termes de concepts et des relations entre eux. Czarnecki [11] justifie qu'il y a une grande similarité entre la notion de concept dans le modèle de features et celui des classes dans les langages orientés objet.
ii– La deuxième raison est plus liée à la nature des diagrammes de classes eux mêmes. En effet, les diagrammes statiques en particulier ceux de classes sont la vue principale dans la notation UML.

2.5.2 État de l'art sur la modélisation de LdP en UML .

Les auteurs dans leur majorité distinguent deux types de points de variation : les points de variation obligatoires (mandatory) et les points de variation optionnels. Les variants sont des cas d'utilisation stéréotypés dans la plupart des travaux par <<*variant*>>. Un point de variation peut définir plusieurs cas d'utilisation variants ; Ceci est modélisé par des liens

reliant les cas d'utilisation variants aux points de variation associés. Ces liens portent des cardinalités spécifiant le nombre minimum et maximum de variants à choisir dans un produit particulier.

Gomaa [57] s'est intéressé à la modélisation de lignes de produits en utilisant UML dans plusieurs travaux. Trois stéréotypes : <<kernel>>, <<optional>> et <<variant>> ont été introduits pour décrire les classes obligatoires, optionnelles et variantes respectivement dans les diagrammes de classes.

KobrA [58] est une méthode pour l'ingénierie des composants logiciels. L'objectif principal de KobrA est de combiner les approches par composant et ligne de produits pour le développement des systèmes. La structure statique des composants KobrA (Komponents) est décrite par des diagrammes de classes d'UML. La variabilité est introduite à l'aide d'un seul stéréotype : <<variant>>.
Ce stéréotype peut être appliqué aux paquetages, aux classes, aux attributs et aux méthodes de classes dans un diagramme de classes.

SPLIT-Daisy [54]. La méthode SPLIT (Software Product Line Integrated Technology) est développée dans un laboratoire commun entre Alcatel et Thales appelé LCAT. SPLIT-Daisy s'intéresse à la description des architectures de lignes de produits en utilisant UML .La variabilité est introduite par la notion de point de variation. Un point de variation est représenté comme une classe UML stéréotypée <<variation-point>> avec un ensemble d'attributs. Un point de variation est associé à un élément particulier de modèle statique (une classe ou un paquetage UML) et il est défini par un ensemble de classes ou de paquetages variants. Un point de variation est défini aussi par un mécanisme de variabilité ; il est représenté comme un stéréotype appliqué à une association reliant le point de variation et son élément. SPLIT introduit trois mécanismes de variabilité : **<<extend>>, <<insert>> et <<parametrize>>.**

VPM [55] propose une méthode appelée VPM (Variation Point Model) pour décrire les points de variation dans les lignes de produits. Un point de variation dans VPM est désigné par le symbole « • ». La vue statique de l'architecture de LdP est spécifiée comme un diagramme de classes UML étendu par un tagged value désignant un point de variation. Il est appliqué aux méthodes de classes affectées par ce point de variation. Les méthodes associées à un point de variation seront raffinées au moment de la dérivation des produits. Le raffinement est basé sur trois mécanismes de l'orienté objet : l'héritage, la paraméterisation et la délégation.

Flege [56] définit l'optionalité comme le seul mécanisme de variabilité. L'optionalité est introduite en utilisant deux stéréotypes : <<Optional>> pour désigner qu'un élément de modèle est optionnel. Le stéréotype <<HasVariability>> est associé à un élément non optionnel mais qui contient un ou plusieurs éléments optionnels (par exemple un paquetage UML qui contient une ou plusieurs classes optionnelles).

Clauß [57] a étudié la possibilité d'étendre le méta-modèle d'UML pour décrire la variabilité. Le premier travail de cet auteur se concentre en particulier sur l'introduction d'un ensemble de stéréotypes pour permettre de décrire un modèle de caractéristiques FODA comme un diagramme de classes d'UML. Chaque feature est spécifiée comme une classe UML et Clauß introduit trois stéréotypes, un pour chaque type de features dans FODA : <<mandatory>>,<<optional>>, <<alternative>>. **Clauß** propose des extensions pour spécifier

la variabilité au niveau de l'architecture statique de la LdP modélisée par un diagramme de classes. Ici trois stéréotypes appliqués aux classes d'UML ont été introduits : <<optional>> pour désigner une classe optionnelle, <<variationPoint>> et <<variant>> respectivement pour désigner un point de variation et ses sous-classes variantes.

Robak [58] présente une étude sur les travaux relatifs à la modélisation de la variabilité dans UML. Les auteurs ont étudié les travaux qui ont fait la correspondance entre les modèles de features et les modèles UML et ils proposent par la suite un stéréotype <<variable>> et un tagged value feature pour spécifier la variabilité dans les modèles de composants d'UML.
Un composant UML, d'après l'auteur, implante une ou plusieurs features et, pour garder la trace sur la *feature* implantée par le composant, le tagged value *feature* est associé au stéréotype <<variable>>. Il prend comme valeur le nom de la feature implantée par le composant.

Ziadi [47] a étendu UML pour permettre la spécification de la variabilité dans deux types de modèles d'UML : les diagrammes de classes et les diagrammes de séquences. Il Utilise des opérateurs de composition pour spécifier algébriquement les diagrammes de séquence d'UML2.1 sous forme d'expressions de référence.
Il définit ensuite deux types de contraintes dans les LdP : les contraintes génériques et les contraintes spécifiques et propose d'utiliser le langage OCL pour les formaliser. Sa troisième contribution concerne la dérivation de produits. Pour lui une approche de manipulation de LdP en UML doit aller au delà des buts descriptifs et doit proposer un support pour instancier la variabilité introduite pour dériver les modèles UML de produits membres de la LdP. Il a formalisé cette dérivation en utilisant la transformation de modèles. Pour réaliser la dérivation de comportements, il a étudié la problématique de la génération (appelée aussi synthèse) automatique de machines à états à partir de diagrammes de séquence.

Certaines approches, telles que [59,60,61,62] ont tenté de mettre au point un support de traitement systématique des variabilités dans tout le cycle de vie du logiciel. Cependant, Les plus récentes d'entre elles ne sont pas parvenues à supporter pleinement la représentation orthogonale des variabilités architecturales [63].
Certaines techniques de modélisation de la variabilité se fondent exclusivement sur l'utilisation des modèles de features tandis que d'autres [59,62] utilisent des mécanismes de modélisation avancés uniquement pour relier les modèles de features aux modèles architecturaux.

Table 2.2. Récapitulatif des principaux travaux sur la modélisation de la variabilité

Auteurs	Diagrammes Statiques	Diagrammes Dynamiques
Robak	X	X
Clauß	X	
Gomaa	X	X
Flege	X	X
KobrA	X	X
SPLIT Daisy	X	
VPM	X	
Ziadi	x	X
Pohl, K.,	X	X
Bachmann, F	X	X
Pure: variants	X	X
Loughran, N	X	

OVM [59] par exemple se limite à documenter la variabilité plutôt que d'exprimer sa composition avec des éléments architecturaux communs, par contre VML [77] en se concentrant sur quelques vues seulement oblige les intervenants sur les LdP à maitriser un langage supplémentaire pour pouvoir modéliser et gérer la variabilité.
La Table 2.2 récapitule les travaux sur la modélisation de la variabilité avec UML.

Deuxième partie

Contributions

Chapitre 3.

I

Un sous-modèle pour l'optionalité

&

la variabilité

des attributs et méthodes.

3.1 Introduction.

Les approches basées sur la stratégie ligne de produits et utilisant UML comme notation, souffrent d'une certaine ambigüité pour la perception et la représentation de la variabilité, notamment au niveau modélisation architecturale. Cette étape est souvent sous-estimée alors qu'elle permet une vision précoce des propriétés fonctionnelles du système. Dans ce chapitre, nous présentons une approche décrivant une partie de notre modèle qui étend l'aspect optionnel de la variabilité dans les classes et affine sa représentation au niveau des attributs et méthodes .Ces deux aspects de la variabilité pris en compte dans notre modèle fourniront un apport certain lors des étapes de conception et de développement des Produits. Nous appliquons ces aspects de la variabilité à un exemple significatif pour montrer leurs importances.

Les approches actuelles ne traitent pas suffisamment la variabilité au niveau des attributs et méthodes .Elles se limitent principalement à des modélisations au niveau des classes avec des extensions UML [68] [69]. Mêmes les approches qui traitent cet aspect utilisent les mêmes mécanismes d'expression pour la définir au niveau des classes ,des attributs et méthodes .A la différence de notre approche qui s'intéresse pour modéliser la variabilité, en plus des deux aspects traités par les approches précédentes, à son expression par rapport au comportement des attributs et méthodes sur les instances des classes . Elle étend également la variabilité optionnelle lui permettant de prendre deux possibilités : accessoire et stricte. La première décrit une variabilité optionnelle avec une prise en compte d'un point d'extension pour la feature optionnelle décrite .La seconde ne prévoit pas de point d'extension pour cette feature optionnelle.

Pour l'aspect variabilités des méthodes et attributs, notre modèle est basé sur le fait que la variabilité peut aussi concerner la population,: il cherche à lever l'ambigüité existante dans l'identification et la modélisation de la variabilité à travers un modèle logique basé essentiellement sur une séparation du comportement des méthodes et des attributs vis-à-vis des instances de la classe.

Pour la variabilité optionnelle, notre modèle propose une extension de ce type de variabilité à deux autres types qui complètent cet aspect : le type optionalité *accessoire* et le type optionalité *stricte.*

A ce stade de notre travail, notre modèle se limite à l'étude de la variabilité fonctionnelle dans un aspect statique à travers un seul modèle UML, le diagramme de classes.

3.2. Prise en compte de la variabilité dans notre modèle

Notre perception de la variabilité dans le diagramme de classes repose sur le modèle de la figure 3.1. La variabilité dans ce diagramme s'exprime dans les classes à travers la présence ou l'absence des classes variantes ou optionnelles dans les membres de la ldp. Elle est représentée par une classe variable qui peut être soit optionnelle soit paramétrée soit des alternatives de variantes. La variabilité optionnelle peut prendre deux formes :

- **accessoire** qui exprime l'extensibilité de la feature optionnelle décrite.
- **stricte** pour laquelle aucune extension de la feature n'est prévue dans tous les produits de la LdP.

La variabilité s'exprime également dans les attributs et les méthodes des classes variables à travers trois aspects :

i- La présence ou l'absence de l'attribut ou de la méthode dans la classe .Cet aspect s'exprime par l'optionalité (est présent ou absent) et l'obligation (doit être présent) de figurer dans la structure de la classe.

ii- Leurs comportements vis-à-vis des instances des classes auxquelles ils/elles appartiennent .Cet aspect s'exprime par le fait qu'un attribut décrive ou fasse référence à une instance de la classe ou à toute la collectivité ,et qu'une méthode traite une instance à ou doit traiter dans son exécution toute la collectivité avant de ne pouvoir fournir un résultat.

iii- Leurs préoccupations vis-à-vis des instances des classes auxquelles ils/elles appartiennent du fait que cet attribut ou que cette méthode soit commun(e) à toute la population ou concerne une ou quelques instances particulières de la classe qu'on doit spécifier.

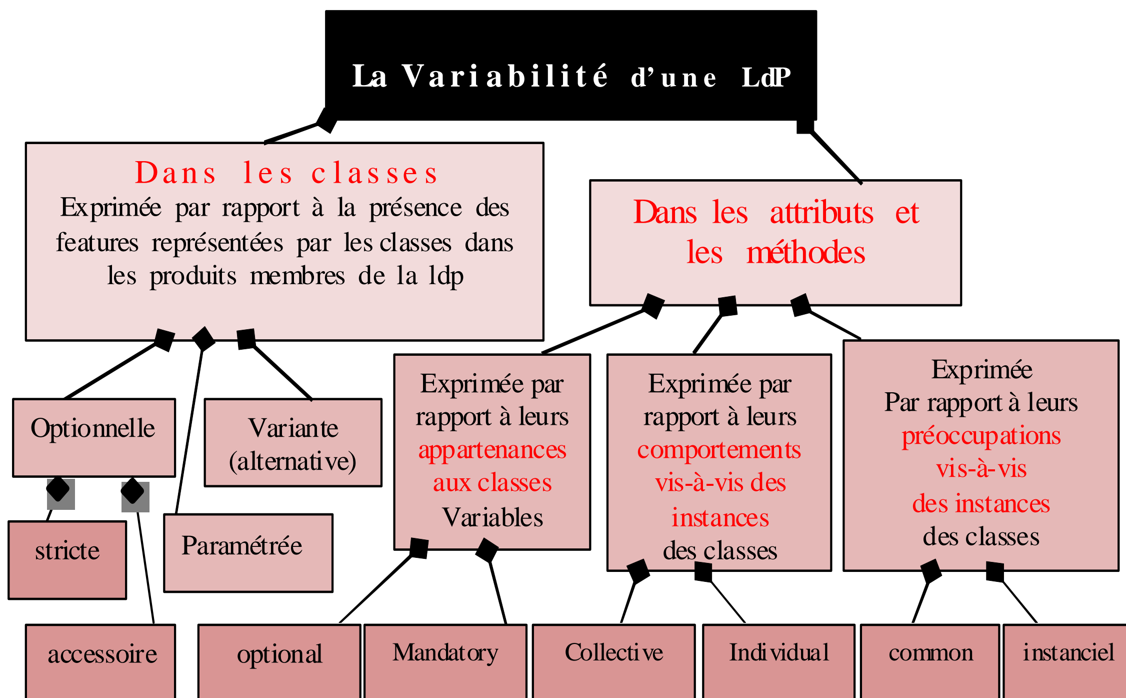


Fig. 3.1- Notre Modèle d'expression de la variabilité dans le diagramme de classes

Notre approche aborde graduellement la construction de cette partie du modèle par l'intronisation des concepts nécessaires à sa construction à travers les solutions apportées à un certain nombre d'insuffisances ayant trait à notre champ d'étude.

3-2-1- Première insuffisance : la classe dans UML ne fournit aucune information sur le nombre de ses instances.

Nous considérons un simple exemple du système d'information – **système d'allocation de ressources-** où un processus cherche à utiliser une ressource d'un système informatique dans lequel les processus sollicitent l'utilisation temporaire des ressources.

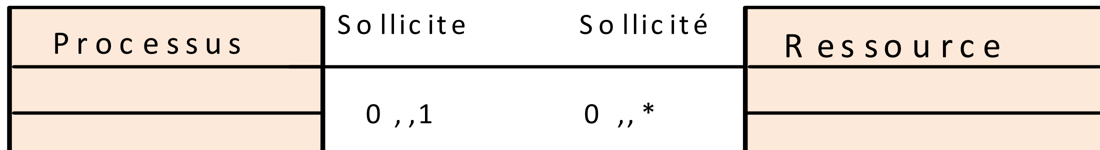


Fig 3.2 -Modèle logique -système d'allocation de ressources

La figure 3.2 illustre le modèle de classes d'un exemple réel. Nous supposons que le domaine du problème dispose de plusieurs processus, et de plusieurs ressources. La classe *processus* représente plusieurs processus, et la classe *ressource* plusieurs ressources. Le modèle logique ci-dessus ne mentionne pas explicitement si les classes ont en effet plusieurs processus, et plusieurs ressources. À ce stade, on peut supposer, que quelles que soient les classes qui existent dans le modèle logique, elles représentent plusieurs objets dans le domaine problème. Mais cette hypothèse ne peut souvent tenir. Pour essayer de montrer le contraire de l'hypothèse, ajoutons une autre information qui stipule que le système d'allocation des ressources dispose d'un composant système, qui n'est pas un processus et qui sollicite lui aussi les ressources. On l'appellera *systemthread*.

En résultat, nous aurons le modèle logique représenté dans la figure 3.3.Dans ce modèle logique, on remarque que la nouvelle classe *systemthread* ne représente qu'un seul objet. Cela contraste avec les deux autres classes qui représentent plusieurs objets. D'où la question : Quand est-ce que, une classe dans un modèle logique exprimé dans UML représente un ou plusieurs objets? En fait, il n'est pas expressément compris quand une classe représente un ou plusieurs objets dans les modèles logiques exprimés dans UML[70][71] .Toutefois, on peut faire valoir que l'on peut établir les nombres d'objets de chaque classe actuellement dans le domaine système/problème à l'aide du diagramme des occurrences et/ou par l'étude avancée des associations entre les classes dans le modèle logique [72]. Ce qui est très difficile à réaliser dans le cas d'un nombre important de classes.

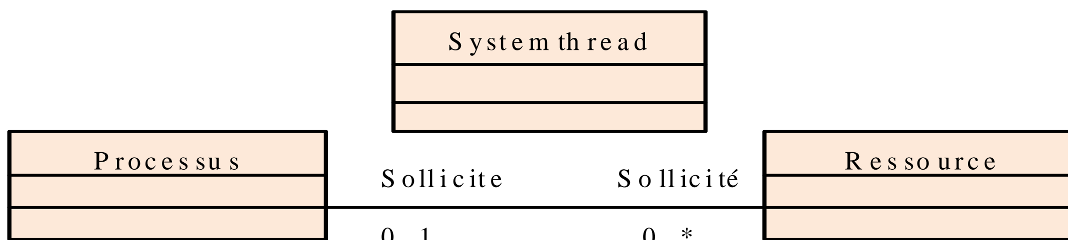


Fig.3.3 Sys. allocation de ressources avec une nouvelle classe systemthread

Solution proposée.

Notre solution à ce problème consiste à ajouter dans la structure de la classe un nouvel attribut que nous avons appelé *dimension* et qui contiendra l'information relative au nombre d'objets de la classe. Cet attribut bivalué nous renseignera à l'aide d'un cadrage par deux valeurs sur le nombre minimal et le nombre maximal d'instances de la classe.

Toute information relative aux associations existantes (cardinalités et autres) entre les classes sera maintenue car l'attribut *dimension* proposé ne peut s'y substituer.

3-2-2 *Seconde insuffisance* : *ambigüité dans les méthodes de la classe.*

Dans une classe UML, on ne sait pas exactement quelles sont les méthodes qui s'exécutent uniquement sur une instance de la classe et celles qui en s'exécutant traitent la population ou la collectivité. Pour expliquer cette problématique, prenons l'exemple de la classe *processus* issue de notre exemple précédent. Cette classe dispose des opérations suivantes : +solliciter (), +ajouter (), réserver (), +actualiser () , +rechercher(),+trier().

Remarquons que l'opération +ajouter () ne peut concerner qu'une seule instance de la classe à la fois .Elle correspondra à la création d'un seul et nouvel enregistrement à la fois. Par contre si on prend l'opération chercher (), cette dernière doit concerner lors de son exécution l'ensemble des instances de la classe avant de ne pouvoir donner un résultat. L'opération *actualiser ()* doit être toujours précédée par une recherche pour localiser l'instance concernée par la modification. L'opération *trier ()* doit toujours s'effectuer en parcourant toute la population des instances de la classe avant de donner un résultat du tri. De ce point de vue, les trois dernières méthodes sont différentes de la première. *Ajouter ()* agit sur une instance, alors que les trois autres méthodes agissent sur l'ensemble des instances de la classe.

Cet état de fait nous a motivé à chercher un autre formalisme dans la classe qui puisse prendre en charge cet aspect des méthodes.

Solution proposée:

Distinction des méthodes selon leurs comportements vis-à-vis des instances lors de leurs exécutions.

Cette séparation se traduit à travers l'utilisation de deux stéréotypes dans le compartiment réservé aux méthodes dans la structure de la classe :

i- « Individual » avec lequel seront décrites les méthodes qui traitent une seule instance.

ii- «Collective » avec lequel viendront toutes les méthodes qui traitent la population de la classe.

3-2-3 Troisième insuffisance : *ambiguïté dans les attributs de la classe.*

Dans une classe UML, On ne sait pas exactement quels sont les attributs qui concernent une seule instance de la classe et ceux qui concernent la population. Pour expliquer cette problématique, prenons l'exemple de la classe *processus* issue de notre exemple précédent Cette classe dispose des attributs suivants: (dimension , id_process , name, pripority, type.priority).

Remarquons que l'attribut *type.priority* qui indique le mode de calcul de la priorité pour les processus (LIFO, FIFO …..) est commun à toute la population des processus .Il en est de même pour l'attribut *dimension.*

Par contre les attributs *priority* qui indique la priorité (de 1 à 5 par exemple) du processus et *id.proces* qui indique l'identifiant du processus ne peuvent concerner qu'une seule instance de la classe. De plus lors de l'écriture du code source de l'application, le programmeur est parfois confronté à ce type de problèmes et se trouve contraint d'adapter la description des structures qu'il utilise en ajoutant des champs de type *set* ou *collection* [72] dans les classes pour avoir un code source adéquat qui puisse offrir après son implémentation un exécutable correspondant au modèle conceptuel des données. Cette situation nous a motivé à chercher un autre formalisme dans la structure de la classe qui puisse prendre en charge cet autre aspect des attributs.

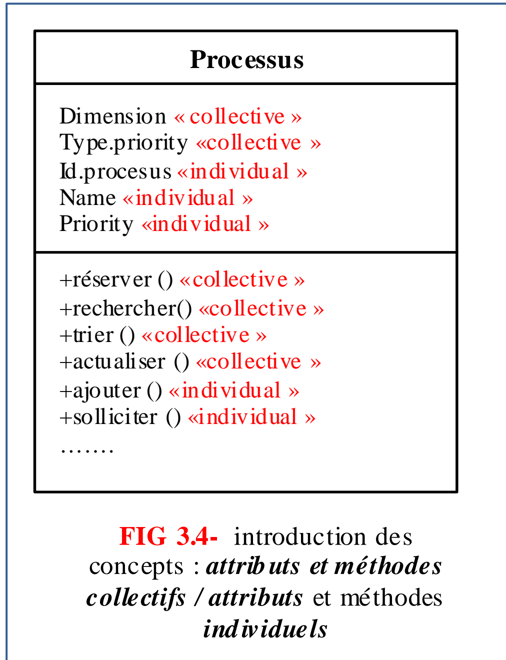


FIG 3.4- introduction des concepts : ***attributs et méthodes collectifs / attributs*** et méthodes ***individuels***

Solution proposée :

Distinction des attributs selon que leurs descriptions concerne une instance de la classe ou la collectivité. Cette séparation se traduit par l'utilisation de deux stéréotypes dans le compartiment réservé aux attributs:

i-« Individual» avec lequel seront décrits les attributs qui concernent une seule instance

ii- « collective » avec lequel seront décrits les attributs qui concernent la collectivité. Ainsi, la classe *processeur* sera modélisé selon le formalisme de la figure 3.4.

3-2-4 *Quatrième insuffisance* **:** *ambiguïté de la variabilité optionnelle.*

L'aspect optionnel de la variabilité des classes n'est pas exprimée pleinement à travers l'usage unique du stéréotype « optional » .cette variabilité nécessite davantage de raffinement pour pouvoir être exprimée complètement au niveau des classes.

Solution proposée.

Pour expliquer et solutionner à la fois cet aspect nous considérons un simple exemple de la LdP du système automobile .Dans ce système, les Feux antibrouillard expriment une variabilité optionnelle .Cependant, deux possibilités sont offertes pour les concepteurs :

i-soit ils prévoient ce que nous avons appelé un point d'extension (concrétisé par un emplacement d'installation) pour ces feux dans un emplacement du véhicule pour permettre son installation future au niveau de la LdP ou en dehors de la LdP par le client après acquisition du produit. Ce cas de figure représente pour notre modèle une *optionalité accessoire.* On la représente dans le diagramme de features par le symbole : «◎»

ii-soit ils ne prévoient pas du tout son extension dans les membres de la LdP. Ce cas de figure représente pour notre modèle ce que nous avons appelé une *optionalité stricte. Son* Symbole dans le diagramme de features est celui de l'optionalité : «◯ »

3-2-5 Solution Complète. La première étape de notre modèle se situe au niveau des classes variables (représentant des points de variation ou des options dans le diagramme de features).

Pour l'identification et la prise en compte de la variabilité au niveau des méthodes et des attributs, de ces deux types de classes, Nous avons introduit plusieurs nouveaux mécanismes qui seront utilisés dans chaque étape de raffinement du modèle.

***FIG 3.5**- structure d'un dictionnaire de la variabilité*

- Classe variable : nom-classe et type classe (variante / optionnelle-accessoire / optionnelle-stricte /paramétrée)

1- caractéristiques des attributs :

attributs	types				
	Individual	Collective	optional	mandatory	Instance concernée / common
a1			X		X
a2	X			X	Instane1
a3	X			X	Instance1 , instance2
a4 . . .		X			

2- caractéristiques des méthodes :

Méthodes	types				
	Individual	Collective	optional	mandatory	Instance concernéé common
m1	X				X
m2		X	X		
m3		X		X	
m4	X				Instance1
m5 . .			X		X

1ere étape de raffinement. Séparer Les méthodes et les attributs selon les types de variabilité apportés par le modèle proposé -figure 3.1- .Pour cela nous utiliserons ce que nous avons désigné par « *un dictionnaire de la variabilité* » –cf.figure 3.5- qui organise la description de la variabilité au niveau attributs et méthodes des classes variables .Une fois appliqué, Notre modèle logique prendra le formalisme de la figure 3.6.

FIG 3.6- variabilité dans la classe(entités prises en compte : la classe + attributs et méthodes)

Nom de la classe « variable »

-Dimension
-a1 « individual » « nom – instace 1 »
-*a2* « *Individual* » « nom – instance 2 »
-*a3* « *individual* » « nom – instance 3 »
-*a4* « *common* »
……….

+m1 «individual» « nom – instance 1 »
+m2 « collective » « nom – instance 2 »
+m3 « collective » « common »
+m4 « *individual* » «nom – instance 3 »
……

2eme étape de raffinement. *Modélisation des entités classes.*

Le modèle logique résultant de l'étape précédente peut être raffiné davantage en scindant la classe variable en une classe mère et autant de sous classes que d'instances concernées recensées dans le dictionnaire de la variabilité. Ces classes filles seront liées à la classe mère par une relation d'héritage. La classe mère comportera les Méthodes collectives

et attributs collectifs communs à toutes les classes filles.

Tab 3.1 .Mapping applicable aux classe

Type variation	stéréotype
Variation point	« variable »
Mandatory	/
Optional strict	« optional strict »
Optional accessory	« optionalaccessory »
exclusive variant	« excludes »
Inclusive variant	« requires »
Alternative variant	« variant »

Tableau 3.2 .Mapping applicable aux attributs et méthodes

Type variabilité	*Stéréotype correspondant*
Individual	*Individual*
Collective	*Collective*
Common	/
Mandatory	/
optional	*optional*

Le mapping du tableau 3.1 sera appliqué à ces classes. La classe mère renfermera les attributs et méthodes identifiant la commonalité quelque soient leurs types de variabilité.

***3eme étape de raffinement. Modélisation** des attributs et méthodes. **L**e* modèle logique obtenu dans l'étape précédente peut être raffiné davantage en appliquant le Mapping spécifié dans le tableau 3.2 sur les attributs et méthodes dans le diagramme de classes .Le résultat sera un diagramme de classes avec les stéréotypes appliqués aux classes, attributs et méthodes pour chaque classe variable du diagramme. Par exemple, le modèle final pourrait être celui de la figure 3.7 si notre classe mère représente une feature avec deux classes filles (instance1 et instance2) et une classe fille obligatoire ou mandatory (instance3) .

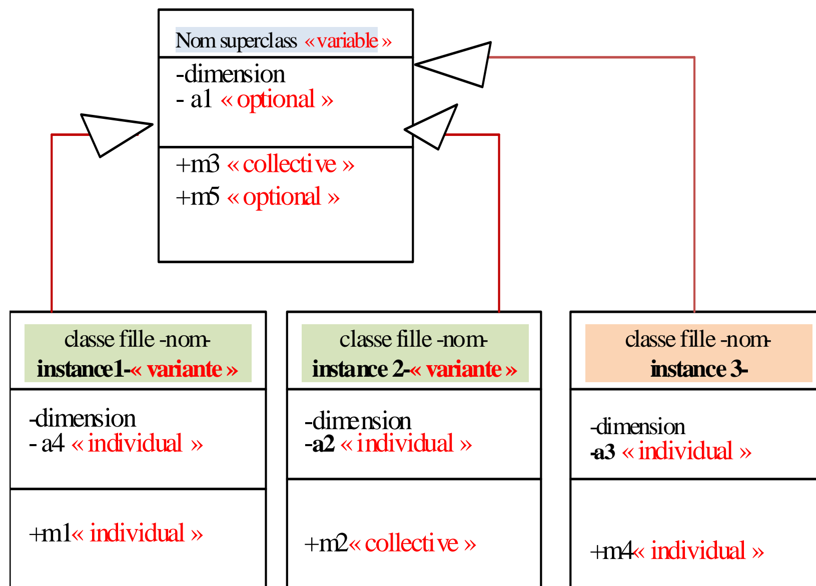


Fig.3.7- Modèle logique final

3.3. Discussions .

a. L'attribut dimension.

*L'*attribut dimension permet de connaitre pour chaque classe le nombre maximal et minimal de ses instances. Ceci permet d'avoir une approximation sur le nombre maximal de produits pouvant être dérivés dans la LdP .Il permet aussi de renforcer le contrôle au niveau des points de variation lors de la dérivation d'un produit en veillant à ne pas dépasser ou descendre en dessous de la dimension de chaque classe variable

et en vérifiant par exemple qu'il est impossible que des classes variables de dimension 1 ,1 puisse avoir des classes filles.

Les classes optionnelles sont les seules classes qui ont une dimension bivaluée à 0 ,1. La dimension d'une classe variable remplace les cardinalités [min, max] pour le cas de la multiplicité des features dans les modèles de features basés sur la cardinalité .Cet aspect des features est essentiel pour la variation dans les LdP.

b. Les stéréotypes introduits.

Les stéréotypes introduits pour les attributs et les méthodes peuvent être très utiles pour l'évolution de la LdP et la suppression de la variabilité négative .Exemples : le concepteur peut opter pour la création d'une autre classe variable de type « *optional* » à partir des attributs et méthodes d'une classe variable qui contient plusieurs (deux ou plus) attributs et méthodes de type « *optional* » et changer ainsi l'architecture de la LdP. Le concepteur peut opter pour la création d'attributs ou de méthodes optionel(le)s dans les classes variantes à partir des attributs et des méthodes d'une classe variable de type « individual » qui concernent un nombre limité d'instances , deux instances (une à la fois ou les deux en même temps) ou plus (exemple : classes concernées :instance1,instance2 dans la figure 3.5) .

Pour une meilleure modélisation de la variabilité dans la LdP , Le concepteur doit veiller tant que cela est possible à avoir dans les classes variantes des attributs et méthodes de type Individual .

c. L'extension de la modélisation relative à la variabilité optionnelle.

Exprimer l'optionalité d'une feature par sa présence ou non dans les membres de la LdP est insuffisant que ce soit pour les clients des produits de la LdP ou même pour ses concepteurs. En étendant la notion d'optionalité au fait de savoir si un point d'extension est prévu pour cette feature permet aux clients de pouvoir étendre le produit acquis à partir du point d'extension (exemple : dans une LdP de cartes mère pour PC . la feature carte fax-modem est optionnelle . cette variabilité est insuffisante . Le client a besoin de savoir si un slot d'extension existe sur la carte mère qui permet de connecter cette feature séparément (en dehors de la LdP) ou si ce dispositif n'existe pas sur la carte mère .Il en est de même pour le concepteur pour qui prévoir ce dispositif d'extension en dehors du fait qu'il permet de distinguer les membres de la LdP d'autres produits concurrents sur le marché , lui permet également de mieux maitriser l'évolution de la LdP (prévoir des produits futurs de la LdP dans lesquels ce dispositif deviendrait obligatoire par exemple ….).

3.4- Intégration à *UML* des mécanismes d'extension proposés.

Cette section propose un ensemble de mécanismes pour l'intégration à UML des stéréotypes « accessory » , « strict » , « individual » et « collective » . Mais avant cela nous allons décrire brièvement l'architecture à quatre couches d'UML, les paquetages logiques permettant d'organiser les différents méta-modèles du langage et le formalisme de spécification du méta modèle d'UML.

3.4.1. L'architecture d'UML.

UML est caractérisée par une sémantique à base de méta-modélisation .Depuis ses premières versions, le standard UML est caractérisé par sa sémantique définie par une approche de méta-modélisation.
Un méta-modèle est la définition des constructions et des règles de création des modèles [79].

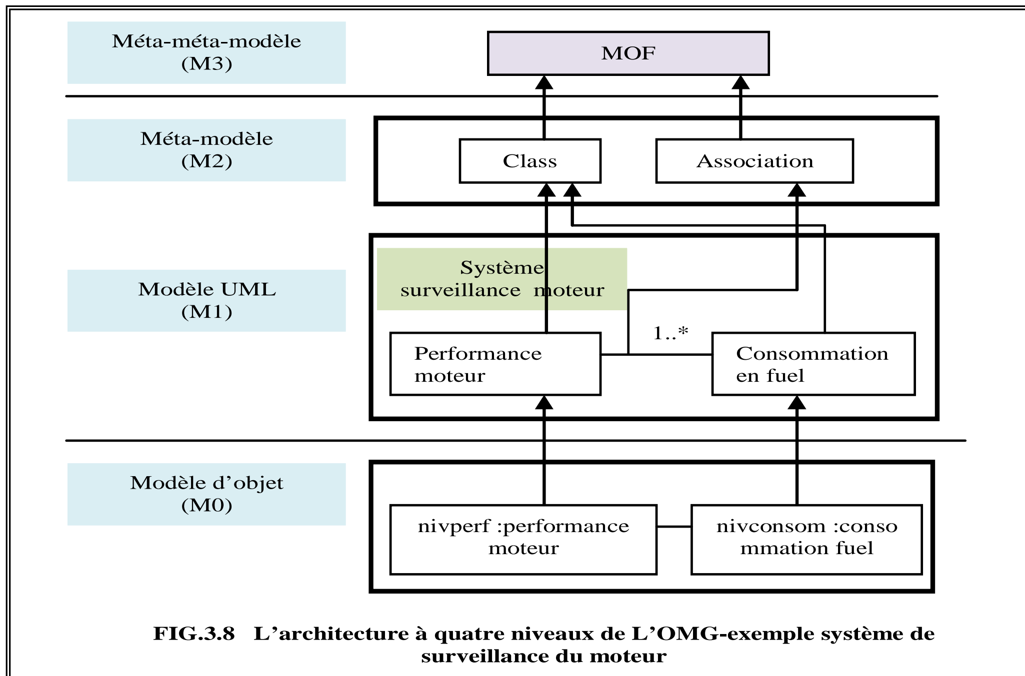


FIG.3.8 L'architecture à quatre niveaux de L'OMG-exemple système de surveillance du moteur

Le méta-modèle d'UML définit donc la structure que doit respecter tout modèle UML. Le méta-modèle d'UML1.x est défini dans un seul document, cependant le standard UML2.1 est divisé en deux documents : UML2.1 Infrastructure [17] et UML2.1 Superstructure [16].
UML Infrastructure décrit les constructions fondamentales utilisées pour la définition

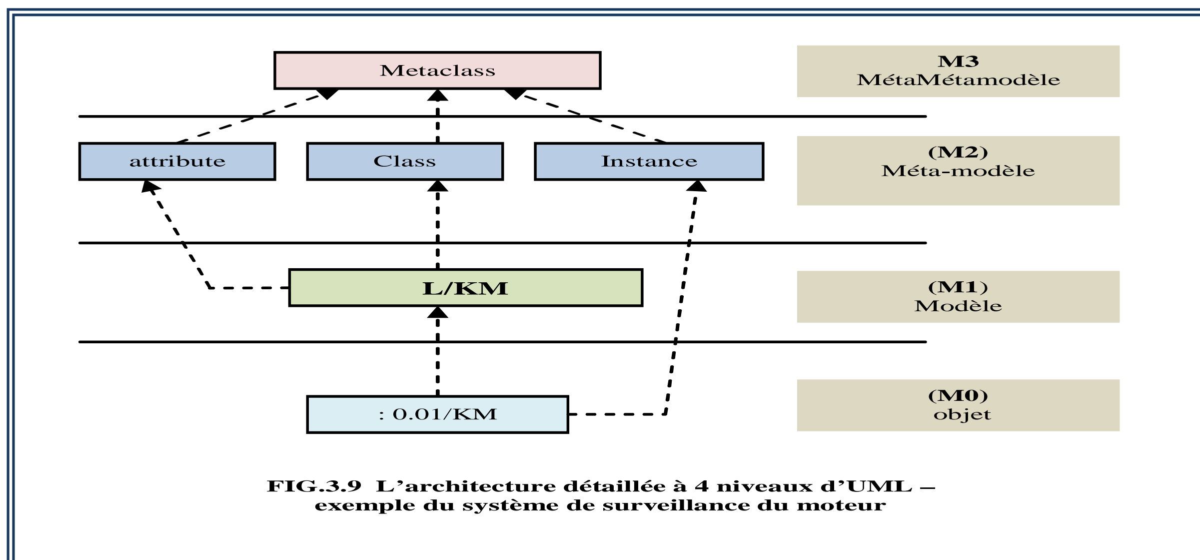


FIG.3.9 L'architecture détaillée à 4 niveaux d'UML – exemple du système de surveillance du moteur

d'UML2.1 sous forme d'une librairie d'infrastructure (Infrastructure Library). UML Superstructure réutilise et raffine la librairie d'infrastructure et définit le méta-modèle proprement dit, vu par les utilisateurs.

L'approche de méta-modélisation adoptée par l'OMG (Object Groupe Management) est connue comme une hiérarchie à quatre niveaux [15] (cf. Figure 3.8) et donc on peut considérer que L'architecture d'UML est basée sur une structure à quatre niveaux d'abstraction: M0 (objets ou données utilisateurs), M1 (modèle), M2 (méta-modèle) et M3 (méta-méta-modèle) (OMG, 2004). Chaque niveau Mi est vu comme "instance" du niveau supérieur Mi+1 (cf. figure 3.9). Certains chercheurs parlent plutôt de conformité entre les niveaux Mi et Mi+1 .

a. Le Niveau méta-méta-modèle (M3).

M3 est le niveau méta-méta-modèle, qui définit le langage de spécification du méta-modèle. Le MOF (Meta Object Facility) [80]) est un exemple d'un méta-méta-modèle.Cette couche définit un langage de haut niveau pour spécifier les méta-modèles. Elle est une instance d'elle même. Le méta-modèle d'UML (niveau M2) en est une instance particulière.

b. Le Niveau méta-modèle (M2).

M2 est le niveau méta-modèle .C'est la couche qui définit les éléments syntaxiques et sémantiques des modèles du niveau M1. Les concepts UML, "Class", "Attribute", "Instance", etc., sont définis à ce niveau. Le méta-modèle d'UML se situe à ce niveau et il est spécifié en utilisant le MOF, c.à.d. les concepts du méta-modèle d'UML sont des instances des concepts de MOF. La Figure 3.8 montre deux méta-classes du méta-modèle UML : Class et Association.

c. Le Niveau modèle (M1).

M1 correspond au niveau des modèles UML des utilisateurs. C'est la couche qui héberge les modèles UML développés pour spécifier un domaine d'application donné. Elle décrit la structure de la couche M0.Les concepts d'un modèle UML sont des instances des concepts du méta-modèle UML. La Figure 3.8 montre un extrait de diagramme de classes pour un système de surveillance du moteur contenant deux classes « *performance moteur* » et « *Consommation fuel* » liées par une association UML. Les deux classes sont des instances de la méta-classe « *Class* » et le lien est une instance de la méta-classe « *Association* » du méta-modèle UML.

3.4.2 Les contraintes OCL

Le méta-modèle UML spécifie la structure que doit respecter tout modèle UML. En d'autre terme, il spécifie des contraintes structurelles sur ces modèles. UML inclut le langage OCL (Object Constraints Language) [86] comme un moyen supplémentaire pour renforcer les contraintes structurelles des modèles UML en ajoutant des invariants sur les classes du méta-modèle d'UML. Les contraintes OCL au niveau méta-modèle représentent donc des règles de conformité des modèles UML, elles sont exprimées au niveau méta-modèle et elles sont évaluées sur tous les éléments de modèles UML, instances des éléments du méta-modèle d'UML. Les contraintes OCL sont étendues pour être utilisées aussi pour exprimer des propriétés sur un modèle UML (niveau M1). Elles sont utilisées pour spécifier des invariants, des pré et des post conditions sur les opérations et des gardes sur les transitions dans les machines à états UML. Les contraintes exprimées au niveau M1 sont évaluées et vérifiées sur les modèles d'objets (niveau M0).

3.4.3 Les diagrammes.

La notation UML est décrite sous forme d'un ensemble de diagrammes. La première génération d'UML (UML1.x), définit neuf diagrammes pour la documentation et la spécification des logiciels.
Dans UML2.1 Superstructure [16], quatre nouveaux diagrammes ont été ajoutés : il s'agit des diagrammes de structure composite (Composite structure diagrams), les diagrammes de paquetages (Packages diagrams), les diagrammes de vue d'ensemble d'interaction (Interaction overview diagrams) et les diagrammes de synchronisation (Timing diagrams).

3.4.4 Les paquetages logiques du méta-modèle d'UML .

La complexité du méta modèle d'UML (niveau M2) est gérée grâce à une organisation en paquetages logiques.

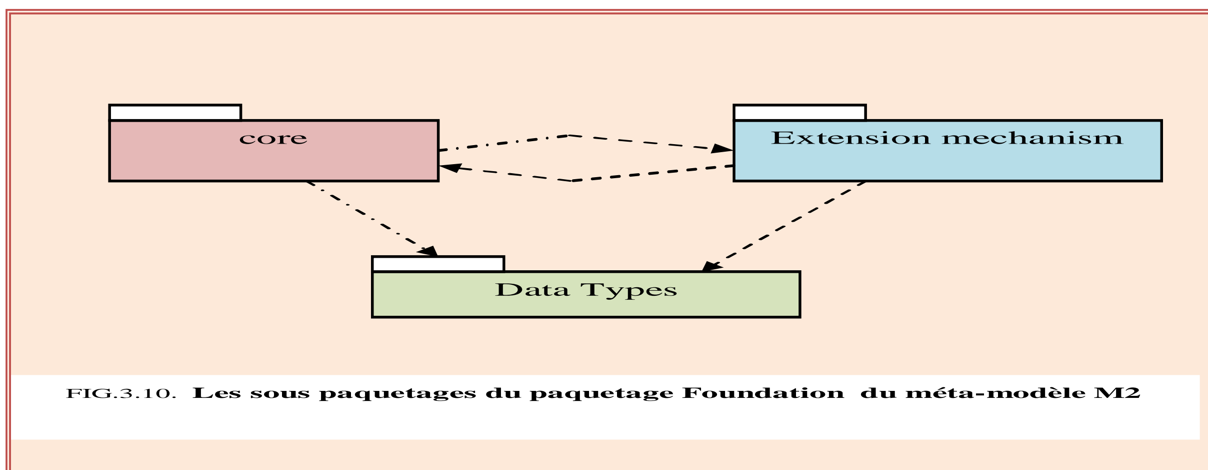


FIG.3.10. **Les sous paquetages du paquetage Foundation du méta-modèle M2**

Ces paquetages regroupent des méta classes qui représentent une forte cohésion interne et un faible couplage avec les méta-classes d'autres paquetages. Cette couche comprend, entre autres, le paquetage « Foundation ».

Ce paquetage est l'infrastructure du langage UML qui définit la structure statique des modèles du niveau M1. Il est composé de trois sous-paquetages : « Core », « Extension Mechanisms » et « Data Types » (cf. figure 3.10) :

Le paquetage *Core* . Définit les concepts de base pour la construction d'un modèle objet. On y distingue deux types de constructions, abstraites et concrètes. Les constructions abstraites ne sont pas instanciables et sont généralement employées pour réifier les constructions de base et organiser le méta-modèle d'UML.

Les constructions abstraites définies dans le paquetage « *Core* » incluent « *ModelElement* », « *Relationship* » et « *Classifier* ».

Un élément quelconque du modèle est une sous-classe directe ou indirecte de *ModelElement*, héritant de lui l'attribut « *name* », qui confère un nom à n'importe quel élément existant dans UML.

Les constructions concrètes sont, par contre, instanciables. Elles incluent *« Class », « Attribute », « Operation »* et *« Association »*.

Le paquetage *Extension Mechanisms.* Offre des mécanismes permettant d'ajouter de nouveaux éléments dans UML sans devoir changer sa structure de manière substantielle.

Le paquetage *Data Types* définit les types de données de base utilisés dans la spécification des autres paquetages.

Une extension substantielle d'UML affecte le paquetage « Core ». C'est la raison pour laquelle, dans le cadre de notre travail, les concepts relevant des stéréotypes : « individual », « collective », « accessory » et « strict » seront incorporés dans le paquetage extension mechanisms.

3.4.5 L'extensibilité à UML.

Les mécanismes que nous avons introduits pour la prise en compte de la variabilité dans notre modèle sont quatre stéréotypes : « individual », « collective » pour les attributs et « optional accessory » et « optional strict » pour l'optionalité notamment au niveau des classes UML
Le document UML Infrastructure [17] propose une présentation détaillée de ce type de mécanisme. Les stéréotypes dans le standard UML2.1 font partie de la librairie d'infrastructure [17]. La Section Extension Mechanisms dans UML1.x a été remplacée par un paquetage appelé Profils dans UML2.1 Infrastructure.
Nous décrivons dans ce qui suit le concept de stéréotype dans ce paquetage et qui est utilisé par notre modèle.

Le stéréotype est le mécanisme de base pour l'extension d'UML. Il définit la manière dont une méta-classe particulière du méta-modèle d'UML peut être étendue pour permettre l'utilisation d'une terminologie ou d'une notation spécifique à un domaine ou une plate-forme particulière.

Un stéréotype est lié par un lien d'extension [17] à une méta-classe particulière du méta-modèle. Les tagged values introduits dans UML1.x sont considérés dans UML2.1 comme des propriétés des stéréotypes. Quand un stéréotype est appliqué à un élément du modèle (instance de la méta-classe sur laquelle le stéréotype est défini), cet élément sera noté par <<label stéréotype>> et les valeurs de ses propriétés si elles existent, seront considérées comme des tagged values associés à cet élément. Un tagged value a un nom et un type (cf figure 3.11) -

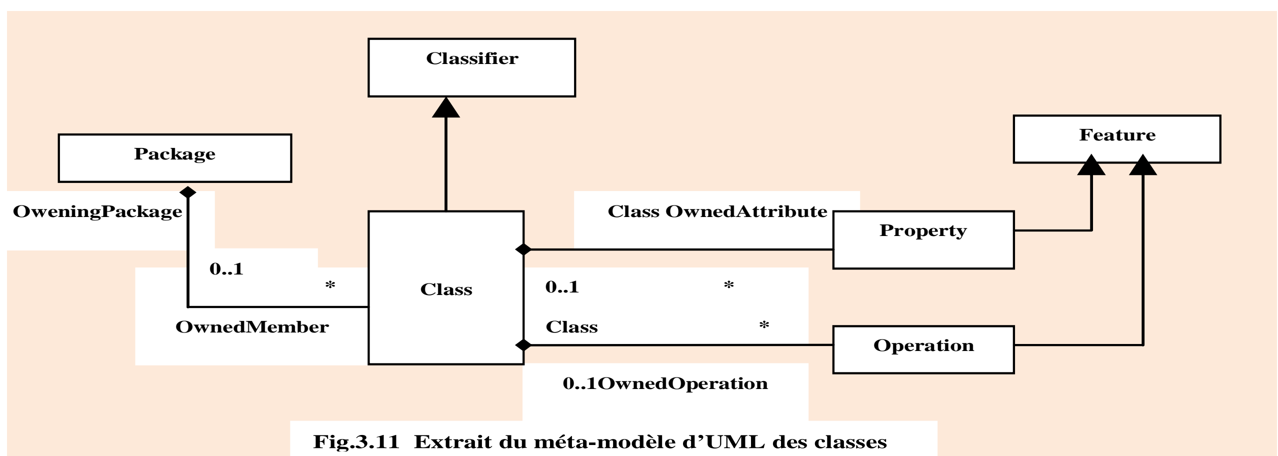


Fig.3.11 Extrait du méta-modèle d'UML des classes

3.4.5.1 Intégration des stéréotypes « optional strict » et « optional accessory ».

Les stéréotypes introduits ci-dessus pour la modélisation de la variabilité dans les diagrammes de classes sont définis comme des extensions sur la partie du méta-modèle d'UML2.1 spécifiant les classes [16].

La Figure 3.11 montre les méta-classes de cette partie concernées par nos extensions. La méta-classe « *Class* » spécifie une classe UML ; elle peut avoir un ensemble d'attributs et un ensemble d'opérations.
Un attribut d'une classe est spécifié par la méta-classe « *Property* » et une opération par la méta-classe « *Operation* ».
Ces deux dernières méta-classes héritent de « *Feature* ».

La méta-classe « *Package* » spécifie un paquetage UML ; un paquetage peut contenir un ensemble de classes. La méta-classe « *Class* » hérite de « *Classifier* ».

Les diagrammes de classes d'UML sont utilisés pour décrire la structure statique d'un système en termes de classes et leurs associations. Pour modéliser l'aspect statique d'une LdP, la variabilité doit être explicitement spécifiée et modélisée dans un diagramme de classes d'une LdP.
Dans la suite de cette section nous présentons deux mécanismes de variabilité dans les diagrammes de classes ainsi que leurs définitions en termes d'extensions au méta-modèle d'UML.

L'optionalité peut concerner les paquetages, les classes, les attributs et les opérations. Ainsi les stéréotypes <<optional accessory>> et << optional strict>> étendent donc les méta-classes « Classifier », « Package », et « Feature » (cf.figure 3.12). Ils remplacent le stéréotype « optional » pour les paquetages et les classes (le stéréotype « optional » utilisé dans la majorité des approches et le stéréotype « optional strict » jouent le même role sémantique. Notre modèle supporte l'usage de « optional » pour designer « optional strict »).

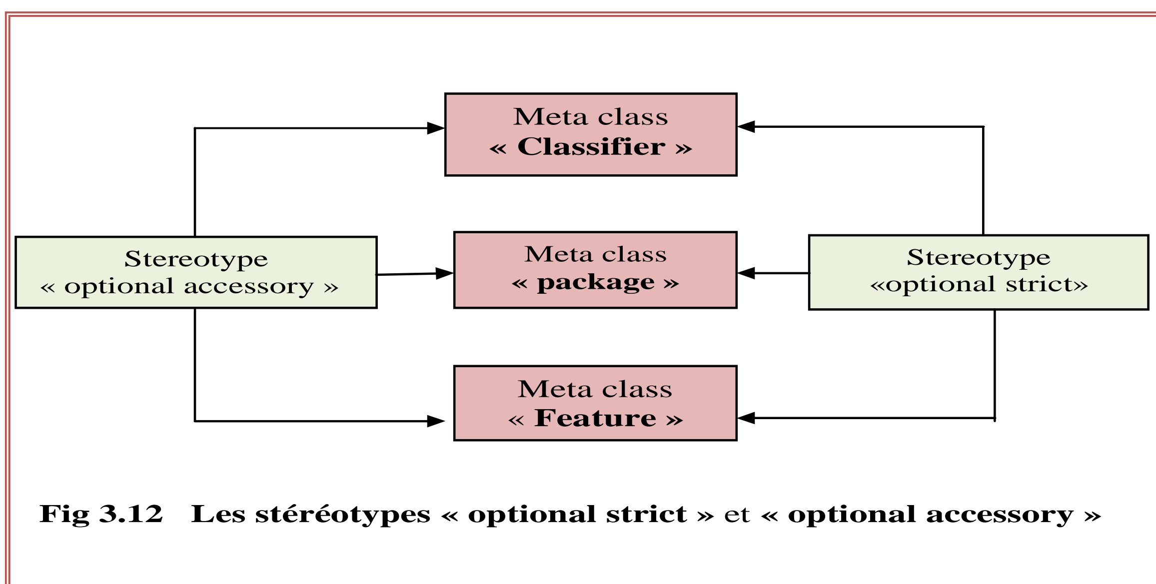


Fig 3.12 Les stéréotypes « optional strict » et « optional accessory »

3.4.5.2 Intégration des stéréotypes <<collective>> ,<<Individual>> et <<Optional>>.

Les trois stéréotypes <<collective>> ,<<Individual>> et <<Optional>> qui améliorent la représentation de la variabilité dans les attributs et méthodes , étendent ces derniers dans le diagramme de classes à travers les méta classes :

- « property » pour les attributs et
- « opération » pour les méthodes.

En étendant « property » et « operation » , Ils étendent également la méta-classe « *Class* ». On peut les modéliser dans le sous méta-modèle UML des classes de la façon suivante - cf. figure 3.13 - :

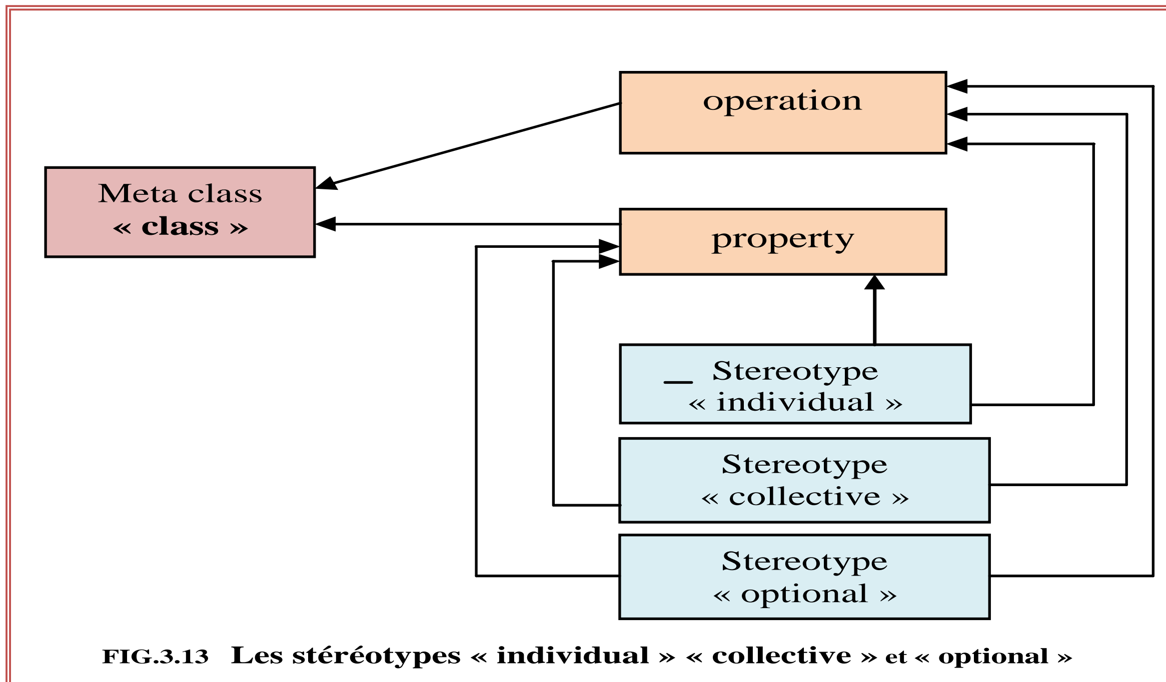


FIG.3.13 Les stéréotypes « individual » « collective » et « optional »

3.5. Conclusion.

Dans ce chapitre, nous avons proposé des mécanismes de spécification de la variabilité à un niveau plus abstrait qui est celui des modèles en s'appuyant sur le standard de la notation orientée objet UML. En se concentrant sur le comportement des attributs et des méthodes sur les instances des classes, en raffinant davantage l'aspect optionnel de la variabilité et en complétant le niveau d'informations de la classe par un nouveau concept qui renseigne sur la population de la classe , nous avons proposé un modèle qui représente mieux la variabilité dans les lignes de produits à travers des extensions à UML sous forme de stéréotypes . En procédant ainsi, nous avons montré que notre modèle tout en complétant les aspects déjà pris en charge de la variabilité lui confère un plus pour sa gestion.

Deuxième partie
Contributions

Chapitre 4

Un sous-modèle d'expression de la variabilité par les roles.

4.1 Introduction.

La variabilité logicielle apparaît en deux dimensions : le temps et l'espace (cf. Figure 2.14 chapitre 2) [52 ,47]. La dimension du temps concerne la variation dans le temps d'un seul produit logiciel. L'évolution dans le temps des produits montre l'évolution d'une version à une autre. La dimension de l'espace concerne la variation entre plusieurs produits de la même famille. Les mêmes éléments logiciels sont utilisés dans plusieurs produits et la variation concerne principalement des variations de fonctionnalités c.à.d. les produits diffèrent dans les fonctionnalités qu'ils supportent.
Dans cette partie du document nous considérons les deux dimensions espace, et temps de la variation.
Cependant, l'évolution de la variabilité à travers les variants est spécifiée d'une manière ambigüe dans le diagramme de classes. A ce titre, l'évolution de la variabilité est laissée à la charge du concepteur et de l'architecte système .Ces intervenants sur les LdP utilisent chacun les techniques et les méthodes qu'ils jugent appropriées pour contrôler cette évolution au niveau du diagramme de classes.

Pour apporter une solution à cette problématique, notre modèle préconise l'expression de la variabilité à travers les roles.

Nous proposons l'expression de la variabilité au niveau des variants relevant des points variables à l'aide d'une nouvelle association désignée « *role* » .Dans le modèle, les points de variation sont représentés sous forme de super-classes et leurs variants sous forme de *sous-classes roles*. A chaque super-classe correspond une machine à état-transitions qui représente l'évolution de la variabilité à travers les variant-roles de ce point de variation. Ces classes roles sous certaines contraintes peuvent aussi évoluer pour jouer d'autres rôles. Cet autre aspect du rôle est également traité dans notre modèle .L'association « *role* » permet également aux classes variant-roles d'hériter leur super-classes .Pour compléter le niveau d'information de cette structure , un listing des contraintes d'acquisition des rôles par les instances de la super classe ainsi que des contraintes d'évolution des rôles peut être joint à ce modèle . Ces deux types de contraintes peuvent être exprimés sous la forme de prédicats dans le langage formel OCL.

Dans ce qui suit nous allons examiner la possibilité d'exprimer la variabilité à travers l'utilisation du concept de role revu et adapté au problème des LdP .Nous avons proposé pour cela un diagramme des roles qui représente les variants sous forme de *classes role* associées à chaque classe variable (représentant un point de variation) et permettant de montrer à travers des états transitions les évolutions possibles de ces roles ainsi que leurs interdépendances.

4-2- Eléments de notre Modèle :

4.2.1. Règles du modèle.

1- Le role est représenté par une classe.

2- L'association *role* représentée par le symbole << O >> dans le diagramme de classes relie une classe variable appelée classe d'instances (représentant un point de variation), à des sous classes variantes appelées classes de roles (variant-roles), qui décrivent les roles dynamiques de la classe d'instances et permet donc d'identifier la variabilité au niveau des points de variation par l'usage des roles.

3- La classe de roles décrit les rôles dynamiques pour la classe variable avec laquelle elle est associée. Cette classe est une classe exprimant la variabilité à travers soit une classe optionnelle soit une classe de variation.

4- Les instances de la classe role sont des roles.

5- Les instances de la classe variation expriment la variabilité en jouant des rôles si certaines conditions sont réunies .Ces conditions sont appelées contraintes d'acquisition de roles (CTR i) qui lui sont alors associées. Ces contraintes peuvent être définies en tant que prédicats de transition et décrits dans un listing des contraintes en utilisant le langage déclaratif formel OCL (*Object Constraint Language*) d'UML [10].

6- Les classes de rôles sont utilisées pour modéliser la variabilité de la superclasse variable. Elles représentent les sous-classes (classes filles) variantes de la superclasse variable (classe mère). Elles héritent la super-classe représentant le point de variation.

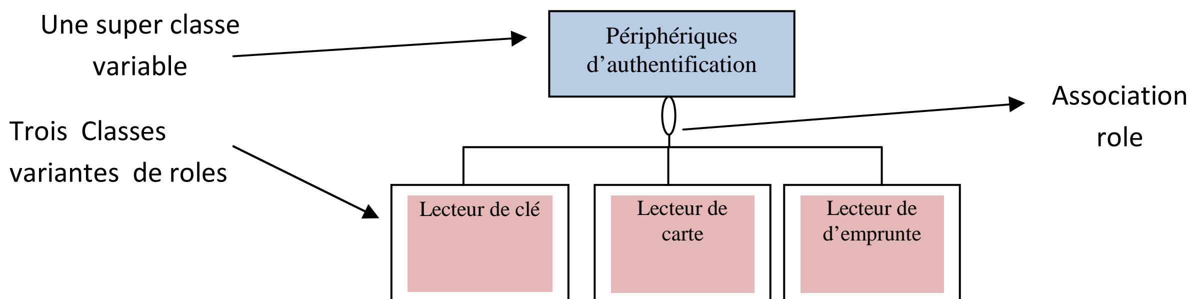


Figure 4.1. Un exemple de notre modèle de role .

4.2.2 Exemple d'expression de la variabilité dans une classe variable :

Soit la classe *périphérique-d'authentification* d'une maison intelligente .Cette classe peut jouer les roles suivants : lecteur de clé, lecteur de carte ou lecteur d'emprunte*s*.

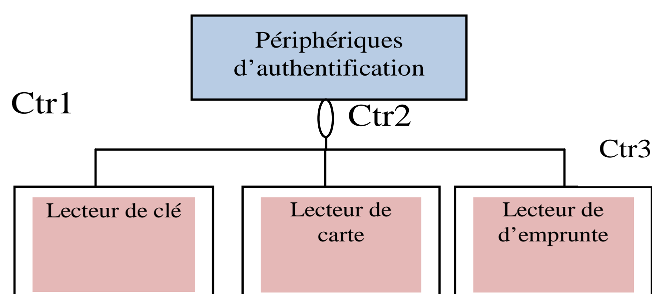
La classe *périphérique-d'authentification* peut jouer dans la plupart des cas l'un des 3 roles seulement.

Dans notre modèle, il lui est permis de jouer plusieurs roles en même temps càd que la maison intelligente sous certaines contraintes peut être dotée de deux types de périphériques d'authentification en même temps : *le lecteur de carte* et *le lecteur de clé* par exemple.

Ce sont deux roles différents. Ils sont représentés chacun par une sous classe liée à la classe mère dont elles héritent les propriétés communes définies dans le chapitre précédent relatif à la variabilité des attributs et méthodes.

Elles renferment les propriétés variantes et propres chacune à une instance particulière de la super classe. La super classe peut acquérir des roles

Fig 4.2 –Modèle de role amélioré -prise en compte des contraintes –



selon certaines conditions. En introduisant ces conditions sous forme de labels CTRi, le modèle de la figure 4.1 sera amélioré pour donner celui de la figure 4.2.

CTR1, CTR2 et CTR3 sont les conditions selon lesquelles la maison intelligente sera dotée de tel ou tel périphérique d'authentification.

4.2.3 Prise en compte de l'évolution des variant-roles

A ce stade ; le diagramme de classes ne peut représenter plusieurs situations relatives aux points de variation dont les plus importantes sont :

1-Une classe dont les instances jouent plusieurs roles différents en même temps. Par exemple lorsque la super-classe « *périphérique d'authentification* » utilise deux types d'authentification à la fois.

2-Ce modèle ne peut pas modéliser la dépendance d'un role vis-à-vis d'un autre.

3-Ce modèle ne peut décrire explicitement les conditions d'acquisition des roles.

Essayer de représenter ces aspects des roles dans le diagramme de classes donnera comme résultat un diagramme surchargé et difficile à exploiter. Cet état de fait nous a motivés pour proposer un autre diagramme qui prend en charge cet aspect de la variabilité à travers les roles et leurs évolutions. Ces éléments sont les suivants :

a- Nous avons proposé une machine d'états-transitions pour représenter l'évolution des variant-roles des classes variables et qui montre l'interaction inter-roles d'une même classe variable où :

i- Les états représentent les roles ;

ii- Les transitions libellées CTRi indiquent les contraintes auxquelles sont soumises les instances de la classe variable pour pouvoir évoluer.

iii- Un état initial et un état final sont ajoutés pour parachever le formalisme de ce diagramme.

b- Les contraintes d'évolution inter-roles CTRi :

Les contraintes d'évolution peuvent être décrites en utilisant le langage déclaratif formel OCL (*Object Constraint Language*) d'UML [10].

c- Les contraintes d'intégrité (de dépendance) inter-roles :

Ce type de contrainte est obtenu par une évolution unidirectionnelle d'un role vers un autre. Les flèches unidirectionnelles montrent les dépendances ou contraintes d'intégrité inter-roles dans le diagramme.

A la différence des autres roles, un role concerné par ce type de contraintes sera supprimé -aura cessé d'exister- si le role source est supprimé. Son existence aussi est tributaire du role source c.à.d. que ce role ne peut être créé sans que le role source l'ait été auparavant .Cette situation est modélisée par le formalisme de la figure 4.3 suivant:

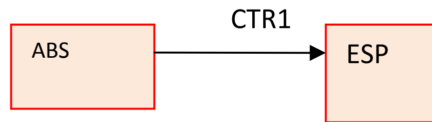


Fig 4.3 Exemple d'interdépendance entre 2 roles

d- La structure du diagramme.

Le diagramme proposé comporte plusieurs niveaux correspondant chacun à une super-classe variable : optionnelle ou variante .Chaque niveau compte deux colonnes (ou deux sous diagrammes) : une colonne qui recense les roles de chaque classe ; l'autre qui lui correspond représente les interactions inter-roles sous forme de machine à états pour chaque super-classe selon les cas de variabilité exprimés dans le tableau 4.1 ainsi que les contraintes d'acquisition de ces roles. La figure 4.4 représente la variabilité à travers les deux sous diagrammes .Ainsi on peut dire que *la machine à états exprime la vue évolutive des roles*.

Une machine à états d'UML est généralement associée à une classe particulière. Elle décrit le comportement complet de toutes les instances de la super-classe à travers ses variant-roles. Le comportement d'un variant-role est décrit par un ensemble d'états et de transitions.

Les états décrivent certaines conditions sur le role et les transitions indiquent les réactions possibles du role avec le respect des conditions de l'état courant et les événements qui déclenchent ces réactions. Une transition est définie par :

– Un événement déclencheur de la transition.

– Une condition garde de la transition.

– Une action exécutée lorsque la transition est tirée.

Nous avons choisi de ne représenter dans notre modèle que les machines à états sans hiérarchie et sans orthogonalité.

Dans ces sous diagrammes, La variabilité est exprimé selon deux aspects : L'optionalité et la variation.

d.1 Pour l'optionalité.

Les superclasses concernées par l'optionalité ont une dimension de 0,1.

Ceci représente dans un diagramme de classe classique une association entre la classe mère et la classe optionnelle de cardinalités : 1 ⟶ 0,1. Le 1 du coté de la classe mère et les 0,1 du coté de la classe optionnelle.

Ces classes ont un seul role ou pas de role du tout et ne peuvent évoluer vers d'autres roles. Elles sont représentées dans notre modèle du diagramme de roles dans la figure 4.4 comme suit :

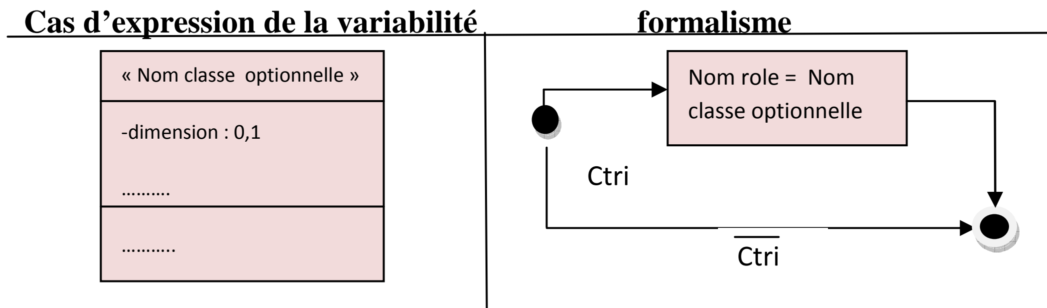


FIG 4.4 Formalisme d'une super-classe optionnelle

d.2 Pour la variation.

Les sous classes variantes sont représentées sous forme de roles de la super-classe variable .Cette dernière peut avoir des sous-classes (roles) obligatoires ou variantes. Ces deux types sont représentés sous forme de classes role appartenant à la super classe variable.

Un listing des contraintes est annexé au diagramme.

d-3- Mapping représentant l'aspect dynamique des variants roles à travers des machines à état.

Le tableau 4.1 résume les principaux cas modélisant l'évolution des variants-roles.

A noter que l'emboitement des formalismes utilisés à travers des regroupements est possible dans le sous diagramme d'état-transitions.

4.3 Intégration de l'association role.

Cette section propose un ensemble de mécanismes pour l'intégration à UML de l'association role et de l'évolution des variant-roles par une machine à état pour chaque point de variation.

4.3.1 L'introduction du role à travers l'association « *role* » :

L'association « *role* » est proche de la généralisation associant une sous-classe à une super-classe mais elle en diffère par son aspect dynamique. Dans le contexte de la généralisation, un objet est une instance statique d'une classe, contrairement à notre association « *role » ou à travers les classes roles le modèle prend en compte les changements dynamiques des variants de la super-classe.*

Nous avons choisi le concept d'association pour représenter la variabilité dans les diagrammes de classe à travers le concept de role.

On décrira dans la section suivante comment intégrer cette association dans le méta modèle UML.

Point de variabilité	*formalisme de représentation*
Classe role optionnelle Nom classe optionnelle	CTR1 Nom classe optionnelle $\overline{CTR1}$
Classe role obligatoire Nom classe obligatoire	Role i
Groupement de classes roles obligatoires Nom classe i · Nom classe j · Nom classek	role i · role j · role k
Exclusion mutuelle de classes role variantes Classe k Role i · Role j	CTRi ROLE i ROLE j $\overline{CTRi}$
Role réflexif ; role jouant le même role plusieurs fois Nom classe role	Nom classe optionnelle CTRi

Point de variabilité	*formalisme de représentation*
Dépendance Classe variants Classe Role i · Classe Role j	Role i · Role j Le role j dépend du role i
OU exclusif –exclusion systematique- Classe variants Nom classe i · Nom classe j · Nom classe k	Role i CTRi Role j CTRj Role k CTRk
OU inclusif –inclusion systématique- Classe k Role i · Role j	CTRi ROLE i ROLE j

Le tableau 4.1 Récapitulatif des principaux cas modélisant l'évolution des variant-roles.

a- Sémantique de l'association « *role* » : Dans cette section nous présentons la sémantique de l'association « *role* ».

a.1 Structure générale

- L'association « *role* » (schématisée par V : v ← R) relie une classe, appelée classe variable ou classe de variants (V), et une autre classe, appelée classe de rôles (R), qui décrit les rôles dynamiques pour la classe variable V.

- Les instances de la classe variables V jouent des rôles, lesquels sont des instances de la classe de rôles R.

- Les instances de la classe variable s'appellent des variants, tandis que les instances de la classe des rôles s'appellent des rôles.

- Les classes de variants sont utilisées pour représenter les propriétés statiques des entités alors que les classes de rôles sont utilisées pour modéliser leurs aspects dynamiques.

- Une entité non évolutive (role non évolutif ou classe optionnelle) est représentée comme une instance permanente et exclusive d'une classe.

- Une entité évolutive, en plus d'être une instance permanente et exclusive d'une classe de variants, est représentée par un ensemble d'instances de la classe de rôles.

- Quand une classe variable acquiert un nouveau rôle, une nouvelle instance de la classe des rôles appropriée est créée.

- Si une classe variable perd un rôle, l'instance de la classe des rôles correspondante n'est pas prise en compte.

a.2 Exemple illustrant l'association role.

La Figure 4.5 montre deux associations *role* qui relient la classe de variants *méthodes* (méthodes de calcul) aux 2 classes de rôles *basée sur la distance* et *basée sur le type de conduite.*

La classe *méthode de calcul* définit les propriétés permanentes des variants « *méthode* » dont les attributs *sont : type fuel ; indice fuel ; quantité consommée ; quantité disponible ; initialiseur du compteur de kilométrage cumulé ; compteur de kilométrage cumulé date-début-utilisation-plaquettes-freins et .date-changement-plaquettes-freins….*

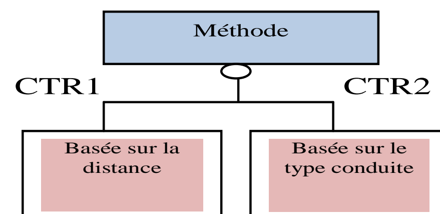


Figure 4.5 Un exemple de l'association role .

La sous classe *basée sur la distance* définit quelques propriétés transitoires en tant que méthode basée sur la distance parcourue du véhicule dont les attributs *: distance parcourue entre deux arrêts successifs, nombre arrêts du véhicule …*

De la même façon, la sous classe *basée sur le type de conduite* définit quelques propriétés transitoires en tant que méthode basée sur le type de conduite dont les attributs *: périodes entre les changements de vitesse, périodes entre les freinages successifs et la vitesse maximale utilisée……*

Notre modèle de rôles supporte également l'instanciation multiple de la même classe. En effet, un variant peut devenir une instance plus qu'une fois de la même classe, tout en maintenant son appartenance à la classe. Par exemple, une méthode basée sur la distance peut être utilisée avec deux moteurs différents essence et diesel. Elle jouera donc deux rôles différents.

4.3.2 Expression de la variabilité par le changement dynamique de la classe :

Soit une classe de variants V représentant un point de variation. Les variants de ce point de variation sont des classes de rôles Role1 et Role2 représentant deux classes variantes de la classe variable V. Les variants peuvent évoluer selon 4 scenarios différents :

- **L'acquisition**, notée →R, indique qu'un variant peut acquérir un rôle de R sous certaines conditions ou automatiquement. Par exemple, "→basée sur la distance " signifie que la méthode peut acquérir le role de méthode basée sur la distance.
- **La cession ou la perte,** notée R→, indique qu'un variant peut cesser de jouer un role de R automatiquement ou sous certaines conditions. Par exemple, "méthode basée sur une commande →" signifie que la méthode de calcul cesse d'être basée sur la commande.
- **La Mutation.** ou acquisition d'un nouveau role avec perte de l'ancien .Elle est notée $R_1 \rightarrow^{-} R_2$ pour les classes de rôles R_1 et R_2, avec R_1 distincte de R_2 .Cette notation indique qu'un variant jouant le rôle r1 (instance de R_1) peut cesser de jouer le role r1 et acquérir le rôle r2 (instance de R_2). En plus, les rôles de R_2 ne peuvent pas être créés indépendamment des rôles de R_1. ce type d'acquisition peut être dans un seul sens ou dans les deux sens selon les contraintes imposées (ou exclusif /ou Mutex/ ou dépendances).
- **Le Prolongement .**ou acquisition d'un nouveau role sans perte de l'ancien .Elle est notée $R_1 \rightarrow + R_2$, avec R_1 non nécessairement distincte de R_2, indique que le variant qui joue le rôle r1 de R_1 peut gagner un nouveau rôle r2 de R_2 en retenant r1 (ou inclusif).

On remarquera que les 4 situations relatives aux différents états de l'association « *role* » et des types de variants couvrent sémantiquement la totalité des cas d'expression de la variabilité décrits dans la section précédente.

4.3.3 Expression des contraintes de transition à travers les prédicats.

Soit une classe de variants V représentant un point de variation en relation avec les classes des rôles R_1 et R_2**.** Un prédicat de transition est associé à R_1 pour décrire les conditions nécessaires et/ou suffisantes sur la façon dont les objets jouant des rôles de R_1 peuvent explicitement ou automatiquement acquérir des rôles dans R_2. Les prédicats de transition sont décrits en utilisant le langage déclaratif formel OCL (*Object Constraint Language*) d'UML. Dans ce qui suit, nous donnons quelques exemples de prédicats de transition exprimant les 4 scenarios d'évolution des variants par les roles :

- *Pour la contrainte CTR1. « Une méthode jouant le role d'une méthode de calcul basée sur la distance peut l'être pour le fuel essence normale et pour le fuel essence sans plomb en même temps ». le prédicat* « nombre-type-fuel ≤ 2 *Λ* énergie = 'essence'» déclare 2 comme étant le maximum de types de fuel que cette méthode peut prendre en charge l'énergie de type essence en même temps. La notation pour CTR1 est la suivante :

  B*asée sur la distance* ⟶ + B*asée sur la distance*

- *CTR2. « une méthode jouant le role d'une méthode de calcul basée sur la distance peut être en même temps basée sur le type de conduite si la puissance du moteur est supérieure ou égale à 9 CV* » car nécessitant une économie dans la consommation du fuel

La notation pour CTR2 est la suivante :

Basée sur la distance ⟶ + *Basée sur type conduite*

- *CTR3. « Une méthode jouant le role de méthode basée sur une commande automatique peut basculer vers une méthode basée sur type de conduite si statut de la commande = 0 »* càd si la commande tombe en panne. le prédicat statut-commande = 0 est associé aux méthodes qui basculent vers un calcul basé sur le type de conduite.

La notation pour CTR3 est la suivante :

Basée sur la commande ⟶ - *Basée sur type conduite*

- *CTR4. « Une méthode jouant le role de méthode basée sur la distance parcourue peut basculer vers une méthode basée sur une commande automatique si quantité fuel restante inferieure 30 % quantité globale »* le prédicat « *quantité fuel restante < 30 % quantité globale* » est associé aux méthodes basées sur la distance qui basculent vers un calcul basé sur une commande automatique.

La notation pour ctr4 est la suivante :

Basée sur la distance ⟶ - *Basée sur commande*

- *CTR5. « la méthode de calcul basée sur la distance est la seule méthode prise en compte » le prédicat « puissance du moteur inferieure ou égale à 5 CV » est associé à cette situation car dans cette situation le moteur ne consomme pas beaucoup de fuel .*

La notation pour ctr5 est la suivante : ⟶ *Basée sur la distance*

Nous présentons ci-dessous le processus d'intégration de l'association « *role* » dans UML en respectant les éléments du formalisme de spécification présentés dans les sections précédentes.

4.3.4 Intégration de l'association role dans les méta-modèles UML .

4.3.4.1 Syntaxe abstraite et meta-modele de l'association role .

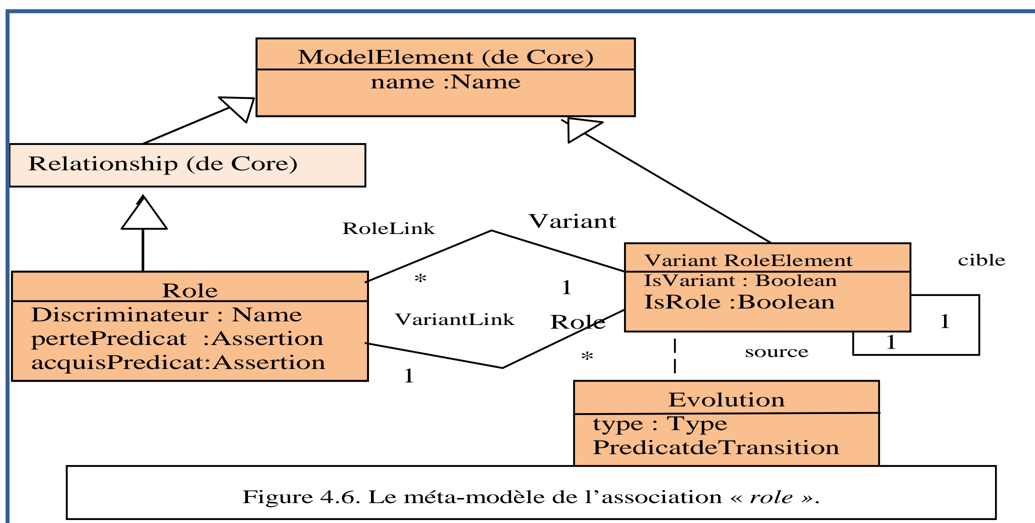


Figure 4.6. Le méta-modèle de l'association « *role* ».

Nous définissons ici le méta-modèle de l'association role présentée dans la section précédente. Nous définissions « *role* » comme sous-classe directe de *Relationship* appartenant au paquetage Core (cf. fig 4.6).

-Au lieu de créer deux méta classes *VariantElement* et Ro*leElement comme sous classes directes de ModelElement* et partant du principe qu'un role est toujours associé à un variant , nous avons opté pour la création d'une seule méta-classe que nous avons appelé VariantRoleElement et qui permet de gérer les deux concepts en même temps grâce à deux attributs s'excluant mutuellement (exclusif) Is*Variant* pour les variants et IsRole pour les roles.

-Nous définissons *VariantRoleElement* comme sous-classe directe de *ModelElement*.

-La méta-classe *VariantRoleElement* représente en même temps les classes de variants et les classes de rôles impliquées dans une association « *role* ».

Notre méta-modèle est reporté sur la figure 4.6. Nous allons détailler dans la suite chacun des éléments de ce méta-modèle.

La figure 4.7 présente une association « *role* » reliant deux classes A et B.

Les classes V et R sont des instances de la méta-classe VariantRoleElement.

Pour simplifier le méta-modèle nous diront que A et B sont des VariantRoleElement avec les particularités suivantes :

- V (la classe de variants) est un VariantRoleElement avec l'attribut IsVariant= « Vrai » et IsRole= « Faux ».
- R (la classe de rôles) est un VariantRoleElement avec l'attribut IsRole= « Vrai » et IsVariant= « Faux ».

On peut ainsi considérer que V et R de la figure 4.7 représentent, les qualificateurs ***variant*** pour V et ***role*** pour R dans la figure 4.6.

L'association « *role* » entre V et R, peut être décomposée en deux sous-associations : VariantL*ink* (reliant un rôle à un variant) et *roleLink* (reliant un variant à un rôle) comme le montre la figure 4.6

4.3.4.2 Définition de l'association *« role »*

Dans le méta-modèle, « role » est une sous classe directe de *Relationship* qui sert à modéliser les associations .Elle représente le lien entre une classe de variants et une classe de rôles. Elle comporte les attributs suivants :

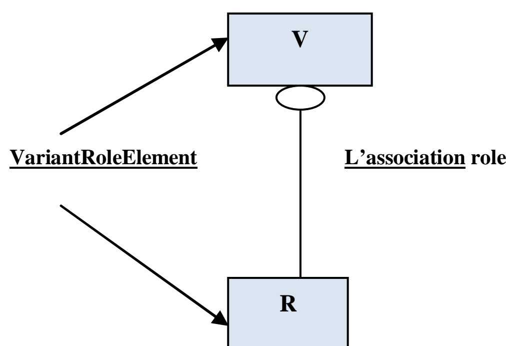


Fig 4.7 Exemple de roles.

- **discriminateur :** indique la partition à la quelle le lien de role appartient. Il désigne tous les liens de role qui partagent la même classe de variants. Chaque partition représente une dimension orthogonale qui regroupe un ensemble de classes de rôles correspondant à une classe de variants. Les classes de rôles de la même partition ont toutes le même discriminateur.
- **pertePredicat :** une assertion de type ocl_expression décrivant les conditions nécessaires et/ou suffisantes sur la façon dont les variants peuvent explicitement ou automatiquement perdre des rôles.

- **acquisPredicat :** une assertion de type ocl_expression décrivant les conditions nécessaires et/ou suffisantes sur la façon dont les variants peuvent explicitement ou automatiquement acquérir des rôles.

La méta-classe *role* est associée par deux associations Variant et Role à la méta-classe VariantRoleElement.

a. Les autres Associations du méta-modèle :

- **Variant :** désigne un *VariantRoleElement* qui peut jouer un rôle.
- **Role :** désigne un *VariantRoleElement* qui représente un éventuel état d'un variant.

b. Définition des cardinalités des associations.

- **1** du côté du qualificateur *Variant* : un seul VariantRoleElement de type *classe de variants* (càd un VariantRoleElement avec les attributs IsVariant=vrai) participe dans l'association « *role* ».
- **1** du côté du qualificateur *role* : un seul VariantRoleElement de type *classe de rôles* (càd un VariantRoleElement avec les attributs IsRole=vrai) participe dans l'association « role ».
- **1** du côté du qualificateur *VariantLink* : un VariantRoleElement ne peut avoir qu'une seule relation envers la classe de variants.
- ***** du côté du qualificateur *roleLink* : un VariantRoleElement peut avoir plusieurs relations avec les classes de rôles.

c. Définition de VariantRoleElement.

C'est une sous-classe de « ModelElement » qui peut participer dans une association « *role* ». VariantRoleElement est une méta-classe abstraite.

Attributs :

- **IsVariant :** est de type Booléen. Il a la valeur *Vrai* pour une classe de variants.
- **IsRole :** est de type Booléen. Il a la valeur *Vrai* pour une classe de rôles.

Pour les classes intermédiaires telle que R_1 dans la composition de roles $R_2 \rightarrow V$ $R_1 \rightarrow V$, les attributs IsVariant et Is*Role* auront tous les deux la valeur « Vrai », puisque R_1 est à la fois une classe de variants et une classe de rôles.

Associations :

- **RoleLink :** c'est la liaison qui part de la classe de variants vers la classe de rôles.
- **VariantLink :** c'est la liaison qui part de la classe de rôles vers la classe de variants.

d. Le role :

- La cession ou la perte d'un rôle est contrôlée par l'attribut pertePredicat du modèle role. Quand l'assertion pertePredicat est satisfaite, on détruit le rôle de cette liaison.
- L'acquisition d'un rôle est contrôlée par l'attribut acquisPredicate du modèle role. Quand l'assertion acquisPredicate est satisfaite on construit un nouveau rôle pour cette liaison.

e. La classe association Evolution. L'association *« Evolution »* est une classe-association récursive qui relie une classe de VariantRoleElement appelée classe *source* et une classe de VariantRoleElement appelée classe *cible* .

Elle permet de spécifier les notions d'évolution càd les prolongements et les mutations décrites dans la section 4.3.2 ainsi que les prédicats de transition des variants décrits dans la section 4.3.3.

Les Attributs de l'association « Evolution ».

- **Type** est un attribut à deux valeurs qui définit le type d'évolution des roles. Il peut prendre une des deux valeurs : « prolongement » ou « mutation ».
- **PrédicatdeTransit*ion*** est un attribut de type Assertion. Il décrit les conditions nécessaires à une instance de la classe source pour pouvoir évoluer vers la classe destination.

Les deux associations liées à évolution :

- **Source**. désigne une classe de rôles (une classe VariantRoleElement avec l'attribut IsRole=vrai) qui va subir soient les règles du prolongement ou de la mutation.
- **Cible**. désigne la classe des rôles destination après que les conditions de transition sur la classe source aient été vérifiées.

4.3.5 Intégration de la machine à états par l'usage du mécanisme de virtualité.

La technique utilisée pour intégrer la machine à état dans UML est la virtualité d'une machine à état transition .Ceci signifie que son comportement peut être redéfinie par une autre machine de raffinement associée à un produit particulier.
C'est l'une des techniques utilisées pour paramétrer une machine à états en vue de l'appliquer non pour un état généraliste mais pour un état spécifique. Pour notre cas cet état est une classe « role ». On peut aussi utiliser cette technique pendant la dérivation de produit ou le comportement de la machine virtuelle sera remplacé par le comportement de la machine à états de raffinement associée au produit.

La virtualité est introduite par le stéréotype <<virtual>> et le tagged value ***virtualPart*** qui indique l'occurrence de la machine virtuelle.

VirtualPart pour notre cas, représente une partie du diagramme de classes qui exprime la variabilité. Le diagramme de classes est divisé en plusieurs partitions virtuelles au nombre de points de variation et d'optionalité existants. Chaque partie peut renfermer soit une classe variation avec ses sous-classes variantes (roles) soit une classe optionnelle.

Les partitions sont nommées partition du diagramme de classes numéro i (DC1, DC2…).
Prenons l'exemple de la partition DC4 dans la figure 5.4 du chapitre 5.
La Figure 4.8 montre un exemple d'une machine à états qui référencie une machine virtuelle DC pour la partition DC4 du système surveillance du moteur.

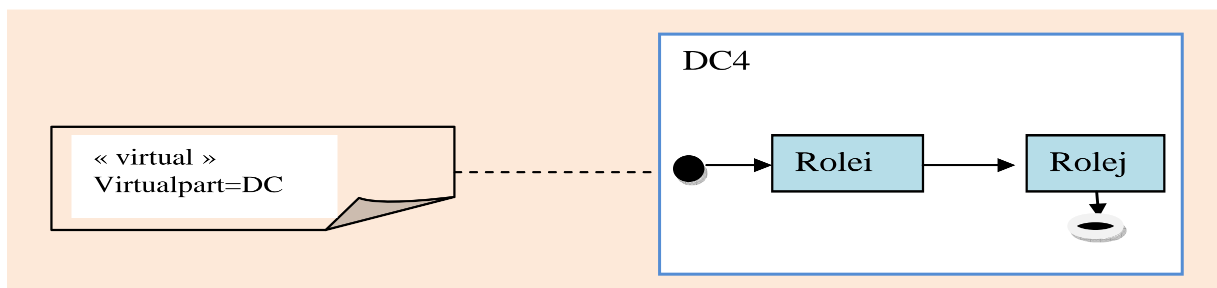


Fig.4.8 Un exemple d'une machine à états référençant une machine virtuelle DC pour la partition DC4

4.4 Vers un Profil UML :

UML introduit la notion de profil UML pour regrouper un ensemble de stéréotypes, de tagged values et de contraintes.

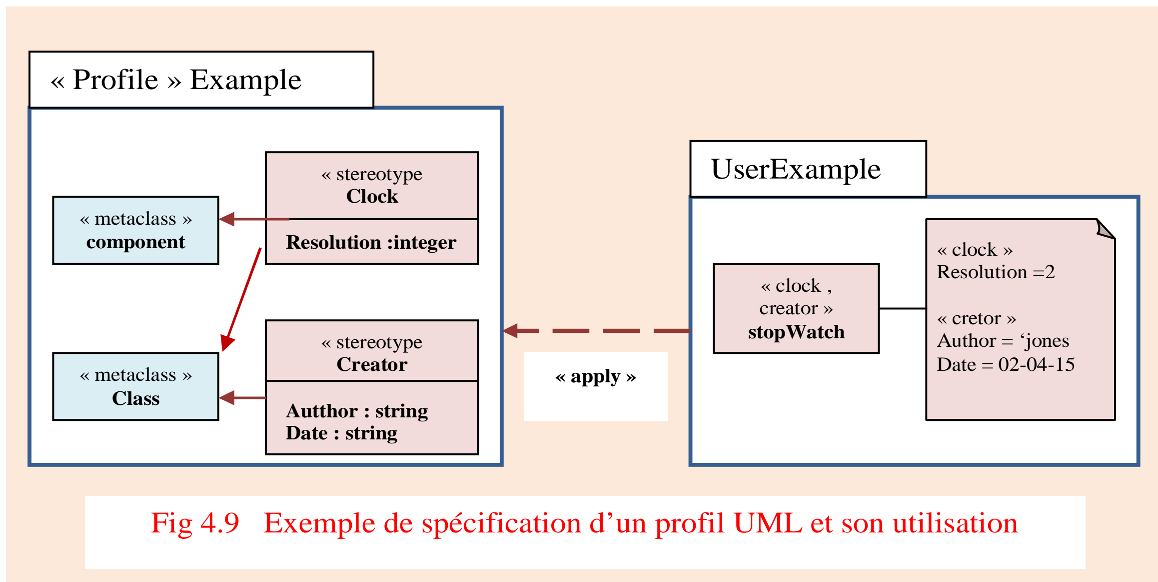


Fig 4.9 Exemple de spécification d'un profil UML et son utilisation

La Figure 4.9 montre un exemple de définition de deux stéréotypes [17] : *Clock* et *Creator*. Le stéréotype Clock étend les deux méta-classes *Class* et *Component* et il définit une propriété résolution comme un tagged value dont le type est integer. Le lien d'extension est noté par une flèche orientée (cf. Figure 4.9). Cette figure montre un exemple du profil UML appelé *Exampl*e regroupant les deux stéréotypes *Clock* et *Creator.*

Les profils dans UML2.0 sont notés comme des paquetages UML avec le stéréotype <<profile>>. Le paquetage UserExample est un exemple de modèle utilisateur basé sur le profil Example. La classe StopWatch est définie avec les deux stéréotypes de profil.
Les tagged values associées aux stéréotypes sont définis comme des notes UML (cf. Figure 4.9).

Dans le chapitre 3 et les sections précédentes de ce chapitre, nous avons proposé cinq mécanismes pour la spécification de la variabilité dans les diagrammes de classes (3 nouveaux stéréotypes, une nouvelle association et une machine à état virtuelle).

Dans ce qui suit, nous allons regrouper ces extensions sous forme d'un profil UML pour modéliser la variabilité dans les LdP. La Figure 4.10 montre la structure de ce profil.

Nous suivons la notation des profils dans UML2.1 ;

- Une méta-classe est présentée comme une classe stéréotypée <<metaclass>>
- Un stéréotype est présenté comme une classe stéréotypée <<stereotype>>.

Ces extensions sont regroupées dans le paquetage que nous avons désigné par « *software Product Line* » de la figure 4.10.

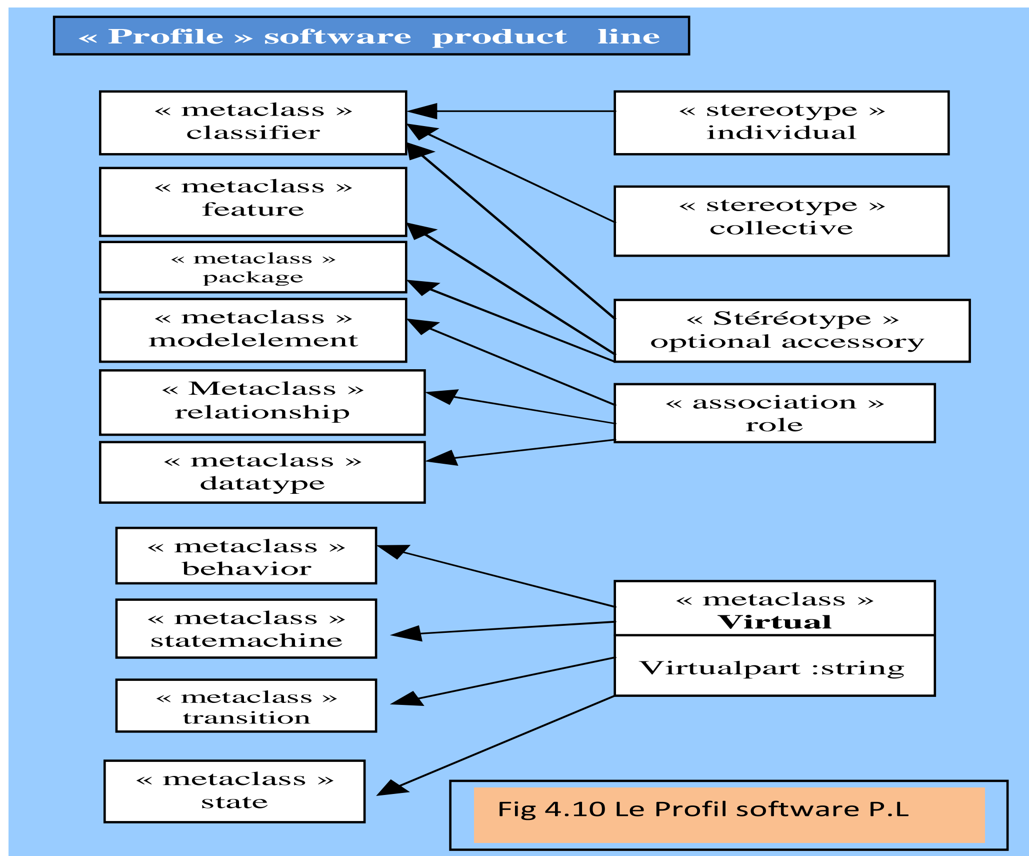


Fig 4.10 Le Profil software P.L

4 .5 Conclusion.

Nous avons proposé dans ce chapitre, une gestion de l'évolution de la variabilité à travers celle des variant-roles par l'usage de machines à état-transitions. Pour cela nous avons introduit une nouvelle association désignée « *role* » qui lie les variant-roles modélisés sous forme de classes aux classes variables dans le diagramme de classes.

Les variants sont perçus dans le diagramme de classes comme des roles pouvant être joués par les classes représentant des points de variation. Ces dernières peuvent jouer un seul role parmi plusieurs ou plusieurs roles en même temps. De nouveaux roles peuvent être joués et donc ils seront ajoutés aux classes *role* liées à cette classe variable. De plus, l'évolution des roles exprime celle de la variabilité au niveau des points de variation. Ce modèle à l'aide d'un autre diagramme UML, les machines à état-transitions, offrira des facilités au concepteur pour la gestion de l'évolution de la variabilité des LdPs.

Deuxième partie
Contributions

Chapitre 5.

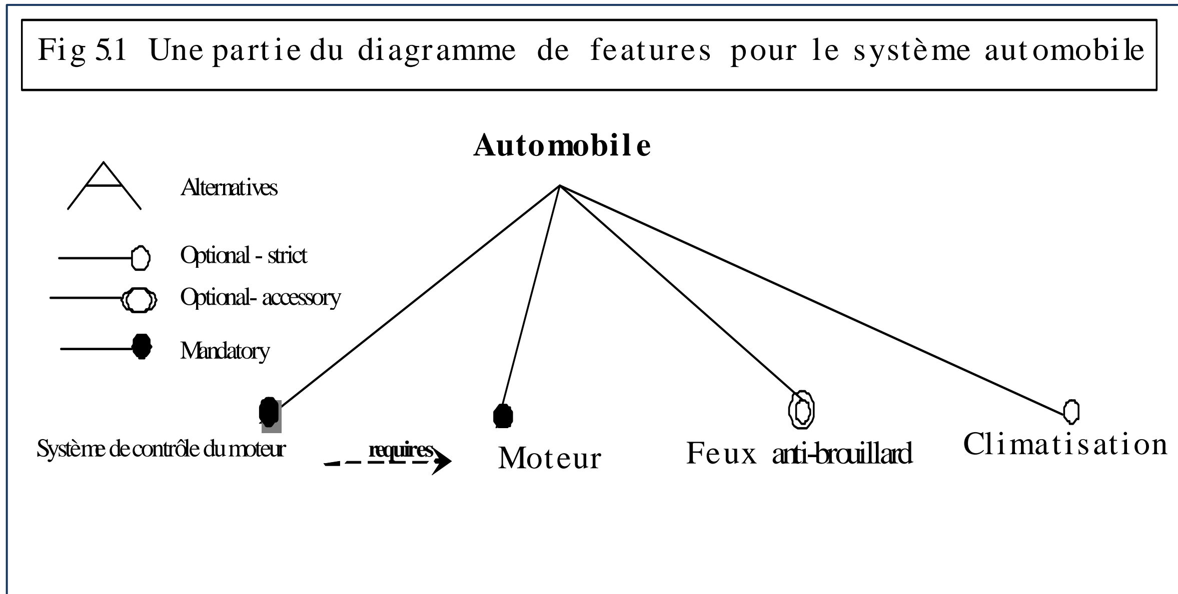
Application

au système automobile.

5-1- Introduction.

Le domaine de l'automobile est très représentatif de la situation dans laquelle les systèmes embarqués se développent. Il comprend de très riches fonctionnalités tant complexes que diversifiées. C'est là où les systèmes embarqués sont les plus nombreux. C'est un domaine où

Fig 5.1 Une partie du diagramme de features pour le système automobile



de nouvelles fonctions du véhicule sont de plus en plus basées et contrôlées sur/par le logiciel [73]. Cet état de fait nous a motivés pour prendre le sous-système moteur/feux antibrouillard qui fait partie du système automobile comme exemple applicatif de notre solution pour la

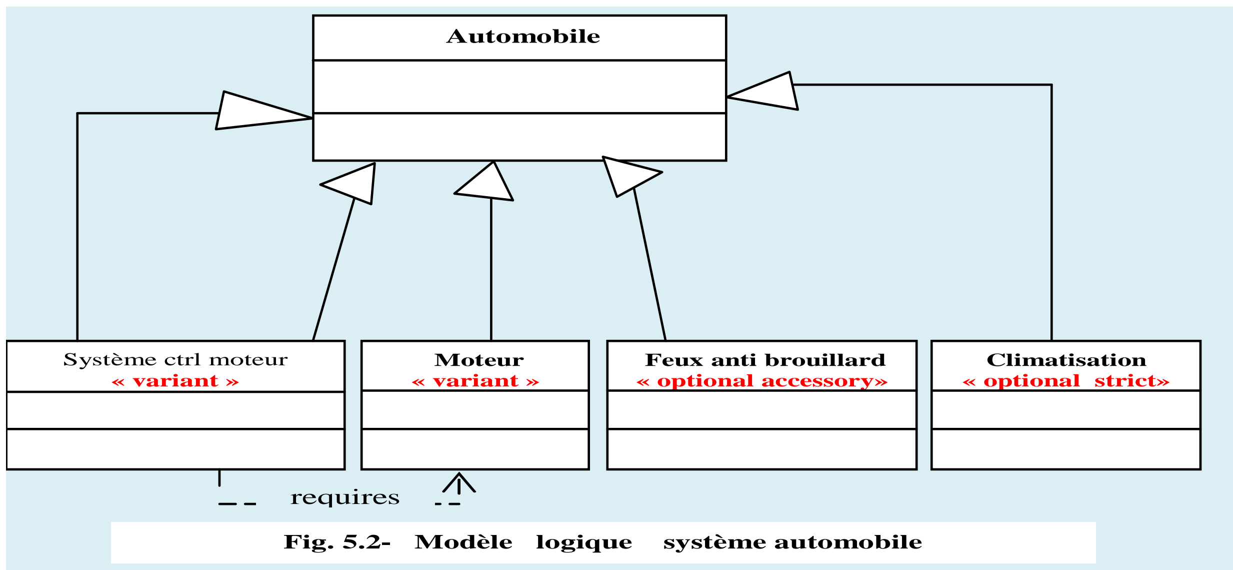


Fig. 5.2- Modèle logique système automobile

modélisation de la variabilité d'une LdP .Le diagramme de features correspondant est représenté dans la figure 5.1.

5-2- Application des solutions apportées par notre modèle.

Notre approche utilise comme point de départ les points variables (point de variation et optionnels) comme représentés dans le diagramme de classes de la figure 5.2.

5.2.1 Pour l'optionalité.

L'exemple de la classe « *feux antibrouillard* » –cf fig 5.2- convient pour démontrer l'utilité de ce mécanisme .En effet , cette classe représente une classe variable de type « *optionnel accessoire* ». Un point d'extension est prévu pour ce type de feux. L'endroit prévu se situe généralement sous le pare-choc avant du véhicule .Tous les attributs et toutes les méthodes de cette classe accessoire prennent automatiquement le type optionnel et sont représentés dans la classe par le stéréotype « optional » -selon le mapping du tableau 3.2 du chapitre 3.

La climatisation par contre est un exemple d'une feature optionnelle stricte.

5.2.2 Pour les extensions par les stéréotypes au niveau des classes.

L'exemple de la classe *Moteur* convient pour démontrer l'importance de ces mécanismes.

Cette classe est de type variable .Elle représente un point de variation dans la LdP automobile.

-*Les Attributs de moteur sont :* Marque ; Volume ; Puissance ; Type (électrique, essence) ; Charge-totale-batteries ; Autonomie-batteries ; Type-batterie (lithium……..); Refroidissement ; Période-vidange ; Réf-lubrifiant ; Type-filtre-huile; Dispo-anti-pollution , Smart-engine-contrôle (ou module de contrôle intelligent du moteur).

- *Les Méthodes de moteur sont :* Ajout ; Modification ; Recherche ; MAJ-charge-batteries, MAJ-filtre ; Maj-dispositif-anti-pollution ; Détection-anomalie ; Consulter-état-batteries , Consulter-niveau-lubrifiant ;consulter-niveau-Essence; Consulter-niveau-température-moteur; Consulter-niveau-température-lubrif et Maj-smart-engine-ctrl.

Appliquons les étapes du sous-modèle pour la prise en compte de la variabilité dans la classe **Moteur** .

Etape1.

1.1 Déterminons les attributs et les méthodes de M*oteur* qui ne concernent qu'une seule instance de Moteur. Ce sont :

i-instance *: moteur électrique*.

les attributs sont : charge-totale-bat ; autonomie-batteries ; type-batteries
les méthodes sont : MAJ-charge-bat, consulter-état-batt .

ii-instance : *moteur essence* .

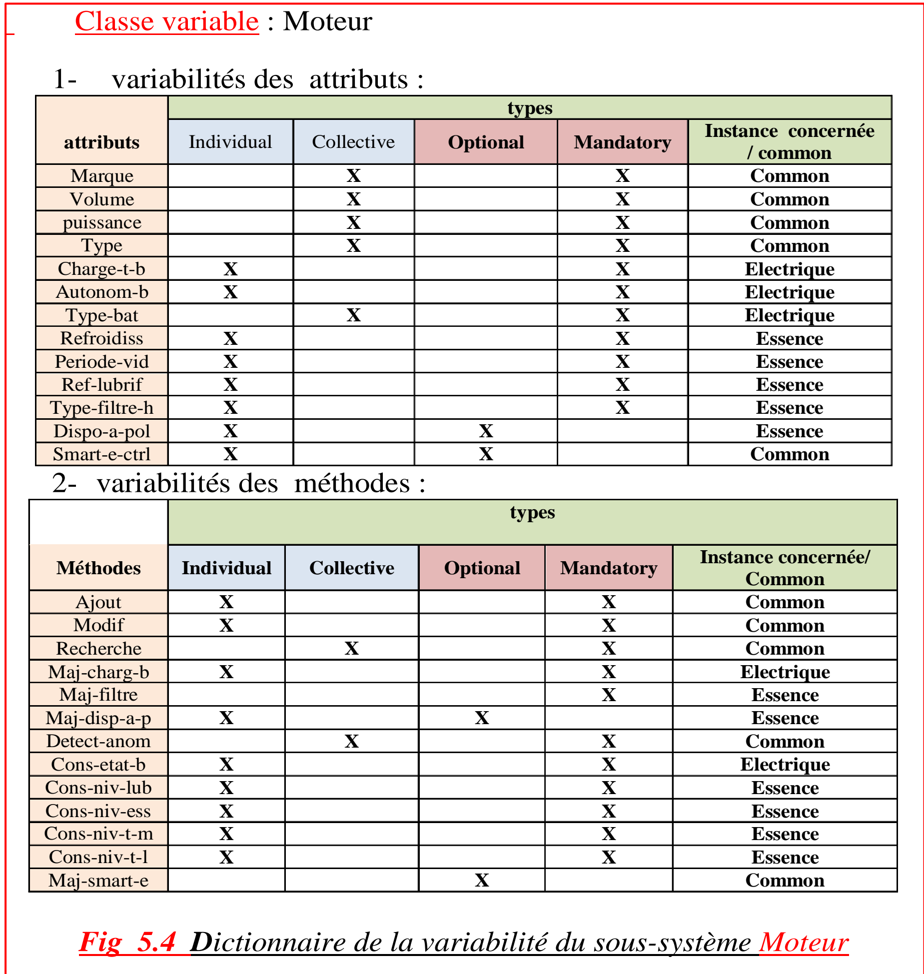
Les attributs sont : refroidissement ; période-vidange ; référence –lubrifiant ; type-filtre-huile, dispo-anti-pollution

Les méthodes sont : Maj-dispositif-anti-poll ; consulter-niv-lubrifiant; consulter-niv-Essence ; consulter-niv-temp-lubrifiant .

les attributs et les méthodes de M*oteur* qui sont communs à la population quelque soit le type de moteur sont :

i. *Les Attributs* : Marque ; volume ; puissance ; type et smart-engine-control

ii. Les méthodes : Ajout ; modification ; recherche ;détect-anomalies; consulter-niveau-tempera-moteur et MAJ-smart-engine-ctrl .



Classe variable : Moteur

1- variabilités des attributs :

attributs	types				
	Individual	Collective	Optional	Mandatory	Instance concernée / common
Marque		X		X	Common
Volume		X		X	Common
puissance		X		X	Common
Type		X		X	Common
Charge-t-b	X			X	Electrique
Autonom-b	X			X	Electrique
Type-bat		X		X	Electrique
Refroidiss	X			X	Essence
Periode-vid	X			X	Essence
Ref-lubrif	X			X	Essence
Type-filtre-h	X			X	Essence
Dispo-a-pol	X		X		Essence
Smart-e-ctrl	X		X		Common

2- variabilités des méthodes :

Méthodes	types				
	Individual	Collective	Optional	Mandatory	Instance concernée/ Common
Ajout	X			X	Common
Modif	X			X	Common
Recherche		X		X	Common
Maj-charg-b	X			X	Electrique
Maj-filtre				X	Essence
Maj-disp-a-p	X		X		Essence
Detect-anom		X		X	Common
Cons-etat-b	X			X	Electrique
Cons-niv-lub	X			X	Essence
Cons-niv-ess	X			X	Essence
Cons-niv-t-m	X			X	Essence
Cons-niv-t-l	X			X	Essence
Maj-smart-e			X		Common

Fig 5.4 Dictionnaire de la variabilité du sous-système Moteur

Comme seulement ,certaines marques et versions de moteurs supportent la smart technologie, l'attribut smart-engine-control est optionnel. La méthode qui le traite (MAJ-smart-engine-ctrl) l'est aussi.

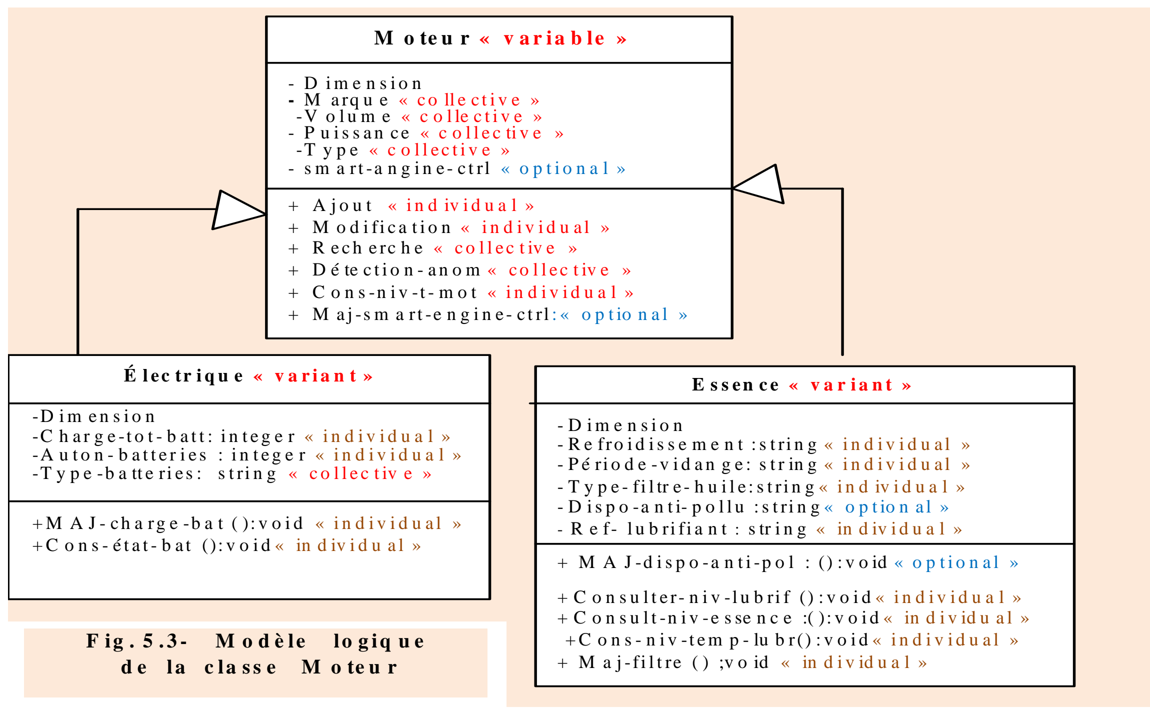


Fig. 5.3- Modèle logique de la classe Moteur

Le traitement de la méthode *consulter-niveau-temp-moteur* ne peut concerner qu'une seule instance. Elle est de type « Individual » tout en étant commune aux deux instances de la classe Moteur.

***Etape 2*.**

Regroupons le résultat de la premiere etape dans le dictionnaire de la variabilité relatif à la classe *Moteur*. Ce mécanisme d'expression de la variabilité est représenté dans la figure 5.4.

***Etape 3*.** Scindons la classe moteur selon les préoccupations des méthodes et attributs vis-à-vis des instances .Le nombre de sous-classes obtenu est de 2 : la classe *moteur électrique* et la classe *moteur à essence*. Appliquons les stéréotypes des tableaux 3.1 et 3.2 du chapitre 3. La version finale du modèle prendra le formalisme de la figure 5.3 et la variabilité au niveau du point de Variation *Moteur* sera exprimée à travers deux alternatives exclusives : *électrique* et *essence*.

3- *Etape 4* (représentation de l'*évolution de la variabilité à travers les variant-roles).*

Pour cette étape, nous allons *construire le diagramme de roles pour montrer l'évolution de la variabilité à travers les variant-roles.* Appliquons le mapping du tableau d-3 du chapitre 4 –section 4.2.3 pour construire les deux volets du diagramme de roles relatif aux classes variantes et optionnelles du diagramme de classes.

L'exemple du système de surveillance du moteur (figure 5.5) convient pour démontrer les

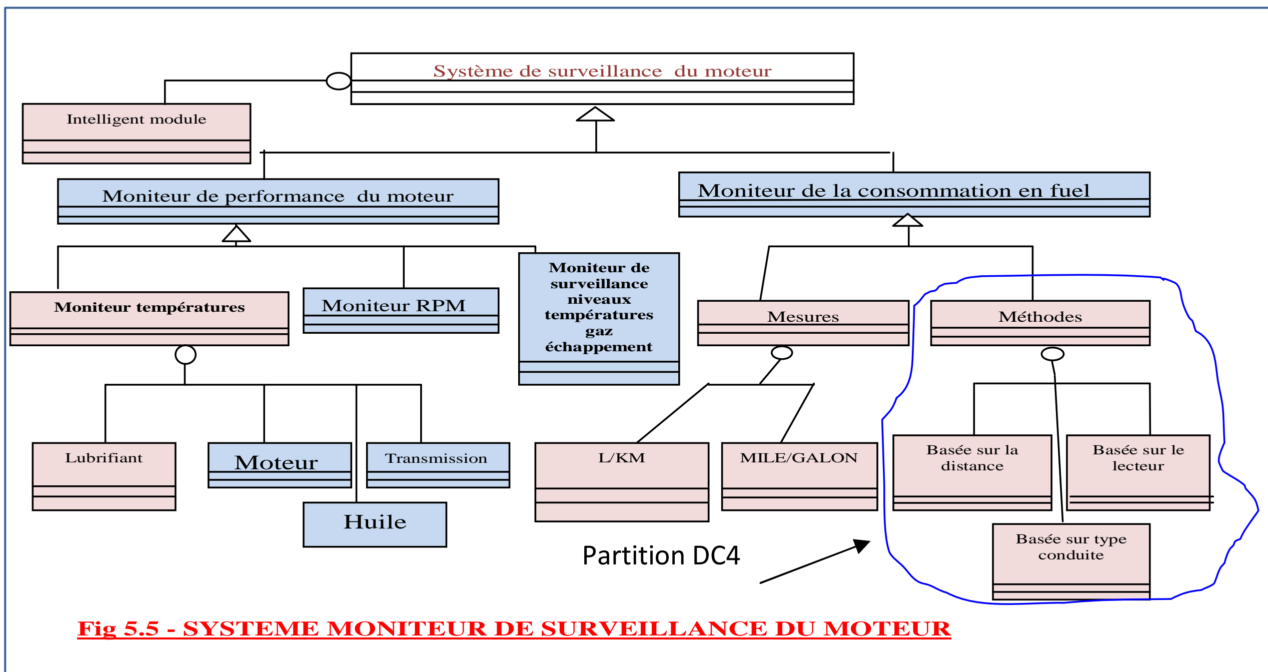


Fig 5.5 - SYSTEME MONITEUR DE SURVEILLANCE DU MOTEUR

mécanismes de notre approche à ce niveau de modélisation de la variabilité.

Les instances de la classe variante *méthode* peuvent jouer 3 roles : un role obligatoire (méthode basée sur la distance) et 2 roles variants (méthode basée sur le type de conduite et méthode basée sur le lecteur) - cf. figure 5.5 - .

Dressons à présent le sous diagramme des états transition pour les roles de la super classe méthode puis la liste des conditions selon lesquelles chaque role peut être joué en rappelant que par défaut la méthode adoptée est basée sur la distance parcourue .Ce role est toujours joué par la classe *méthode* pour tous les produits dérivés -cf. figure 5.6-.

a- <u>Les contraintes d'évolution inter-roles CTRi:</u>

- *CTR1. « Une méthode jouant le role d'une méthode de calcul basée sur la distance peut l'être pour le fuel essence et pour le fuel diesel en même temps ». Le prédicat* « nombre-type-fuel- ≤ 2 » déclare 2 comme étant le maximum de types de fuel que cette méthode peut prendre en charge en même temps pour le type d'énergie.

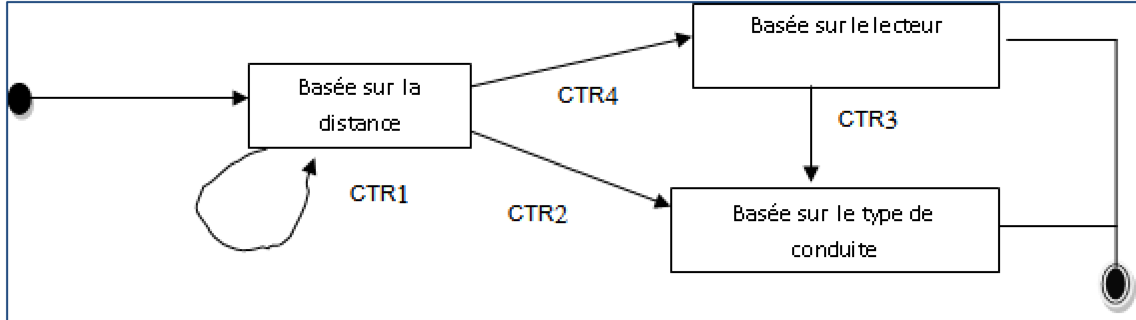


Fig.5.6.Diagramme états-transitions qui montre l'évolution des roles.

- *C1 ... ıt être en même temps basée sur le type de conduite si la puissance du moteur est supérieure ou égale à 9 CV car nécessitant une économie dans la consommation en fuel »*

- *CTR3. « Une méthode jouant le role de méthode basée sur le lecteur peut basculer vers une méthode basée sur le type de conduite si le statut de cette méthode = 0 » càd si le lecteur tombe en panne».* Le prédicat statut-lecteur = 0 est associé aux méthodes qui basculent vers un calcul basé sur le type de conduite.
- *CTR4. « Une méthode jouant le role de méthode basée sur la distance parcourue peut basculer vers une méthode basée sur le lecteur si puissance-moteur ≤ 5 CV .and. statut-basée-distance = 0 » .*Le prédicat « *puissance < 5 et statut-méthode indiquant méthode désactivée »* est associé aux méthodes basées sur la distance qui basculent vers un calcul basé sur un lecteur.

b- Ajoutons à la classe méthode une sous classe optionnelle basée sur la commande automatique et supposons que *basée sur le lecteur* et *basée sur le type de conduite* s'excluent mutuellement et gardons la sous classe basée sur la distance obligatoire tout en supposant qu'aucun role ne peut évoluer vers un autre ou vers lui-même. Le sous diagramme des états transitions correspondant à la classe variation *méthode de calcul* prendrait le formalisme de la figure 5.7.

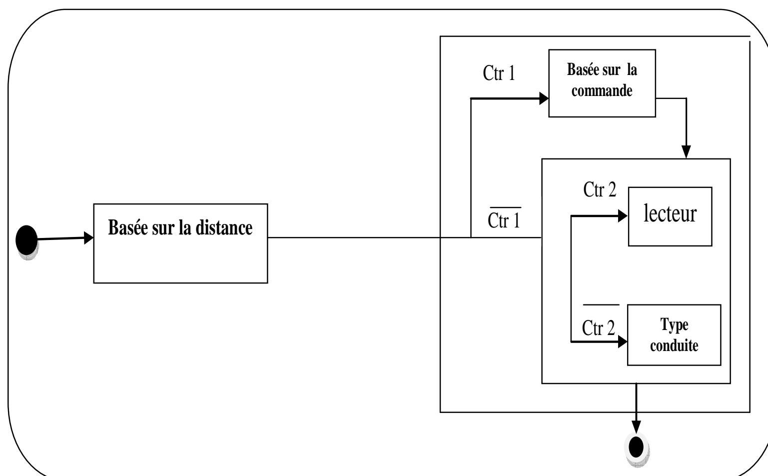


Fig. 5.7 - Le sous diagramme des états transitions correspondant à la classe méthode.

c- <u>Listing des contraintes :</u>

Ctr1 : « modèle = ' GLX' ou 'GLS' ».

Ctr2 : « *puissance <= 5 .and. statut-méthode = 0* »

Deuxième partie
Contributions

Chapitre 6 .

Un sous-modéle pour la gestion de la variabilité architecturale selon les vues

6-1- Introduction

La gestion et la composition des variabilités architecturales a été pendant longtemps un challenge pour les architectes des lignes de produits. Ces derniers ont besoin de décrire comment les variabilités conceptuelles sont représentées et réalisées à travers les décompositions architecturales des lignes de produits. Les variabilités architecturales décrites en termes de décomposition et de choix de conception n'ont pas souvent de correspondance biunivoque avec les décompositions des modèles de features. Cependant, la granularité fine de certaines variabilités architecturales rend difficile leur représentation modulaire et la description de la manière dont ils sont composées à travers les différentes vues .Dans le but de réaliser ces objectifs, nous présentons dans cette partie du mémoire un sous-modèle de gestion de la variabilité architecturale qui permet sa représentation sous des formes hétérogènes dans différentes vues. Le modèle utilise des mécanismes UML capables de référencer des points de variation à travers des vues architecturales multiples tout en supportant les compositions évolutives des variabilités de différentes granularités. Le sous-modèle complète les approches existantes de modélisation des variabilités architecturales mais s'en distingue par l'usage exclusif de mécanismes UML qui lui offrent par leur standardisation une facilité d'adoption pour la représentation de la variabilité.

L'architecture d'une ligne de produits logicielle doit être conçue de manière à répondre rapidement et efficacement aux différentes exigences des utilisateurs relatives à la représentation de la variabilité [74]. Ceci est réalisé en se basant sur les éléments communs des différentes vues conceptuelles des LdPs associés à un ensemble d'éléments variables introduits et manipulés selon les préoccupations des concepteurs et architectes , leur offrant ainsi une conception à large architecture récurrente par opposé aux conceptions traditionnelles d'architectures monolithiques. La problématique est que la réalisation et la composition des variabilités relevant de l'architecture ne peut pas être uniquement exprimée par l'usage de modèles de features [53] conventionnels comme illustrés par [59] mais a besoin d' identifier les points de variation dans le contexte de multiples vues architecturales compte tenu du fait que les variants de niveau architecture sont de nature hétérogène (allant des interfaces de composants optionnels jusqu' aux nœuds d'alternatives hébergeant des éléments d'architecture dans la vue de déploiement) .
De plus, les concepteurs de LdP ont besoin d'exprimer la manière dont des variants architecturaux sont composés avec les éléments architecturaux communs. Un grand nombre d'approches ,exemples [59,60,61,62,63,75,76,77,78] ont tenté de solutionner ses problèmes et supporter le traitement systématique des variabilités dans tout le cycle de vie du logiciel .Cependant ,Les plus récentes d'entre elles ne sont pas parvenues à supporter pleinement la représentation orthogonale des variabilités architecturales [63].
Certaines techniques de modélisation de la variabilité se fondent exclusivement sur l'utilisation des modèles de features [53,61,76] tandis que d'autres [59,62] utilisent des mécanismes de modélisation avancés uniquement pour relier les modèles de features aux modèles architecturaux. OVM [59] par exemple se limite à documenter la variabilité plutôt que d'exprimer sa composition avec des éléments architecturaux communs, par contre VML [77] en se concentrant sur quelques vues seulement oblige les intervenants sur les LdP à maitriser un langage supplémentaire pour pouvoir modéliser et gérer la variabilité.

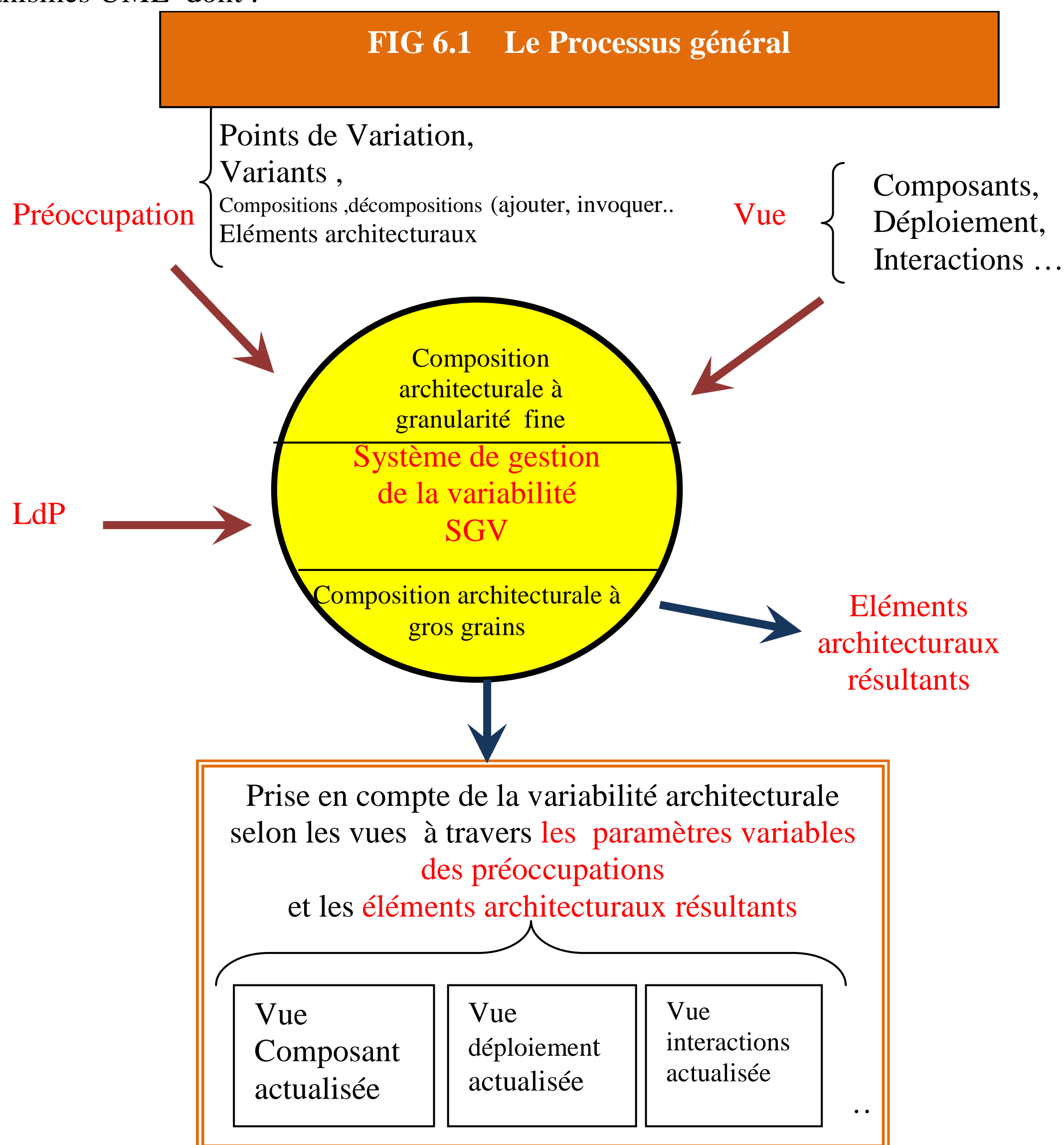
Dans le même contexte que les précédentes approches présentées dans les derniers chapitres et qui proposent des solutions visant à mieux entreprendre l'identification , la modélisation et la gestion de la variabilité dans les LdP ,Ce chapitre complète notre système de gestion de la

variabilité SGV par une autre approche qui cherche à faciliter la représentation de la variabilité architecturale à travers un sous-modèle conceptuel basé sur des mécanismes du standard UML .Notre modèle s'inspire de la documentation OVM [59] et du processus VML[77] en les adaptant à l'aide de mécanismes UML pour permettre la composition de variants dans les modèles architecturaux .

6-2- Représentation de la variabilité architecturale

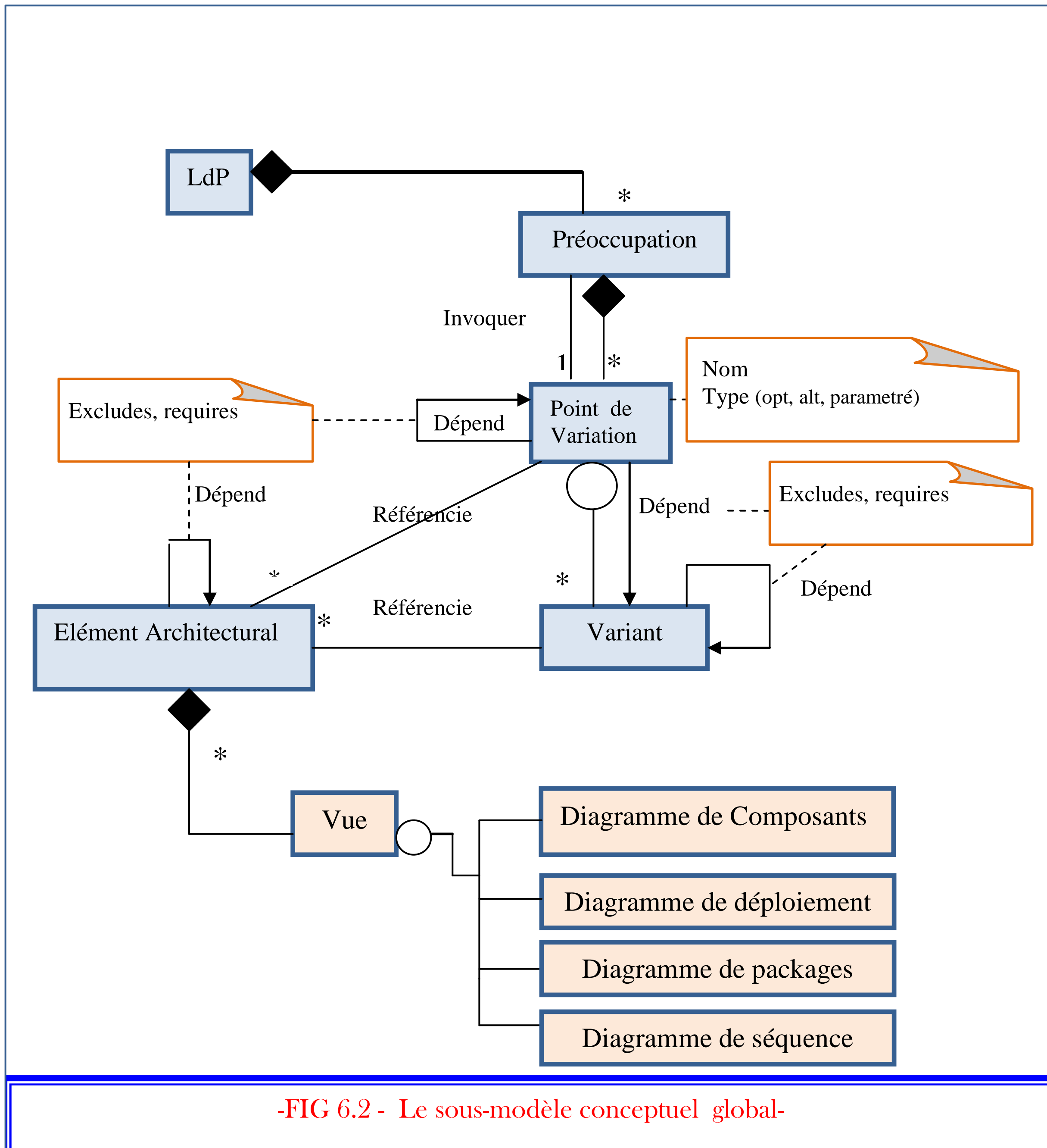
6-2-1- Le Processus global -

Cette section présente le processus général de notre sous-modèle relatif à la représentation de de la variabilité dans les architectures de LdP .Le principal but de ce sous-modèle est d'offrir un moyen de composer ensemble les éléments variables dans des modèles architecturaux – cf. figure 6.1 – .Pour cela , notre sous-modèle utilise un modèle conceptuel basé sur des mécanismes UML dont :



Les classes, les méthodes et les associations .Ces mécanismes lui permettent de référencer et invoquer des actions par lesquelles résulte la composition d'éléments architecturaux. Notre sous-modèle agit comme une vue spécifique du domaine pour la composition architecturale,

en permettant aux features d'invoquer des opérations qui composent ensemble des variables dans les éléments architecturaux (cf. Figure 6.2).



-FIG 6.2 - Le sous-modèle conceptuel global-

La configuration des variabilités est créée à partir du modèle de features par l'invocation des points de variation relatifs à des préoccupations spécifiques désignées par l'architecte du domaine.

Ainsi, les méthodes spécifiques contenues dans les classes préoccupations, les points de variation et les associations entre ces classes faciliteront la composition des éléments architecturaux variables dans les différentes vues.
L'approche évite le recours à de nouvelles notations ou extensions UML dans les descriptions d'architecture logicielle ainsi que le recours à un autre langage dont il faut connaitre la sémantique, les formalismes des primitives, les interactions entre elles etc…

6.2.2 – Les éléments du modèle.

On se focalisera uniquement sur la présentation des abstractions du sous-modèle. Avant d'illustrer les différents éléments conceptuels composant le sous-modèle, nous allons présenter les différents concepts qu'il utilise :

-L*'architecture d'une LdP* contient plusieurs préoccupations.

-*La préoccupation* peut être une feature définie dans le modèle de features, ou bien une préoccupation architecturale liée à la phase conception et relevant d'architectures de différents intervenants sur la LdP (concepteur , développeur , architecte système , …..).
-Chaque préoccupation a un nom et des points de variations (PVi) associés entre eux (PV1,PV2…….). Exemple d'une préoccupation P composée dans le diagramme de features du point de variation PV1 et du variant V21 de PV2 .Elle sera définie par *P (PV1, PV2(V21)).*

-*Un point de variation (PV)* a un nom et un type de variation. Les types de variation sont : *« optionnel », « alternative »* ou *« Paramétré »*. Un Point de Variation (PV) peut avoir un certain nombre de variants (donc jouer plusieurs roles): V1, V2 …Vn .Un point de variation doit référencer un ou plusieurs éléments architecturaux dans des vues architecturales à travers les méthodes et les associations du sous-modèle ou à travers ses variants.

-*Les Variants (V) ou variant-roles,* les variants doivent référencer *un ou plusieurs éléments architecturaux dans des vues architecturales* à l'aide des méthodes et associations du sous-modèle. Ces *éléments architecturaux dans différentes vues* représentent *les modèles architecturaux.*

-L*'(es) élément(s) architectural(aux)* correspond(ent) à un variant dans une vue .Ils expriment la variabilité du variant selon une vue spécifiée .Ils jouent des roles dans les vues.
Ils peuvent jouer par exemple le role d'un composant ou d'une interface dans la vue composant.

-Il peut y avoir *des dépendances* entre les Points de Variation et /ou les variants (Vi) .Elles sont de 3 types : *Point de Variation-Point de Variation , Point de Variation-Variant* et *Variant-Variant.*

6.2.3- Actions sur les éléments du sous-modèle.

La gestion de la variabilité architecturale est obtenue par l'invocation des différentes méthodes renfermées dans les classes du sous-modèle .Les principales méthodes ont trait aux actions de composition d'éléments (simples pour la granularité fine et complexes pour le gros grain) du sous-modèle entre eux ou de décomposition à partir de paquetages variables (gros grains) ou d'autres éléments variables (granularité fine) . Les actions les plus utilisées par les intervenants sur les LdP ont été prises en charge dans notre sous-modèle. Nous allons à présent décrire comment elles sont réalisées.

6.2.3.1 Compositions à granularité fine.

a- Aspect statique .

1- *La référence* : associe des éléments architecturaux à un même nom (ou référence) dans le but de simplifier la sollicitation de ce groupe d'éléments par les méthodes du modèle. Elle est modélisée par l'association *« référencer »* qui lie un variant aux éléments architecturaux concernés.

2- *L'invocation* : active soit des points de variation soit des variants au niveau de la préoccupation,des points de variation ou des variants ,. Elle est modélisée par les méthodes *'invoquer'* dans les classes *préoccupation, point de variation* et *variant – figures 6.3 et 6.4.*

3- *La Connexion* : permet de Connecter des éléments architecturaux entre eux dans une vue (ex : connecter deux composants à travers une interface dans la vue composants). Elle est modélisée par l'association n-aire *connecter* qui lie les 3 classes « *élément-architectural* », « *composant* » *et* « *interface* ».

4- *L'ajout* : permet d'ajouter ensemble un élément architectural à un autre élément architectural dans une vue .Elle est modélisée par la méthode *ajout()* au niveau de la classe « *élément architectural* » et au niveau de toutes les classes concernées (composant ; interface dans la vue composant et nœud dans la vue déploiement).

5- *La suppression* : contrôlée permet de supprimer un élément architectural de n'importe quelle vue. Exemple : supprimer un composant et son interface requise.
Elle est modélisée par la méthode *supprimer ()* dans les classes « *élément-architectural* » et toutes les autres classes concernées (*composant, interface* et *nœud*). Ces méthodes ne procèdent pas directement à la suppression des éléments demandés mais opèrent des contrôles au préalable sur la cohérence de la vue conceptuelle concernée en cas de suppression.

6- *Déployer* : Permet d'assigner des éléments architecturaux tels que des composants à un Nœud. Elle est modélisée par une association n-aire reliant les classes « *nœud* » ; « *composant* » *et* « *interface* ».

7- *Ouvrir un chemin de communication* : les variabilités architecturales peuvent nécessiter l'ouverture d'un chemin de communication à des périphériques càd la connexion des nœuds à des périphériques. Elle est modélisée par une association n-aire reliant les classes « *nœud* » *et* « *périphérique* ». Toutes ces compositions sont illustrées dans le cadre de l'aspect statique de notre modèle –en figure 6.4–

8- Permettre des actions spécifiques à la vue interaction (vue dynamique). Nous nous sommes focalisé sur la vue séquence dans notre modèle .Cette vue implique des séquences d'opérations entre des éléments architecturaux selon un ordre d'exécution précis. Exemple : Exécuter la Séquence 1 puis la séquence 2…..Ces interactions seront ajoutées dans le flot des séquences de la vue. Cette composition est prise en charge dans l'aspect dynamique du modèle -cf. figure 6.5-

b. Aspect dynamique.

Se traduit par des actions spécifiques à des vues dynamiques. Notre sous-modèle se limite à la vue interaction pour laquelle nous nous sommes focalisé sur la vue séquence .Cette vue implique des séquences d'opérations entre des éléments architecturaux selon un ordre d'exécution précis. Exemple : Exécuter la Séquence 1 puis la séquence 2…..Ces interactions seront ajoutées dans le flot des séquences de la vue. Cette composition est prise en charge dans l'aspect dynamique du modèle dans la Figure 6.5.

6.2.3.2 Compositions à gros grains.

Ce type de composition concerne la Fusion de *paquetages d'éléments architecturaux composés* avec *un paquetage commun selon la vue*. Ces paquetages sont des conteneurs d'éléments à multiples architectures .Exemple : Fusionner les éléments du paquetage

variable « *détection d’incendies* » au paquetage commun de la LdP « *maison intelligente* ». **L’association « *Merge* » d’UML [16] associant des paquetages entre eux permet cette fusion.**

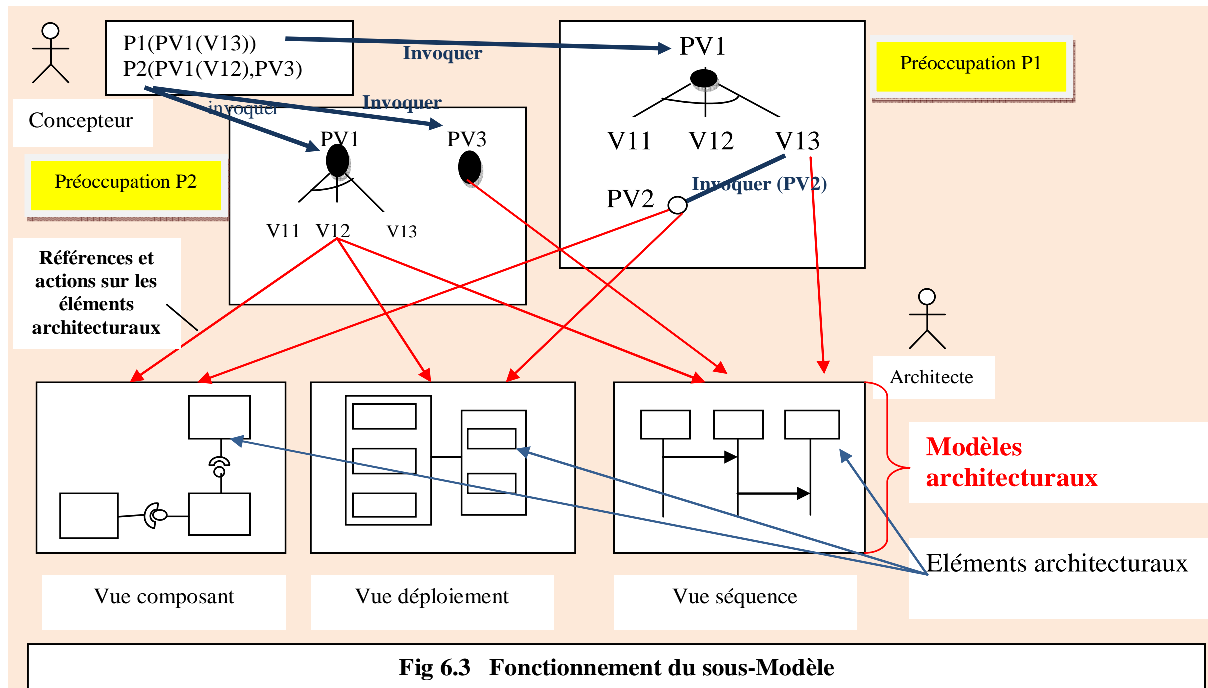


Fig 6.3 Fonctionnement du sous-Modèle

6.2.3.2 Compositions à gros grains.

Ce type de composition concerne la Fusion de *paquetages d’éléments architecturaux composés* avec *un paquetage commun selon la vue*. Ces paquetages sont des conteneurs d’éléments à multiples architectures .Exemple : Fusionner les éléments du paquetage variable « *détection d’incendies* » au paquetage commun de la LdP « *maison intelligente* ». **L’association « *Merge* » d’UML [16] associant des paquetages entre eux permet cette fusion.**

a- Aperçu sur la fusion de Packages par l’association « Merge » d’UML.

La fusion de packages définit comment les contenus d’un paquetage sont étendus par les contenus d’un autre paquetage. Ceci est réalisé dans UML par l’association Merge d’UML.

b-Description

Merge est une association directe entre deux packages qui indique que les contenus des deux packages sont fusionnés. Elle est similaire à la Généralisation dans le sens ou les caractéristiques conceptuelles du pacquage source sont ajoutées à celles du pacquage destination pour composer un pacquage résultat avec des caractéristiques combinées.
Merge est utilisé particulièrement dans les méta-modèles UML pour définir leurs extensions par d’autres pacquages. Elle peut être utilisée également au niveau des modèles de pacquages UML .Conceptuellement une fusion de packages peut être perçu comme une opération qui prend le contenu de deux pacquages et produit un nouveau pacquage qui combine les contenus des pacquages impliqués dans la fusion. En terme de semantique de modèle, il n y a pas de différence entre un modèle avec des fusions explicites de packages, et un modèle dans lequel toutes les fusions ont été opérées.

c-Sémantique Comme pour la généralisation, un package de fusion entre deux pacquages dans un modèle n’implique pas simplement des transformations dans le package résultat, mais aussi que ce résultat est inclut de lui-même dans le modèle.la figure 6.5 montre la vue sémantique d’un merge de deux packages A et B.

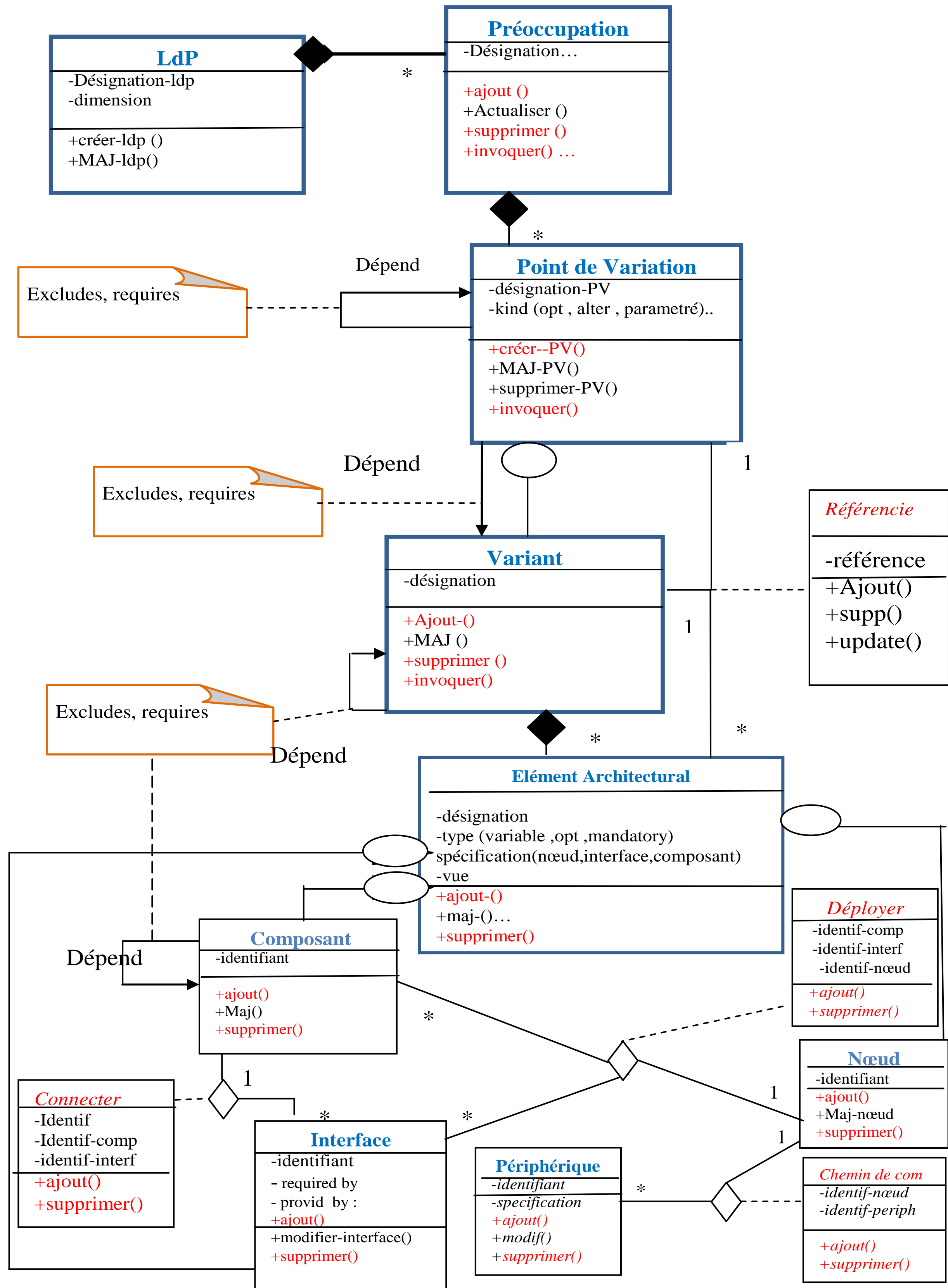
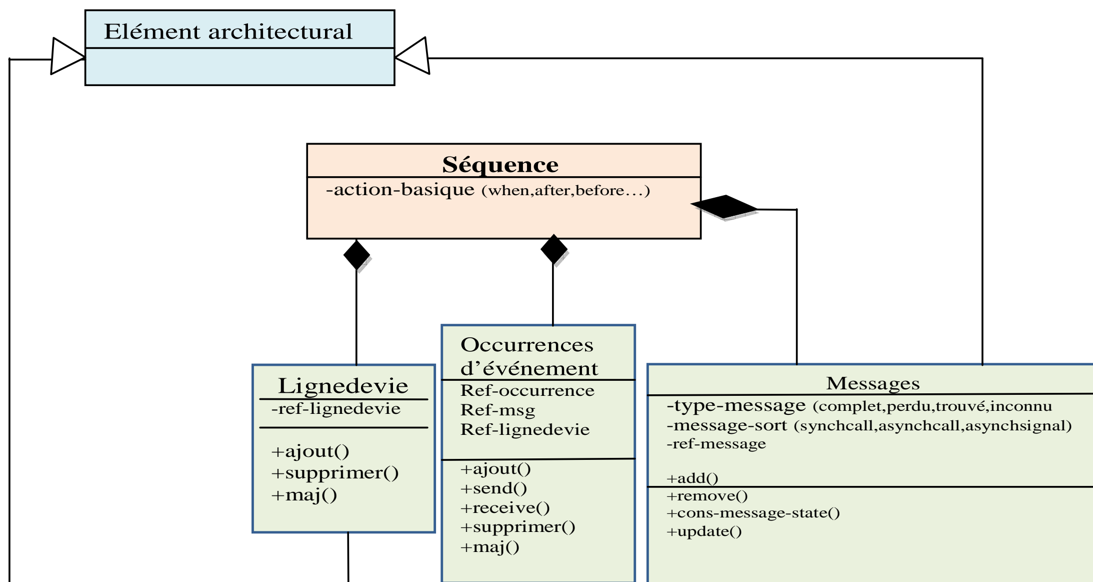


Fig 6.4 - **Aspect statique de notre sous-modele.**

- FIG 6.5 -
Aspect dynamique

Le système de gestion de la variabilité–La vue séquence



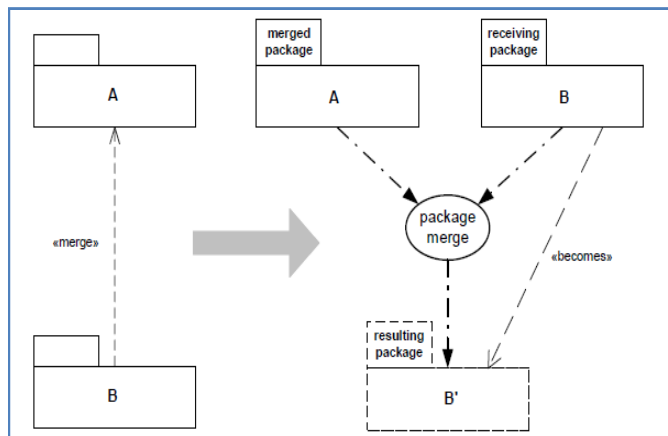


Fig 6.6-Vue Conceptuelle de la sémantique de la fusion de packages par l'association « merge » d'UML

d. Usage de l'association « Merge » dans notre sous-modèle.

Les paquetages fusionnés peuvent être crées par l'architecte à travers soit les compositions à granularité fine soit à partir de décompositions du paquetage commun selon la vue. L'idéal serait que chaque paquetage encapsule un variant (ou un point de variation optionnel) .Ceci permet une encapsulation un à un des variants dans les paquetages. Notre sous-modèle a juste besoin de se référer aux noms des paquetages puis d'utiliser l'association « *merge* » pour fusionner les paquetages de variants avec le paquetage commun selon la vue ou avec d'autres paquetages.

6.2.4 Les Dépendances :

Permettent la spécification des dépendances entre tous les éléments variables (Points de variation, Variants et éléments architecturaux) par l'usage des stéréotypes « excludes » et « requires ».Ce mécanisme est le même utilisé par UML pour gérer cet aspect de la variabilité.

6.3. Conclusion.

Ce chapitre a présenté un sous-modèle permettant aux formes hétérogènes de variabilités architecturales d'être spécifiée d'une manière qui permet de modulariser la définition de la variabilité dans une préoccupation architecturale en un seul endroit (la préoccupation) facilitant ainsi son changement et son évolution .Notre sous-modèle conceptuel basé sur des mécanismes UML fournit une représentation de la variabilité facilement compréhensible par les ingénieurs software et l'architecte de la LdP. Il facilite la représentation de la variabilité et sa composition en permettant l'ajout et la suppression d'éléments architecturaux individuels au modèles communs selon les vues .Il fournit un moyen de décrire la variabilité séparément lui permettant d'être traitée comme une nouvelle vue architecturale (càd un modèle de variabilité).

Deuxième partie
Contributions

Chapitre 7.

I

Etude du système maison intelligente.

7-1- Présentation du système de contrôle d'une maison intelligente.

Dans l'architecture d'une maison intelligente -cf. Figure 7.1-, nous voulons développer différents produits qui ont des compositions différentes de features. En retour, ces compostions vont provoquer des changements dans les éléments architecturaux pour des vues architecturales

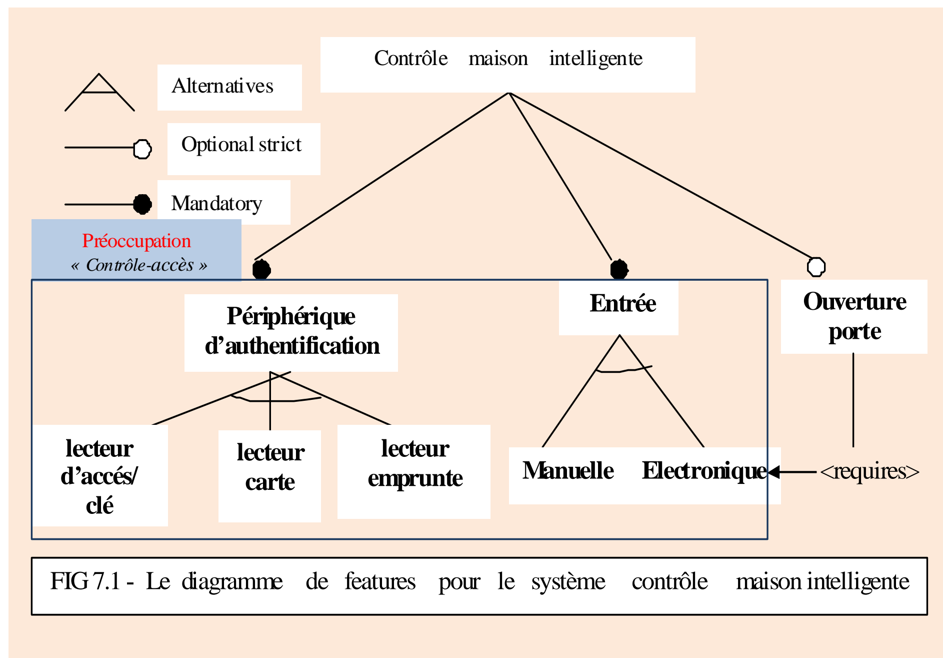


FIG 7.1 - Le diagramme de features pour le système contrôle maison intelligente

différentes. Pour comprendre les variabilités contenues dans ce système, on va prendre en considération le diagramme de features - cf. Figure 7.1- qui utilise la notation de [53].

Dans la Figure 7.1, ce diagramme illustre que contrôle-maison-intelligente peut être développé pour être utilisé avec des périphériques d'authentification différents, qui sont les lecteurs de clés, les lecteurs de cartes magnétiques et les lecteurs d'empruntes et fournit des alternatives de fermeture de la porte d'accès manuelle (clé physique) ou électronique (déverrouillage à l'aide d'un vérin et sur authentification).

La sélection de l'alternative manuelle nécessite que l'utilisateur soit authentifié une fois à l'intérieur de la maison. En plus, il y a une option pour l'ouverture de la porte d'accès à l'aide d'un vérin. Mais cette option n'est valable que si l'alternative électronique est choisie.

Le diagramme de composants est illustré dans la Figure 7.2. Comme il n y a pas une notation standard pour la variabilité qui permet de décrire les ensembles complets de variabilités dans plusieurs vues, nous avons utilisé une notation similaire à celle de Clauss [10]. Ainsi, Le diagramme de composants est utilisé beaucoup plus pour démontrer les problèmes de description des variabilités orthogonales selon les vues dans un contexte visuel .

Les composants : *gestionnaire authentification*, *contrôle-utilisation* et *contrôle-fermeture* représentent le cadre commun basique du diagramme de composants.

Les éléments variables introduits par composition à ce cadre commun par le concepteur ou un autre intervenant sur la LdP sont représentés en traits discontinus.

Le diagramme utilise les annotations <optional>, <alternative>et <requires> pour indiquer les types de variabilités et <requires> pour indiquer les dépendances entre les éléments architecturaux.

Un certain nombre d'éléments architecturaux correspondent directement aux features, mais d'autres ne le sont pas.

Exemple : la sélection de l'option *entrée électronique* va entrainer la connexion du composant *fermeture-électronique* aux composants *contrôle-fermeture* et *gestionnaire-Authentification*, et aussi le composant *vérin-fermeture* au composant *contrôle-fermeture.*

De plus, une nouvelle interface requise doit être ajoutée au composant *gestionnaire-Authentification* et donc une nouvelle interaction doit être créée entre ce composant et le composant *fermeture-électronique*. En plus, chaque périphérique d'authentification doit

requérir un algorithme d'authentification. Les composants *lecteur de clé* et *lecteur de cartes* vont requérir un composant Simple-Algorithme qui ne fait que confirmer l'information reçue par le périphérique périphérique-authentification .Le composant lecteur d'emprunte va requérir un algorithme plus complexe contenu dans le composant emprunte-Algorithme car ce dernier utilise une base biométrique.

En observant le diagramme de composants – Figure 7.2- (qui utilise la notation de [77]) correspondant au diagramme de features –figure 7.1–, on peut constater que même une description d'une simple feature peut entrainer un ensemble complexe de variabilités émergentes à travers la décomposition architecturale. Ainsi, on constate que pour construire le diagramme de composants correspondant à cette feature, il y a lieu de créer d'autres éléments architecturaux dans ce diagramme et donc *il n y a pas de correspondance biunivoque entre le diagramme de features et celui des composants* .C'est cet aspect de la représentation orthogonale des variabilités architecturales que nous allons illustrer à travers l'exemple du système de contrôle d'accès .Cette illustration prendra deux volets relatifs aux actions de compositions/décompositions à granularité fine et à gros grains de la variabilité: le premier, *descriptif* , le second correspondant au premier est *opératoire* .Ce dernier est basé sur l'application des concepts intronisés dans notre sous-modèle .

7-2 Description des compositions à granularité fine.

Par souci de brièveté et de clarté, nous allons seulement détailler la préoccupation « *Contrôle-accès* »illustrée dans la figure 7.1 qui renferme les points de variation : « *périphérique-authentification* » et « *entrée* ». *Elle sera formalisée par :*

Contrôle-accès **(périphérique-*authentification*** *(lecteur-accès-clé, lecteur-Carte, lecteur-emprunte)*,***Entrée*** *(Manuelle, électronique)).*

7.2.1 Le point de variation « *Périphérique authentification* »

a- Ce point de variation de type « alternative » fournit 3 alternatives de variants nommées *lecteur accès par clé, lecteur de Carte* et *lecteur d'emprunte*.
Deux références, AuthPériphériquePV et *AlgorithmePV*, sont crées dans le but de simplifier les sollicitations des éléments qu'ils référencient par les autres éléments de la LdP :

-*AuthPériphériquePV* référencie le composant « *Contrôle-fermeture* » et son interface requise « *IAuthPériphérique* ».

-et AlgorithmePV référencie le composant « *gestionnaire authentification* » et son interface requise « IAuthAlgorithme » .

Un certain nombre de compositions sont alors nécessaires pour connecter les composants puis les déployer dans leurs nœuds respectifs. Comme les mécanismes de prise en charge de ces compositions sont similaires pour tous les variants, nous allons seulement détailler le variant « *lecteur-clé-accès* ».

7.2.1.1 Le variant « *lecteur-clé-accès* ».

b- dans la 1ere action les composants référencés par *AuthPériphériquePV* sont connectés au composant « *Lecteur-clé-accès* » via l'interface fournie IAuthPériphérique.
c-*Dans* La seconde, les composants référencés par AlgorithmePV sont connectés via l'interface fournie « *IAuthAlgorithme* » au composant « *SimpleAlgorithme* » .
d-En dernier, les composants « *Lecteur-clé-accès* » et « *SimpleAlgorithme* » sont déployés respectivement dans les nœuds « *Périphérique-authentification* » et « *contrôle-maison-intelligente* ».

7.2.1.2 Les variants *« lecteur-carte» et « lecteur emprunte ».* Les mêmes compositions décrites au niveau du variant « lecteur-clé-accès » seront appliquées pour décrire les variabilités émergentes de ces deux variants.

- 7.2.2 Le point de variation « *Entrée* » .Il fournit un ensemble plus complexe et intéressant de compositions. Il détaille deux variants, nommés : *Manuel* et *Electronique.*

e- *Contrôle-fermeturePV* référencie le composant « *Contrôle-fermeture* » et son interface requise « *IFermeturePorte* ».

7.2.2.1 Le variant « *Manuel* » connecte les composants référencés par « contrôle-*fermeturePV*» au composant « *fermeture-manuelle* » via l'interface « *IFermeture-porte* », et déploie le composant « *fermeture-manuelle* » dans le nœud « *Contrôle-maison-intelligente* »

7.2.2.2 Le variant « *Electronique* » représente un ensemble de compositions simples incorporant la composition d'éléments à multiples architectures que nous allons décrire :

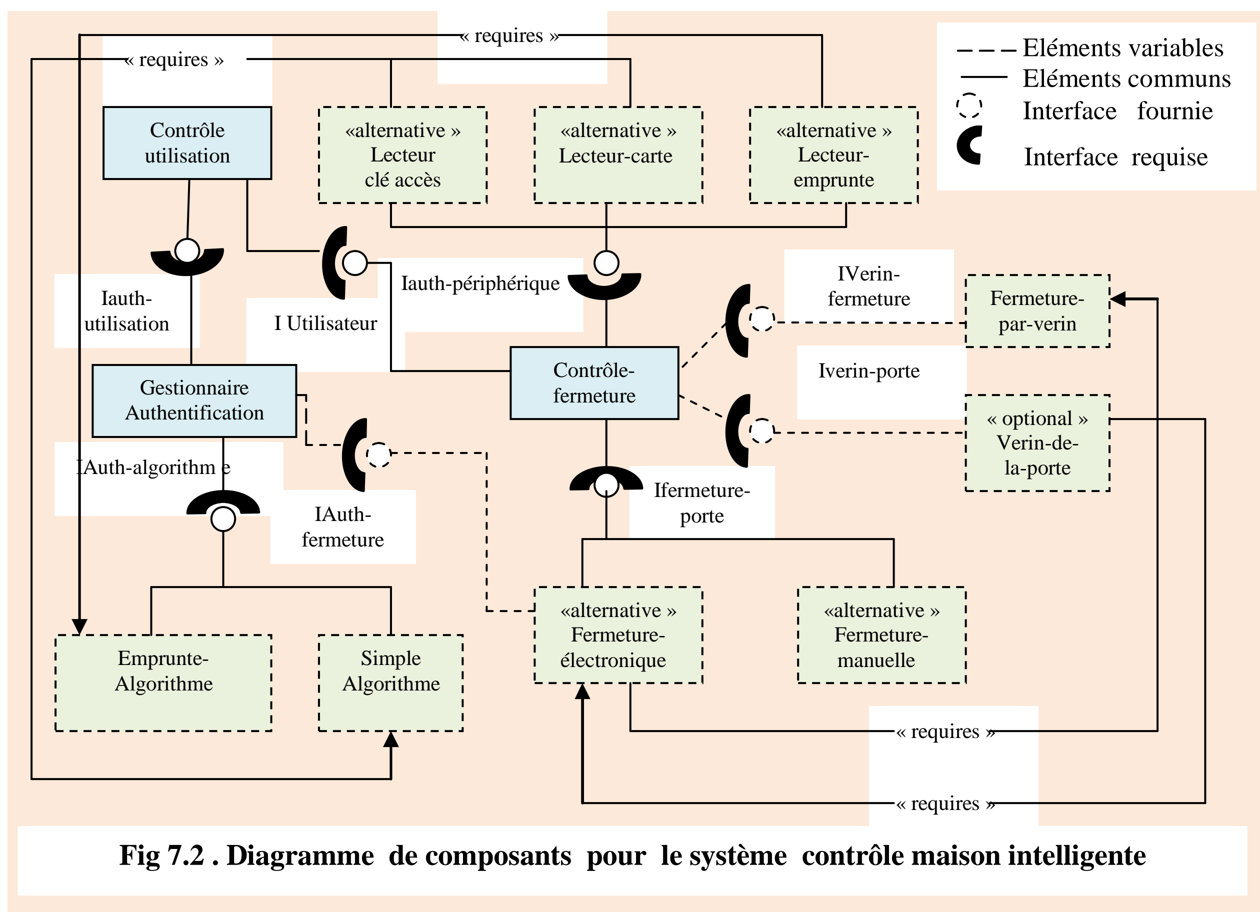


Fig 7.2 . Diagramme de composants pour le système contrôle maison intelligente

f- Les composants référencés par « contrôle-*fermeturePV*» sont connectés via l'interface *IFermeture-porte fournie par le* composant *Fermeture-électronique* .Ensuite, l'interface *IAuth-fermeture* est ajoutée aux composants *Gestionnaire-authentification* (comme requise) et Fermeture-électronique (comme fournie).

g- Par la suite, l'interface requise *Ivérin-Fermeture* est ajoutée au composant *Contrôle-fermeture* puis les composants *Contrôle-fermeture* et *Fermeture-par-vérin* sont connectés via cette interface (requise par *contrôle-fermeture* et fournie par *fermeture-par-vérin*).

Dans la suite (le point « h »), nous allons illustrer l'usage d'interactions de type séquence.

h-L'interaction déclare qu'après que l'opération *authentifier dans G.A (Gestionnaire-authentification)* ait été exécutée, un nouveau *message* doit être ajouté qui envoie *le résultat de l'opération de déverrouillage* appelée par le composant *Gestionnaire-authentification* au composant *Fermeture-électronique* via l'interface *IAuth.*
A la fin, les composants *Fermeture-électronique* et *Fermeture-par-vérin* sont déployés dans le nœud *contrôle-maison-intelligente.*

7.3 Prise en compte des compositions à granularité fine (de a à h).

a- Création d'une préoccupation appelée *'contrôle-accès'* par la méthode *Ajout ( )* dans la classe *préoccupation*.
-Ajout d'un Point de variation *'périphérique-authentification'* par la méthode *créer–PV( )*dans la classe « *Point de Variation* » ayant pour type 'Alternative' .
-Ajout de l'instance « *IAuthPériphérique » par la méthode ajout() dans la classe « Element architectural » .Cette instance jouera le role d'interface (classe interface type required) dans la vue composant .Elle hérite (association role) la classe « élément architectural ».*
-Ajout de l'instance « Contrôle-fermeture » *par la méthode ajout() dans la classe « Element architectural » .Cette instance jouera le role de composant (classe composant) dans la vue composant .Elle hérite (association role) la classe « élément architectural ».*
-Création d'une instance dans la classe association « référencie » identifiée *'Auth-PériphériquePV' qui va référencer* « Contrôle-fermeture » *et « IAuthPériphérique ».*
- Un processus similaire sera appliqué pour la référence 'AlgorithmePV'.

b- Ajout du variant de désignation *'lecteur-clé-accès'* par la méthode ' *Ajout()* ' dans la classe « *variant* ».
- Ajout d'une nouvelle instance dans la classe association ' *connecter* ' en activant la méthode *ajout()* de cette classe pour relier les éléments architecturaux référencés par '*Auth périphérique-PV'* au composant ' *Lecteur-clé-accès'* via l'élément *'IAuth périphérique'* instance de la classe *interface* .

c- Le même processus est appliqué pour connecter *'Simplealgorithme'* via *'IAuth Algorithme' au composant 'Gestionnaire Authentification'.*

d- Le déploiement du composant '*Lecteur-clé-accès*' au niveau du Nœud ' *Authentification-périphérique* ' est obtenu par l'association *'Déployer* '. Une nouvelle instance de la classe « *déployer* » sera créée, elle contiendra 'identif-Nœud' qui correspond à l'identifiant de ' *Authentification-périphérique* ' concaténé à identif-composant correspondant au composant ' *Lecteur-clé-accès'* dans la classe « *composant* ».
Le même processus de déploiement sera appliqué au composant *'Simple Algorithme' au niveau du* nœud *'contrôle-maison-intelligente'.*

e- Ajouter une nouvelle instance à la classe « *Point de variation* » à savoir *'Entrée'* de type alternative.

-Les mécanismes de référence, de création de variants, de déploiement et de connexion ont déjà été décrits pour le Point de variation «*Périphérique-Authentification* ».

f- On s'intéresse maintenant à l'ajout de l'interface requise « *IAuth-fermeture* » au composant '*gestionnaire-Authentification* '. Pour cela les actions suivantes sont nécessaires :
Créer une nouvelle interface par l'activation de la méthode *ajout ()* de la classe interface, *'IAuth-fermeture'* .Elle est requise par le composant '*gestionnaire-Authentification'*.

g- La Connexion du composant *'contrôle-fermeture '* au composant *'Fermeture-par-vérin'* via l'interface *'IVerin-fermeture'* sera réalisée par les compositions suivantes:

> i- Créer une nouvelle instance de la classe interface *'IVérin-fermeture'* par l'activation de la méthode *ajout()* de la classe « *interface* » : ses propriétés sont :Identifiant = 'IVérin-fermeture' , Requise par = ' contrôle-fermeture' et fournie par 'Fermeture-par-vérin'
> ii- Ajouter une nouvelle instance de la classe *association* 'connecter ' par l'opération *Ajout ()* avec comme Identif le résultat de la concaténation des Identifiants des composants *Contrôle-fermeture*, *fermeture-par-vérin* et l'identifiant de l'interface *IVerin-fermeture.*

h- Aspect dynamique.

Cet aspect est représenté par des actions spécifiques aux vues interaction. Notre modèle se limite à la vue séquence – cf. figure 7.3-. Le flot des séquences est le suivant :

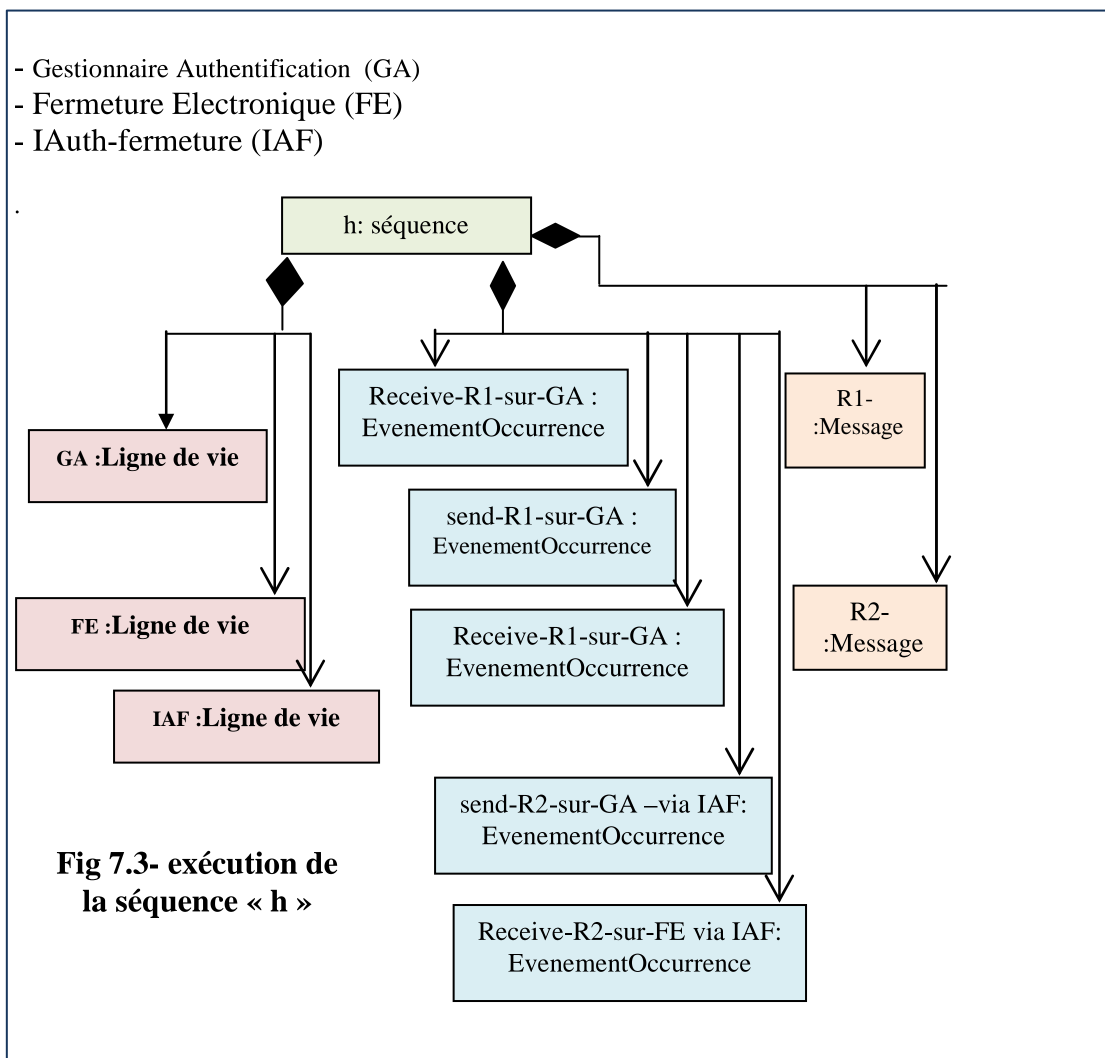


Fig 7.3- exécution de la séquence « h »

i- Il y a lieu en premier lieu d'ordonner le flot des actions à réaliser :

> 1/- Exécution de la méthode « *Authentifier* » du composant « *gestionnaire-authentification* »
> 2/- Activer la méthode « *déverrouiller* » du composant « *gestionnaire-Authentification* » .

3/- Envoyer le résultat de la méthode « *déverrouiller* » au composant « *fermeture-Électronique* ».
4/- Recevoir ce résultat par le composant « *fermeture-électronique* » via l'interface « *IAuth-fermeture* ».

ii-En second lieu, procéder à l'exécution des opérations ordonnées en i par :

1-la création de 3 instances dans la classe « *ligne de vie* » - par la méthode *ajout()* -:
- *i- gestionnaire-Authentification*
- *ii- fermeture-Electronique et*
- *iii- IAuth-fermeture.*

2/-Création de 2 instances dans la classe « *message* » - par la méthode *ajout():* Résultat de la méthode « *authentifier* » = R1 et Résultat de la méthode « *déverrouiller* » = R2 .

3/-Création de 2 instances dans la classe « *occurrences d'événements* » par la méthode *ajout() :*
- i)- Send R1 à partir de la ligne de vie « *gestionnaire-Authentification* » vers la ligne de vie « *gestionnaire-Authentification* ».
- ii)- Send R2 de la ligne de vie « *gestionnaire-Authentification* » vers « *Fermeture-électronique* » via « *IAuth-fermeture* »
- iii)- Receive R2 par « *fermeture-Electronique* ».

7-4- Description et Prise en compte des compositions à gros grains.

Cette section décrit comment le système contrôle-fermeture peut être décrit en utilisant un mécanisme de composition à gros grains pour la fusion de paquetages – les mécanismes de modélisation sont illustrés dans la figure 7.4 :

-

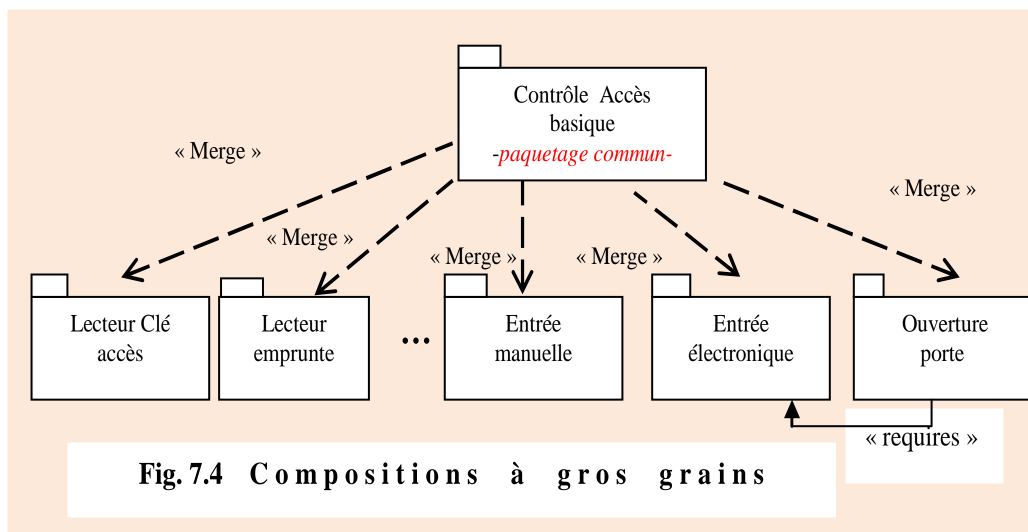


Fig. 7.4 Compositions à gros grains

Création d'une préoccupation appelée '*contrôle-accès*' par la méthode *Ajout ( )* dans la classe « *préoccupation* ».Elle est définie par ***Contrôle-accès*** (**périphérique-*authentification*** *(lecteur-accès-clé, lecteur-Carte, lecteur-emprunte)*, ***Entrée*** *(Manuelle, électronique,* ***Ouverture-porte*** *)).*

-Ajout d'un Point de variation '*périphérique-authentification*' par la méthode *créer–PV( )*dans la classe *Point de Variation* ayant pour type ='Alternative' .

-Ajout des autres Points de variation *Entrée* par la méthode *créer–PV( )* dans la classe *Point de Variation* ayant pour type « Alternative » .

-Ajout du point de variation *'Ouverture-porte'* 'qui dépend du variant « Entrée électronique » par la méthode ' *Ajout()* ' dans la classe Point de *variation .Il est de type « option » (ou option stricte).*
-Ajout du variant de désignation *'lecteur-clé-accès'* par la méthode ' *Ajout()* ' dans la classe *variant.*
-Ajout des autres variants *lecteur-Carte, lecteur-emprunte ,entrée-Manuelle, Entrée-électronique illustrés dans la figure 7.4 par* la méthode ' *Ajout()* ' dans la classe *variant.*
-Associer par l'usage de l'association *« Merge »* le paquetage du variant *« lecteur-clé-accès »* dans le paquetage commun *« contrôle-accès-basique ».…..*

-Associer par l'usage de l'association « Merge » le paquetage du variant *« Entrée* électronique» dans le paquetage commun « *contrôle-accès-basique ».*

-Associer par l'usage de l'association « Merge » le paquetage du variant *« Entrée* manuelle» dans le paquetage commun « *contrôle-accès-basique ».…*

Le même procédé sera appliqué aux autres variants de la figure 7.4.

Remarquons que chaque paquetage encapsule un point de variation sans variants ou un variant qui contient un ensemble d'éléments architecturaux selon les vues.

Cette association a permis une encapsulation un à un des variants dans les paquetages. Notre sous-modèle a juste besoin de se référer aux noms des paquetages (lecteur clé accès, lecteur emprunte ……) puis d'utiliser l'association *Merge* d'UML pour fusionner les paquetages de variants ou de points de variation sans variants avec le paquetage commun selon la vue ou avec d'autres paquetages.

Troisième partie

Conclusion et perspectives

Conclusion et Perspectives

Dans ce travail, nous avons proposé des mécanismes de spécification de la variabilité à un niveau plus abstrait qui est celui des modèles en s'appuyant sur le standard de la notation orientée objet UML.

En raffinant davantage l'aspect optionnel de la variabilité dans toutes les vues et en se concentrant sur le comportement des attributs et des méthodes vis-à-vis des instances des classes tout en complétant le niveau d'informations de la classe par un nouveau concept qui renseigne sur la population de la classe , nous avons proposé un modèle qui représente mieux la variabilité dans les lignes de produits à travers des extensions à UML sous forme de stéréotypes. Nous avons proposé également un contrôle de l'évolution de la variabilité des points variants à travers son expression par les roles associés à des machines à état-transitions, une pour chaque variant évolutif. Pour cela nous avons introduit une nouvelle association désignée « *role* » qui lie les variant-roles aux classes représentant les points de variation dans le diagramme de classe .De nouveaux roles peuvent être joués et donc peuvent être ajoutés aux classes rôles de la classe variable .De plus, l'évolution des rôles exprime celle de la variabilité au niveau des points de variation .Le concepteur peut en contrôler l'évolution et les dépendances à l'aide d'un autre diagramme UML , les machines à état-transitions .

Une autre contribution de modèle permet aux formes hétérogènes de variabilités architecturales d'être spécifiées d'une manière non invasive. Le but étant de pouvoir modulariser la définition de la variabilité en un seul endroit : la préoccupation architecturale, facilitant ainsi son changement et son évolution.

Notre modèle dans sa globalité basé sur des mécanismes UML fournit une représentation de la variabilité facilement compréhensible par les concepteurs et les architectes de la LdP. Il facilite la représentation de la variabilité et sa composition en permettant l'ajout et la suppression d'éléments architecturaux individuels au modèles communs selon les vues .Il fournit un moyen de décrire la variabilité séparément lui permettant d'être traitée comme une nouvelle vue architecturale (tel un modèle de variabilité). Deux illustrations avec des exemples significatifs ont été abordées dans notre travail pour montrer les points forts de nos contributions. L'un prenant en compte les trois premières contributions, concerne le système automobile, l'autre illustrant la dernière contribution concerne le système de contrôle d'accès à une maison intelligente.

L'intégration de notre modèle à UML est proposé graduellement par l'intégration des stéréotypes puis par l'intégration de l'association role et du diagramme des états transition correspondants .Nous avons pour terminer regroupé ces mécanismes dans un profil UML à deux aspects statique et dynamique que nous avons désigné profil LdP.

Notre approche dans sa contribution relative à la variabilité orthogonale s'est concentrée sur la modélisation et la gestion des éléments architecturaux selon un nombre limité de vues seulement. Dans le futur, nous comptons aborder d'autres vues, notamment des vues dynamiques. De plus, la recherche d'une simplification du diagramme de rôles proposé pour exprimer la variabilité et son évolution par les rôles pourront également faire l'objet d'un futur travail.