

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Saad Dahleb Blida 1



Faculté des Sciences

Département d'Informatique

Mémoire de fin d'études en vue de l'obtention du

Diplôme de Master en informatique

Option : Ingénierie des Logicielles & Systèmes Informatiques et Réseaux

Thème :

Conception et développement d'un Editeur Graphique pour l'aide à la spécification des systèmes auto-adaptatifs

Réalisé par:

- AOULA Halima
- CHEKNOUN Sara

Devant le Jury compose de :

- **Examineur** Mme. I. CHIKHI
- **Président** Mr. N.CHKHI
- **Promotrice** Mme. D. GUESSOUM
- **Encadreur** Mme. N. LAHIANI

Année universitaire 2020/2021

ملخص

تغيّر الانظمة ذاتية التكيف هيكلها و سلوكها اعتمادا على التغيرات الموجودة في بيئتها، و مع ذلك هناك صعوبة كبيرة في تحديد مواصفاتها خاصة فيما يتعلق بنوعي التكيف : الهيكلية و السلوكية.

في هذا المشروع، قمنا بطرح مقارنة جديدة لمواصفات الأنظمة ذاتية التكيف لتغطية نوعين من التغيرات التي فرضناها والتي تتمثل في التغيرات على مستوى هيكل النظام و التغيرات على مستوى الإعدادات.

للقيام بذلك، اقترحنا نموذجًا يعتمد على المقارنة الموجهة للمكون مع دمج مفاهيم التغيير الموجودة في مجال خطوط منتجات البرمجيات.

بناءً على النموذج المقترح، قمنا بتطوير محرر رسومات لإنشاء نموذج مواصفات. سيتم تحويل هذا النموذج تلقائيًا إلى كود أطلقنا عليه اسم XASADL ، يتم تنفيذ هذا التحول بفضل تقنيات تحويل النموذج التي اعتمدها في عملنا. يتم فحص كود XASADL الذي حصلنا عليه عن طريق خوارزميتنا من أجل إجراء التحقق الهيكلية لنموذج مواصفات النظام.

الكلمات المفتاحية: نظام متكيف ذاتيًا، تغيير، XASADL، تحويل النموذج، AQL، Sirius

ABSTRACT

Self-adaptive systems change their structure and behavior according to variability existing in their environment. However, there is a great difficulty in specifying them, especially the specification of the two types of adaptation: structural adaptation and behavioral adaptation.

In this project, we have proposed a new approach for the specification of self-adaptive systems in order to cover two types of variability that we have defined: the variability at architecture level and the variability at the configuration level.

To do this, we proposed a metamodel based on the component-oriented approach by integrating concepts of variability found in the field of software product lines.

Based on this metamodel, we have developed a graphical editor in order to make a specification model. This model will be automatically converted into a code that we have named XASADL, this transformation is achieved through the model transformation techniques that we have adopted in our work. The resulting XASADL code will be reviewed by our algorithm to perform structural verification of the resulting specification model of a system.

Key words: Software adaptive system, variability, XASADL, transformation of model, Sirius, AQL.

RESUME

Les systèmes auto-adaptatifs changent leur structure et leur comportement en fonction de la variabilité existante dans leur environnement. Cependant, il y a une grande difficulté dans leur spécification, surtout en ce qui concerne la spécification des deux types d'adaptation : l'adaptation structurelle et l'adaptation comportementale.

Dans ce projet, nous avons proposé une nouvelle approche de spécification des systèmes auto-adaptatifs pour couvrir deux types de variabilités que nous avons défini, il s'agit de la variabilité au niveau de l'architecture et la variabilité au niveau de la configuration.

Pour se faire, nous avons proposé un métamodèle basé sur l'approche orientée composants en intégrant des concepts de variabilité trouvés dans le domaine des lignes de produits logiciels.

A la base de ce métamodèle nous avons développé un éditeur graphique pour réaliser un modèle de spécification. Ce modèle sera automatiquement transformé en un code que nous avons nommé XASADL, cette transformation est réalisée grâce aux techniques de transformation de modèles que nous avons adopté dans notre travail. Le code XASADL obtenu sera examiné par notre algorithme afin de faire une vérification structurelle du modèle de spécification d'un système.

Mots clés : Système auto-adaptatif, variabilité, XASADL, transformation de modèle, Sirius, AQL.

Remerciement

Nous remercions en premier lieu ALLAH le tout Puissant qui nous a inspiré le courage, la force et la santé pour réaliser ce cursus et au final ce modeste travail.

Nous remercions en deuxième lieu notre directeur de recherche Mme GUESSOUM Dalila pour la confiance qu'elle accordé en nous proposant ce thème de recherche, mais surtout pour son accompagnement, soutien, écoute et disponibilité, et pour la façon dont la quelle elle nous a guidé pendant la réalisation de notre travail.

Nous tenons à gratifier aussi les membres de jury pour l'intérêt qu'ils ont porté à notre projet en acceptant d'examiner notre travail.

Ainsi nous tenons à remercier tout l'ensemble des enseignants pour ce qu'ils ont apporté à notre carrière d'étudiants universitaire pendant 5 ans.

Dédicace

A mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études,

A mes chères sœurs et mon cher frère pour leurs encouragements permanents, et leur soutien moral,

A toute ma famille et mes amies pour leur soutien, et a tous les gens qui m'aiment

Je dédie ce modeste projet.

Merci d'être toujours là pour moi.

CH.Sara

Dédicace

*A mes chers parents, pour tous leurs sacrifices, leur amour, leur
Tendresse, leur soutien et leurs prières tout au long de mes études,*

*A ma chère petite sœur pour ses encouragements permanents, et son
soutien moral,*

A mes chers frères pour leur appui et leur encouragement,

*A toute ma famille pour leur soutien tout au long de mon parcours
universitaire,*

A Tout mes proches et mes chers amis

*Que ce travail soit l'accomplissement de vos vœux tant allégués, et le
fruit de votre soutien infailible,*

Merci d'être toujours là pour moi.

A.Halima

TABLE DES MATIÈRES

RESUME	3
Remerciement.....	4
TABLE DES MATIÈRES.....	7
LISTE DES TABLEAUX.....	12
LISTE DES FIGURES	15
LISTE DES ACRONYMES.....	17
INTRODUCTION GENERALE	17
GÉNÉRALITÉ SUR LES SYSTEMES AUTO-ADAPTATIFS	19
1. Introduction	19
2. Définition des systèmes auto-adaptatifs	19
3. Les capacités des systèmes auto-adaptatifs.....	20
4. Propriétés des systèmes auto-adaptatifs.....	20
4.1. Auto-configuration	21
4.2. Auto-optimisation	21
4.3. Auto-réparation	21
4.4. Auto-protection	22
5. Classification des systèmes auto-adaptatifs	22
5.1. Classification selon la distribution	22
5.1.1. Mappage des nœuds adaptatifs.....	22
5.1.2. Collaboration adaptative.....	23
5.1.3. Fonctionnalité adaptative	23
5.2. Classification selon le degré d'adaptation	23
5.2.1. Adaptation fermée	23
5.2.2. Adaptation ouverte	23
5.3. Classification selon l'utilisation de modèle.....	23
5.3.1. Adaptation sans modèle.....	23

5.3.2. Adaptation basée sur un modèle.....	23
5.4. Classification selon le domaine d'utilisation	24
5.4.1. Adaptation spécifique.....	24
5.4.2. Adaptation générique	24
5.5. Classification selon le type	24
5.5.1. Type 1.....	24
5.5.2. Type 2.....	24
5.5.3. Type 3.....	25
5.5.4. Type 4.....	25
6. Modèle conceptuel.....	25
6.1. Système géré.....	26
6.2. Moteur d'adaptation.....	27
6.3. Environnement.....	27
7. Boucle d'adaptation	27
7.1. Surveillance (Monitor)	28
7.2. Analyseur (Analyze)	28
7.3. Plan.....	29
7.4. Exécuteur (Execute).....	29
7.5. Base de Connaissances (Knowledge).....	29
8. Comparaison de quelques approches sur les systèmes auto-adaptatifs	29
8.1. Approches basées sur l'architecture	30
8.2. Approches basées sur les composants	30
8.3. Approches basées sur les modèles	31
8.4. Comparaison	31
8.5. Discussion	34
9. Conclusion.....	34
LES MODELES DE SPECIFICATION DE LA VARIABILITE DANS SPL	35
1. Introduction.....	35
2. Ligne de produits logiciels.....	35
3. Variabilité	37
3.1. Définition	37

3.2. Classification de la variabilité.....	37
3.2.1. Classification selon le niveau de l'exigence.....	37
3.2.1.1. La variabilité essentielle.....	37
3.2.1.2. La variabilité technique.....	37
3.2.2. Classification basée sur les dimensions de la variabilité.....	37
3.2.2.1. La variabilité dans le temps.....	37
3.2.2.2. La variabilité dans l'espace.....	37
3.2.3. Classification selon le niveau de la visibilité.....	37
3.2.3.1. La variabilité externe.....	37
3.2.3.2. La variabilité interne.....	38
4. Les modèles de spécification de la variabilité.....	38
4.1. FM (Feature Model).....	38
4.2 OVM (Orthogonal Variability Model).....	38
4.3. CVL (Common Variability Language).....	38
4.4. DM (Décision Model).....	38
5. Feature Model.....	38
5.1. Feature Models basiques.....	38
5.1.1. Approche FODA (Feature Oriented Domain Analysis).....	39
5.2. Feature Model avec cardinalités.....	40
5.2.1. Approche CBFM (Cardinality-Based Feature Model).....	41
6. Conclusion.....	41
NOTRE APPROCHE DE LA SPECIFICATION DES SYSTEMES AUTO-ADAPTATIFS.....	42
1. Introduction.....	42
2. Variabilité trouvée dans les systèmes auto-adaptatifs.....	42
2.1. Variabilité au niveau d'architecture.....	45
2.1.1. Composant.....	45
2.1.2. Port.....	46
2.1.3. Connecteur.....	47
2.2. Variabilité au niveau de la configuration.....	48
3. Représentation graphique et textuel des concepts proposés.....	49
3.1. Représentation textuelle.....	49

3.2. Représentation graphique	50
3.3. Transformation de modèle graphique au textuel	53
3.4. Le code généré XASADL	54
4. Vérification Structurelle	56
4.1. Vérification automatique	56
4.1.1. Les actions non-autorisées.....	56
4.1.2. Les actions autorisées	57
4.2. Vérification Programmée	57
5. Diagramme de cas d'utilisation	58
5.1. Description des différents Cas d'Utilisation	59
6. Conclusion	60
L'IMPLEMENTATION DE L'EDITEUR GRAPHIQUE	61
1. Introduction	61
2. L'environnement et outils de travail	61
2.1. L'éditeur de logiciels Obeo	61
2.2. Sirius	61
2.3. Acceleo	62
2.4. Eclipse IDE	62
.2.5 Langage d'implémentation JAVA	62
2.6. XML	63
2.7. JDOM	63
.2.8 Langage AQL	63
3. Implémentation	63
3.1. Création d'un projet EMF	63
3.1.1. Création Le model de domaine (*.ecore)	64
3.1.2. Le model de génération (*.genmodel).....	65
.3.1.3 Representation Data (*.aird).....	66
3.2. Implémentation de l'éditeur	67
3.2.1. Modèles de spécification de point de vue (*.odesign)	67
3.2.2. Modeling Project	67
4. Présentation de l'éditeur graphique	68

5. La transformation du modèle	70
5.1. Les règles de transformation	70
5.2. Le code AQL	70
6. Le résultat du code java.....	73
7. Exemples illustratifs.....	74
7.1. Système de calcul simple	74
7.1.1. Spécification du système	74
7.1.2. Fichier XASADL obtenu :	79
7.1.3. La vérification programmée :	80
7.2. Cours en ligne.....	82
7.2.1. Spécification du système	82
7.2.2. Le fichier XASADL obtenu	84
7.2.3. Vérification de la spécification du système.....	84
8. Temps d'exécution	86
9. Conclusion.....	88
CONCLUSION	89
BIBLIOGRAPHIE.....	91

LISTE DES TABLEAUX

Table 1. 1 : Comparaison entre quelques approches qui ont essayé de modéliser les systèmes auto-adaptatifs	33
Table 3.1 : Représentation graphique des concepts	53
Table 3.2 : Approches des transformations de modèles[40].....	53
Table 4.1 : Les règles de transformation pour transformer le modèle après une spécification à un fichier XASADL.	70
Table 4.2 : Temps d'exécution pour Composites.....	86

LISTE DES FIGURES

Figure 1.1 : Propriétés des systèmes auto-adaptatifs [8]	21
Figure 1.2 : Classification des systèmes auto-adaptatifs	22
Figure 1.3 : Modèle conceptuel pour les systèmes auto-adaptative [11].....	26
Figure 1.4 : MAPE-K Loop[12]	28
Figure 1.5 : Quelques approches sur les systèmes auto-adaptatifs[11]	30
Figure 2.1 : Niveaux d'ingénierie de LdP[20].....	36
Figure 2.2 : Les éléments du diagramme de fonctions de base [30, 31].....	39
Figure 2.3 : Exemple de Feature Model FODA[30]	40
Figure 2.4 : La notation de Feature Model avec cardinalités [33, 34]	40
Figure 2.5 : Exemple d'extrait de Feature Model basé sur la cardinalité (CBFM) [35].	41
Figure 3.1 : Plan de conception notre approche.....	42
Figure 3.2 : Métamodèle proposé	44
Figure 3.3 : La partie ' composants ' du métamodèle	45
Figure 3.4 : La partie ' ports ' du métamodèle	47
Figure 3.5 : La partie ' connecteurs ' du métamodèle	47
Figure 3.6 : Diagramme de cas d'utilisation.	59
Figure 4.1 : Le modèle de domaine de l'éditeur graphique.....	64
Figure 4.2 : Le modèle de génération de l'éditeur.....	65
Figure 4.3 : Le modèle Representation Data	66
Figure 4.4 : Le fichier Odesign qui représente le plan de travail de la modélisation	67
Figure 4.5 : La création d'un projet Modeling Project	68
Figure 4.6 : Vue d'ensemble de l'éditeur.....	69
Figure 4.7 : Le Template de base proposée pour générer un code XML.....	72
Figure 4.8 : Les messages d'erreurs	73
Figure 4.9 : Les messages de validation	74
Figure 4.10 : Présentation de spécification 'Système du calcul simple'	74
Figure 4.11 : Présentation du Composite 'Lecture'	75
Figure 4.12 : Présentation du XORComposite 'Operation'	76
Figure 4.13 : Présentation du 1er Sous-Composite.....	76

Figure 4.14 : Présentation du 2ème Sous-Composite	77
Figure 4.15 : Présentation du ORComposite 'Affichage'	78
Figure 4.16 : Le fichier XASADL du 'système du calcul simple'	79
Figure 4.17 : Représentation des erreurs de spécification du 'système de calcul simple' ...	81
Figure 4.18 : Spécification du Système 'Cours en ligne'	83
Figure 4.19 : Le fichier XASADL généré de Système 'Cours en ligne'	84
Figure 4.20 : Représentation des erreurs de spécification du système 'Cours en ligne'	85
Figure 4.21 : Test de temps d'exécution.....	86

LISTE DES ACRONYMES

ADL : Architecture Dynamic Language
AQL : Acceleo Query Language
CBFM : Cardinality-Based Feature Model
CVL : Common Variability Language
DM : Decision Model
DOM : Document Object Model
DSL : Domain-Specific language
DTD : Document Type Déclaration
E : Environnement
EMF : Eclipse Modeling Framework
FM : Feature model
FODA : Feature-Oriented Domain Analysis
GMF : Graphic Modeling Framework
IBM : International Business Machines
IDE : Integrated Development Environment
JDOM : Java Document Object Model
JVM : Java Virtual Machine
LdP : Ligne de produits
LdP : Ligne de produits
M2M : Model To Model
M2T : Model To Text
MA : Moteur d'Adaptation
MAPE-K : Monitor, Analyze, Plan, Environment, Knowledge
MDA : Model Driven Architecture
MOF : Models to Text Transformation Language
OMG : Object Management Group
OVM : Orthogonal Variability Model
RSS : Really Simple Syndication
SAA : Système Auto-Adaptatif
SAX : Simple API for XM

SG : Système Géré

SGML : Standard Generalized Markup Language

SPLE : Software Product Line Engineering, : Ingénierie de la ligne de produits logiciels

SVG : Scalable Vector Graphics

x3ADL : eXtensible Architecture, Aspect and Action Description Language

XASADL : eXtensible Adaptive System Architecture Dynamic Language

XHTML : Extensible HyperText Markup Language

XML : eXtensible Markup Language

XSLT : eXtensible Stylesheet Language Transformations

INTRODUCTION GENERALE

De nos jours, les systèmes logiciels sont caractérisés par des changements fréquents de leur environnement dans lequel ils s'exécutent. Par conséquent, ces systèmes doivent changer leurs fonctionnalités et/ou leur configuration afin de suivre ces changements pour rester toujours fonctionnels, on parle alors des systèmes auto-adaptatifs. En effet, les systèmes auto-adaptatifs sont devenus un besoin et une réalité inévitable.

Cependant, le caractère évolutif des systèmes auto-adaptatifs introduit des difficultés supplémentaires concernant leur spécification.

Problématique

En effet, le concept d'auto-adaptation dans les architectures logicielles introduit des difficultés supplémentaires concernant sa spécification. Toutefois, il existe une approche prometteuse consiste à utiliser des modèles architecturaux à base de composants pour le développement des systèmes auto-adaptatifs, cette approche a l'avantage de pouvoir être réassemblées et adaptées pour différentes raisons par l'ajout, la suppression, le remplacement de composants ou par la reconfiguration d'assemblages de composant sans ajouter ou de retirer des composants. Cependant, cette approche doit faire face à un certain nombre de défis. Parmi les problèmes notoires des modèles de composants actuels, on peut souligner la difficulté à spécifier les propriétés non-fonctionnelle qui peuvent agir sur le fonctionnement ordinaire des systèmes. Au niveau système auto-adaptatifs, les propriétés non-fonctionnelles sont en relation avec la variabilité au niveau des composants.

Un autre défi de conception de systèmes auto-adaptatifs concerne le manque de langage et d'outils qui intègrent les concepts liés à ce type de systèmes. En outre, les approches existantes sont généralement spécifiques à un domaine et manquent de généralité.

Objectif

L'objectif dans ce projet de fin d'étude est de décrire et de concevoir, en premier lieu un métamodèle décrivant les concepts relatifs à la spécification des systèmes auto-adaptatifs.

L'approche proposée devra être générique, ce qui permettra de prendre en considération les deux types d'auto-adaptation : structurelles et comportemental. L'idée est d'utiliser un métamodèle combinant entre l'approche à composants et d'intégrer les concepts trouvés dans les modèles de variabilité dans le domaine des lignes de produits logiciels. Sur la base de ce métamodèle il est demandé en second lieu de concevoir une interface graphique permettant la modélisation graphique du modèle proposé. Cet outil sera développé avec Sirius sous Obeo Designer qui va permettre de générer des éditeurs graphiques, ainsi de faire une vérification structurelle du modèle de spécification obtenu.

Organisation du mémoire :

Notre mémoire est organisé en quatre chapitres :

Le premier chapitre : *Etat de l'art sur les systèmes auto-adaptatifs* : nous allons présenter dans ce chapitre les éléments et les concepts de base des systèmes auto-adaptatifs.

Le deuxième chapitre *Etat de l'art sur les modèles de spécification de la variabilité dans les SPLs* : ce chapitre pour objectif de présenter les modèles de spécification des systèmes auto-adaptatifs notamment feature model.

Le troisième chapitre *Spécification du métamodèle des systèmes auto-adaptatifs* : nous allons présenter les concepts et les éléments essentiels rentrant dans la spécification du métamodèle de notre approche.

Le quatrième chapitre *L'implémentation de l'éditeur graphique* : nous allons présenter dans ce chapitre les éléments de réalisation notre éditeur et vérification de la spécification ainsi de tester notre algorithme de vérification sur plusieurs spécifications des systèmes auto-adaptatifs.

Nous terminons le mémoire par une conclusion générale et des perspectives.

CHAPITRE 1

GÉNÉRALITÉ SUR LES SYSTEMES AUTO-ADAPTATIFS

1. Introduction

De nos jours, les systèmes modernes sont devenus de plus en plus complexes et opèrent dans des environnements, des ressources et des exigences changeants. L'objectif de leur développement nécessite à doter ces systèmes d'un mécanisme d'auto-adaptation leur permettant de faire face aux changements provenant de différentes sources. Par conséquent, ces systèmes doivent changer leurs fonctionnalités et/ou leur configuration afin de suivre ces changements pour rester toujours fonctionnels, cette fonctionnalité rend les systèmes auto-adaptatifs. Ce chapitre sera consacré pour définir le contexte général des systèmes auto-adaptatifs. La dernière partie de ce chapitre sera consacré à présenter quelques travaux sur la spécification de tels systèmes.

2. Définition des systèmes auto-adaptatifs

Malgré que les systèmes auto-adaptatifs sont devenus un besoin et une réalité inévitable. On ne dispose pas encore de normes ou de standard, et il n'y a pas une définition ou vocabulaire commun pour définir un système auto-adaptatif. Parmi les plusieurs définitions existantes, nous présentons quelques-unes :

Définition 1. Un système logiciel auto-adaptatif (SAA) est un système logiciel qui peut adapter son comportement et sa structure en réponse à sa perception de l'environnement, les systèmes lui-même et ses objectifs [1].

Définition 2. Un système auto-adaptatif évalue son propre comportement et modifie sa propre performance lorsque l'évaluation indique que n'est pas accompli ce que le logiciel est censé faire, ou lorsque de meilleures fonctionnalités ou performances sont possibles [2].

Définition 3. Un système auto-adaptatif est un système en boucle fermée qui peut se modifier lui-même dû aux changements continus du système, ses exigences et les tendances existantes de développement et de déploiement des systèmes complexes, réduisant les interactions humaines. La conception des systèmes auto-adaptatifs dépend des

besoins de l'utilisateur et des propriétés et caractéristiques environnementales du système. Les logiciels auto-adaptatifs nécessitent une grande fiabilité, robustesse, adaptabilité et disponibilité [3].

3. Les capacités des systèmes auto-adaptatifs

Selon [4] les capacités essentielles caractérisant un système auto-adaptatif sont : l'observation, la décision et l'intro-action :

- **Observation** : c'est la capacité d'observer l'environnement pour détecter les changements, cela veut dire existence des sondes logicielles et matérielles pour l'aide à mesurer les propriétés pertinentes de l'environnement.
- **Décision** : c'est la capacité qu'un système auto-adaptatif prend une telle décision de configuration par lui-même (de manière autonome) pour un meilleur fonctionnement.
- **Intro-action** : le mot « intro-actions » veut dire : des actions de l'intérieur sur l'intérieur, c'est la capacité qu'un système auto-adaptatif de manipuler et modifier les éléments qui le constituent afin de modifier sa configuration.

4. Propriétés des systèmes auto-adaptatifs

Un système auto-adaptatif s'adapte à l'exécution aux changements de lui-même et de l'environnement. Pour y parvenir, idéalement, les systèmes devraient avoir certaines caractéristiques connues sous le nom de propriétés auto-*. Ces propriétés sont introduites dans l'informatique autonome[5, 6] et ont été mentionnés ci-après dans le contexte d'auto-adaptation dans de nombreux ouvrages[2, 7] comme base de l'adaptation. La Figure 1.1 présente les propriétés d'un système auto-adaptatif.

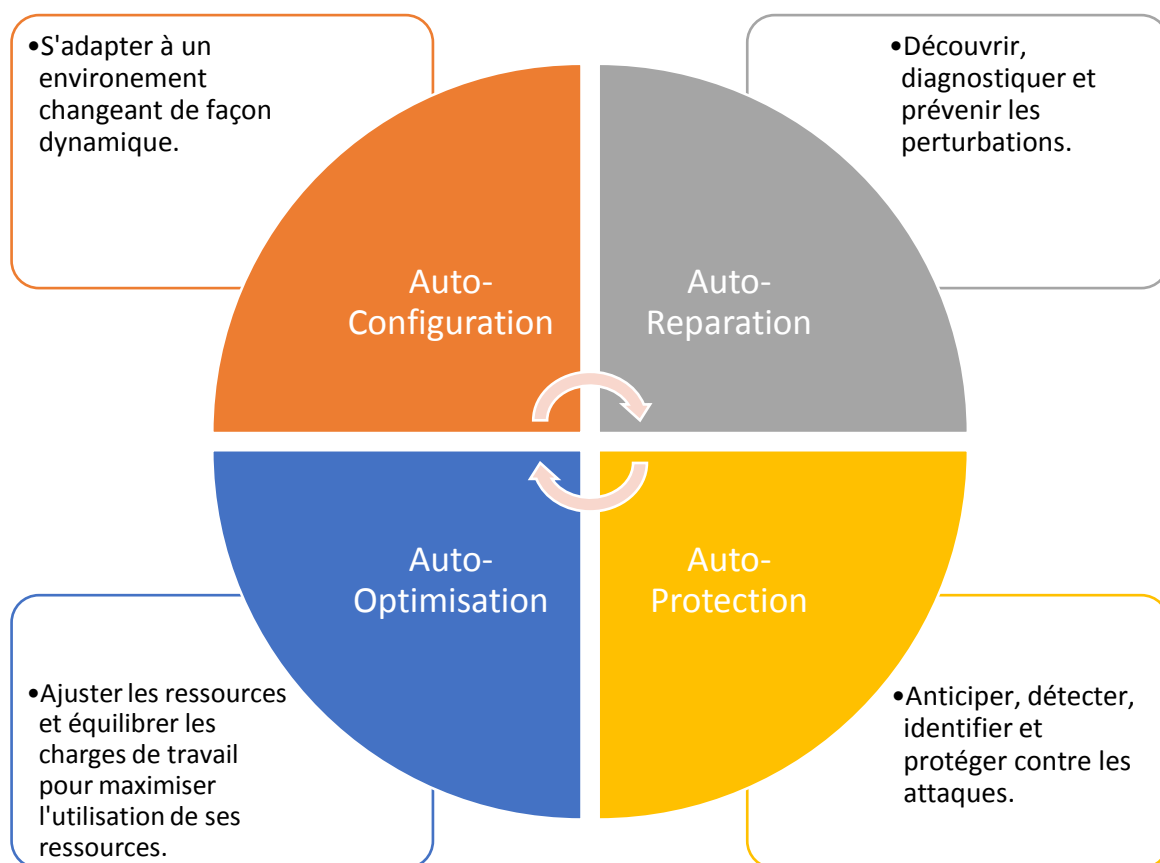


Figure 1.1 : Propriétés des systèmes auto-adaptatifs [8]

Ces propriétés sont composées de quatre catégories suivantes :

4.1. Auto-configuration

C'est la capacité de reconfigurer automatiquement et dynamiquement en réponse aux changements. Cela peut inclure l'installation, l'intégration, supprimer et composer / décomposer des éléments du système.

4.2. Auto-optimisation

C'est la capacité de gérer les performances et l'allocation des ressources tout en satisfaisant les besoins des utilisateurs. Cela inclut des préoccupations telles que débit, temps de réponse, etc.

4.3. Auto-réparation

C'est la capacité de découvrir, diagnostiquer et réagir les perturbations. Cela comprend à la fois une réparation réactive ou proactive.

Dans le cadre de la réparation proactive, les problèmes potentiels sont anticipés et traités

dès le début pour prévenir échec. Tandis que l'auto-réparation se concentre sur la récupération d'eux.

4.4. Auto-protection

C'est la capacité de détecter les failles de sécurité et de récupérer de leurs effets. Cela comprend à la fois une protection réactive et proactive, à savoir la récupération des attaques existantes et anticipées.

5. Classification des systèmes auto-adaptatifs

Dans la littérature, plusieurs travaux ont essayé de classifier les systèmes auto-adaptatifs, dans cette section là nous présentons ces différentes classifications dans la Figure 1.2 et nous les détaillons par la suite.

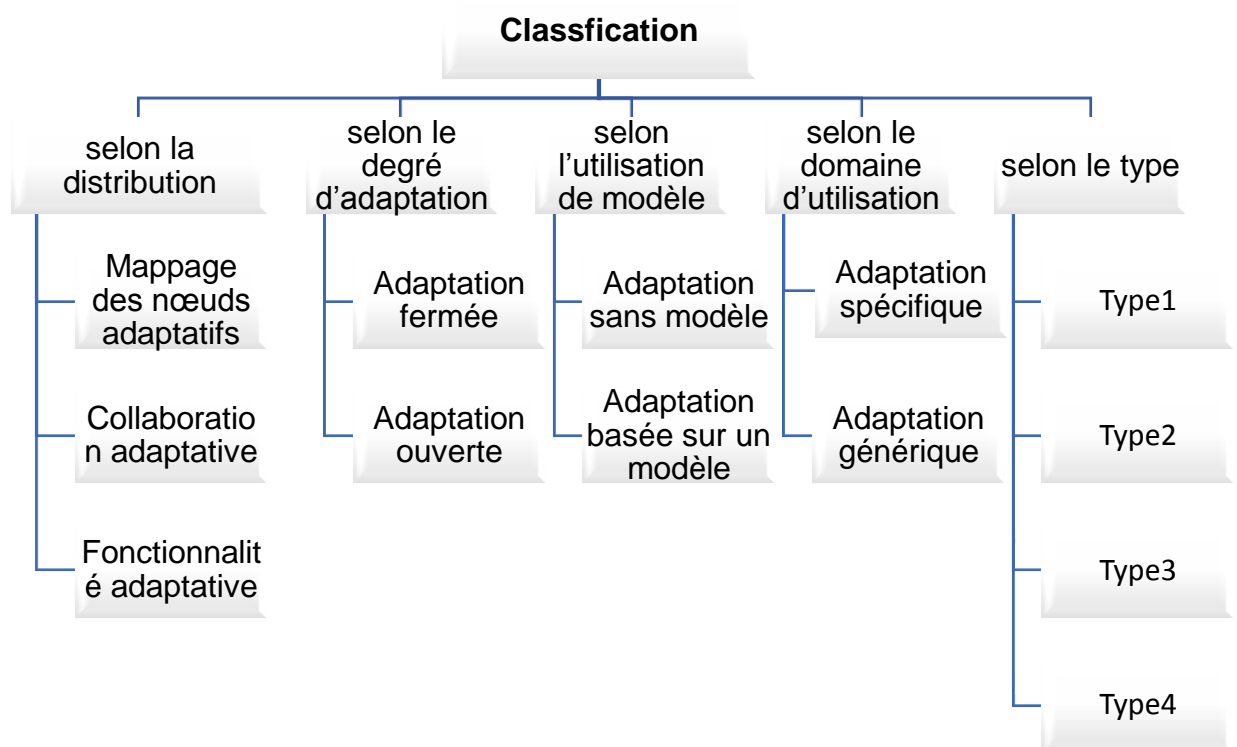


Figure 1.2 : Classification des systèmes auto-adaptatifs

5.1. Classification selon la distribution

5.1.1. Mappage des nœuds adaptatifs

C'est la capacité des systèmes distribués à déplacer certains modules logiciels vers d'autres nœuds matériels lors de l'exécution [9].

5.1.2. Collaboration adaptative

Est obtenue si les modules d'un système peuvent être reconnectés au moment de l'exécution. C'est particulièrement possible dans le cas des informations redondantes, où chaque source d'information peut avoir différente qualité [9].

5.1.3. Fonctionnalité adaptative

Décrit la capacité d'un système à adapter sa fonctionnalité réelle, grâce à quoi tout est possible, de l'adaptation de certains paramètres à un échange complet de l'algorithme [9].

5.2. Classification selon le degré d'adaptation

5.2.1. Adaptation fermée

Un système adaptatif-fermé n'a qu'un nombre fixe des actions adaptatives, et aucun nouveau comportement ni alternative ne peuvent être introduit lors de l'exécution [2].

5.2.2. Adaptation ouverte

D'autre part, dans l'adaptation ouverte, les logiciels auto-adaptatifs peuvent être étendus, et par conséquent, de nouvelles alternatives peuvent être ajoutées, et même de nouvelles entités adaptables peuvent être introduite

s dans le mécanisme d'adaptation [2].

5.3. Classification selon l'utilisation de modèle

5.3.1. Adaptation sans modèle

En adaptation sans modèle, le mécanisme n'a pas de modèle prédéfini pour l'environnement et le système lui-même. En fait, en connaissant les exigences, les objectifs et les alternatives, le mécanisme d'adaptation ajuste le système [2].

5.3.2. Adaptation basée sur un modèle

D'autre part, dans l'adaptation basée sur un modèle, le mécanisme utilise un modèle du système et de son contexte. Cela peut être réalisé en utilisant différentes approches de modélisation [2].

5.4. Classification selon le domaine d'utilisation

5.4.1. Adaptation spécifique

Certaines des solutions existantes ne concernent que des domaines / applications spécifiques, le type spécifique se concentre uniquement sur une adaptation d'artefacts ou d'attributs d'une partie particulière du système [2].

5.4.2. Adaptation générique

Contrairement à l'adaptation spécifique, des solutions génériques sont également disponibles, qui peuvent être configurées pour définir des politiques, des alternatives et des processus d'adaptation pour différents domaines [2].

5.5. Classification selon le type

5.5.1. Type 1

L'adaptation de type 1 est la mise en œuvre la plus simple de systèmes intelligents. Il surgit d'une analyse approfondie du domaine pour analyser tous les changements possibles que le système observera et auxquels il réagira. L'activité de conception comprend l'étude de toutes les réactions possibles que le système va déclencher. La conception conduit à la définition d'un modèle comportemental qui contient des points de décision : un point de décision est analogue à une instruction «si ... alors ... sinon » (« if ... then... else »), alors que les conditions dépendent généralement des perceptions de l'environnement d'exécution [10].

5.5.2. Type 2

L'adaptation de type 2 consiste en des systèmes complexes qui offrent de nombreuses stratégies alternatives pour atteindre le même objectif. Différentes stratégies englobent différentes situations opérationnelles et ont un impact différent sur les exigences non fonctionnelles. Cette situation nécessite de revoir la phase d'analyse traditionnelle des exigences pour inclure l'exploration détaillée d'un grand espace de problèmes. La sélection de la stratégie contextuelle à appliquer est une décision d'exécution qui considère des compromis complexes entre l'état de l'environnement et la qualité de service souhaitée. Cette décision nécessite un niveau de connaissance plus élevé que le type 1. Elle doit inclure la prise de conscience des exigences fonctionnelles / non fonctionnelles et de la manière différente dont chaque opération les affecte [10].

5.5.3. Type 3

L'adaptation de Type 3 représente une implémentation avancée d'un système intelligent qui est construit avec un ensemble de fonctionnalités de base. Le système peut être utilisé pour assembler des comportements dédiés qui ne figurent dans aucune des stratégies de solution prédéfinies. Ce type de système est particulièrement adapté pour travailler avec des connaissances incertaines sur l'environnement et les exigences [10].

5.5.4. Type 4

L'adaptation de type 4 est le niveau supérieur d'un système intelligent capable de s'auto-inspecter, d'apprendre de l'expérience et d'auto-modifier sa spécification. Ils sont conçus pour permettre les pires cas d'adaptation : lorsque le système ne possède pas les actions / stratégies appropriées à utiliser, et qu'il n'est capable d'en générer aucune. Dans ce cas, le système est en mesure de réviser son modèle d'exécution, donc de produire une nouvelle version du logiciel. Dans cette catégorie d'adaptation, il est plus approprié de se référer à l'évolution. En effet, ces systèmes sont inspirés des systèmes biologiques qui possèdent la capacité de faire face à la variance de l'environnement par des changements génétiques [10].

6. Modèle conceptuel

Le modèle conceptuel décrit les éléments abstraits composant un système auto-adaptatif et la relation entre eux. En d'autres termes, il présente les principes de base des systèmes auto-adaptatifs. Le modèle conceptuel est composé de trois éléments : le système géré, le moteur d'adaptation et l'environnement. Par conséquent, un système auto-adaptatif peut être vu comme un tuple :

Systeme Auto-Adaptatif (SAA) = (Systeme Géré (SG), Moteur d' Adaptation (MA), Environnement (E)).

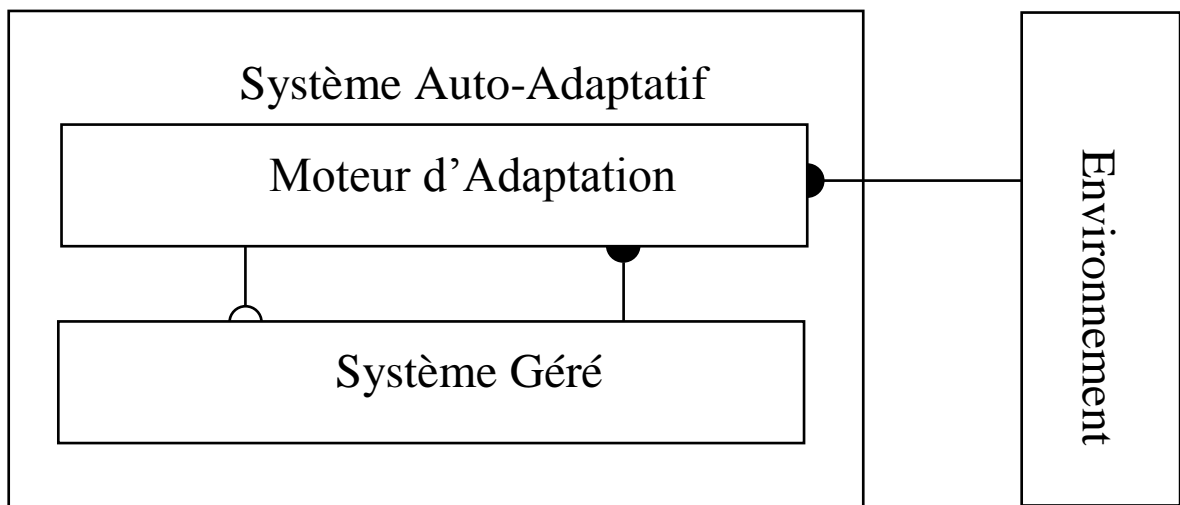


Figure 1. 3 : Modèle conceptuel pour les systèmes auto-adaptative [11]

La Figure 1.3 illustre l'anatomie du modèle conceptuel et une description de chaque entité est présentée ci-après.

6.1. Système géré

Comprend le code d'application qui réalise la fonctionnalité du système. Dans le cas des systèmes adaptatifs collaboratifs, le système géré peut être considéré comme une série de ressources telles que des robots, des véhicules, etc. Pour prendre en charge l'adaptation, le système géré est équipé d'actionneurs. Les actionneurs permettent d'exécuter des demandes d'adaptation sélectionnées par le moteur d'adaptation. Par exemple, étant donné que plusieurs robots collaborent pour transporter un élément du point A au point B, le système géré est responsable de la navigation des robots et du transfert des éléments. Les actionneurs peuvent restreindre cinq robots à participer au transfert de l'élément en fonction de son poids.

Différents termes sont utilisés dans la littérature pour désigner le concept de système géré. Par exemple, il est également appelé élément géré, couche système, logiciel adaptable, ressources gérées, sous-système de niveau de base et couche de contrôle des composants [11].

6.2. Moteur d'adaptation

Supervise et administre le système géré. Il contient la logique d'adaptation nécessaire pour atteindre les exigences ou les objectifs du système. Le moteur d'adaptation est équipé de capteurs qui surveillent à la fois le système géré et l'environnement et adapte le précédent si nécessaire. Le moteur d'adaptation analyse les données suivies et construit un plan d'adaptation. Par exemple, considérons un robot qui adapte sa stratégie de navigation en fonction de la présence d'obstacles (détectés depuis l'environnement) et de son niveau d'énergie (détecté depuis le système géré).

Différents termes sont utilisés dans la littérature pour désigner le concept de moteur d'adaptation. Par exemple, il est également appelé gestionnaire autonome, couche d'architecture, logique d'adaptation et sous-système réfléchissant [11].

6.3. Environnement

Fait référence au monde extérieur avec lequel le système interagit et par lequel il est effectué. Il comprend des entités physiques telles que des obstacles sur le chemin d'un robot [11].

7. Boucle d'adaptation

Le système logiciel auto-adaptatif incarne un mécanisme en boucle fermée. Cette boucle, appelée boucle d'adaptation, est constituée de plusieurs processus, ainsi que de capteurs et d'effecteurs. Cette boucle est appelée boucle MAPE-K dans le contexte du calcul autonome, et comprend les fonctions de surveillance, d'analyse, de planification et d'exécution, avec l'ajout d'une base de connaissances partagée [2].

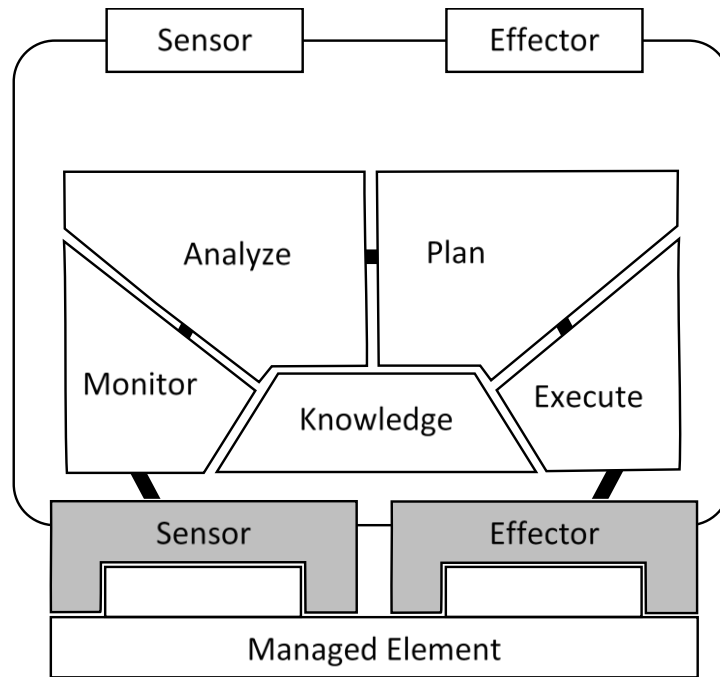


Figure 1. 4 : MAPE-K Loop[12]

L'efficacité du MAPE-K vient de sa structure intuitive dans la gestion des différentes fonctions requises pour une boucle de rétroaction. La Figure 1.4 illustre le processus de la boucle d'adaptation. La première étape consiste à surveiller et à collecter les données de l'environnement et du système géré via des capteurs. Les données collectées sont traitées et les connaissances sont mises à jour. Ensuite, les connaissances à jour sont analysées pour déterminer si une adaptation est nécessaire pour atteindre les exigences ou les objectifs du système. Ensuite, si l'adaptation est obligatoire, un Plan est construit comprenant une ou plusieurs actions d'adaptation. Enfin, le plan est exécuté par le système géré à l'aide d'actionneurs [11].

7.1. Surveillance (Monitor)

Cette partie est responsable de la surveillance des ressources gérées et de la collecte, de l'agrégation et du filtrage des données. La surveillance est effectuée par des capteurs [12].

7.2. Analyseur (Analyze)

Cette partie de la boucle analyse les données rapportées par la partie moniteur. L'analyse vise à comprendre quel est l'état actuel et si des mesures doivent être prises [12].

7.3. Plan

Dans la partie plan, un plan d'action est préparé sur la base des résultats de l'analyse. Le plan est un certain nombre de mesures qui feront passer le système de son état actuel à un état souhaité [12].

7.4. Exécuteur (Execute)

Dans cette partie, le plan est exécuté et contrôlé. Les effecteurs exécutent les actions planifiées sur la ressource gérée [12].

7.5. Base de Connaissances (Knowledge)

La source de connaissances est au cœur de la boucle de contrôle et est accessible par toutes les parties de la boucle. Outre les données collectées et analysées, il contient des connaissances supplémentaires telles que des modèles architecturaux, des modèles d'objectifs, des politiques et des plans de changement [12].

8. Comparaison de quelques approches sur les systèmes auto-adaptatifs

Dans la littérature plusieurs travaux ont réussi à modéliser les systèmes auto-adaptatifs. Cependant ces approches appartiennent à des différents types comme montre la Figure 1.5.

Dans cette section nous détaillons les deux types 'basée sur l'architecture' et 'basée sur les modèles', pour que nous permettra de faire une comparaison entre des différents approches.

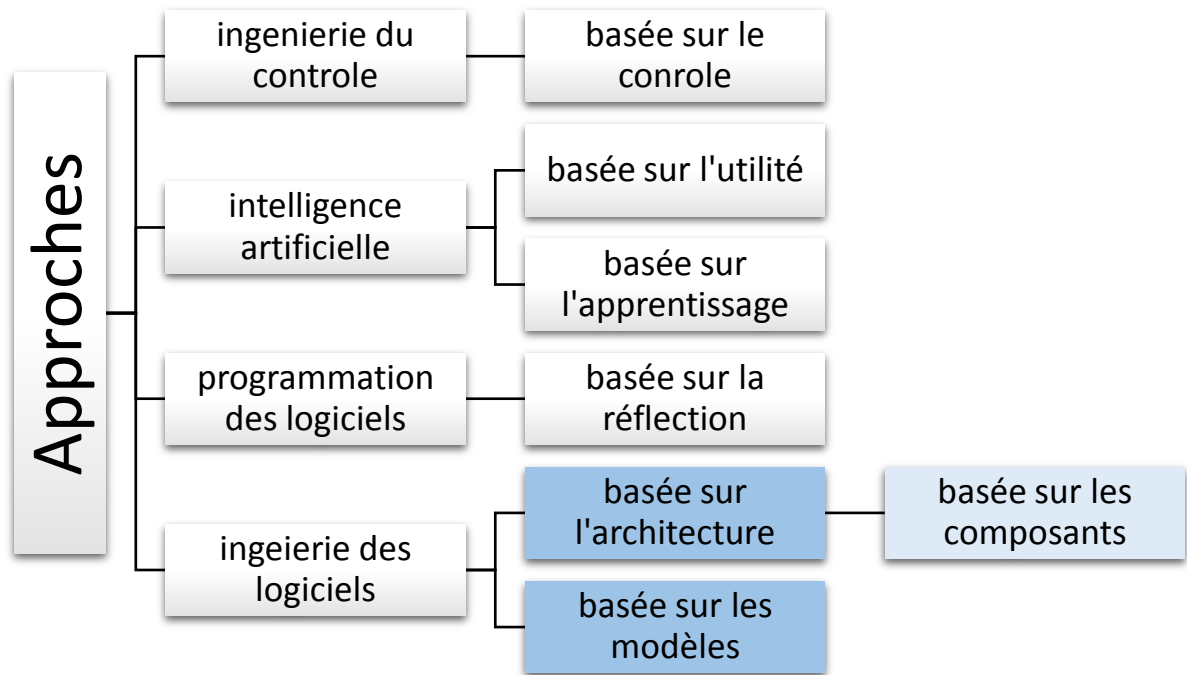


Figure 1. 5 : Quelques approches sur les systèmes auto-adaptatifs[11]

8.1. Approches basées sur l'architecture

Une architecture logicielle est un ensemble d'éléments logiciels (ou de composants), Les relations entre eux, ainsi que les propriétés des deux et désigne la structure de haut niveau du logiciel. Concernant les besoins d'adaptation, les architectures logicielles sont utilisées pour différentes activités. SAA doit être conscient de sa structure, qui s'appelle la conscience de soi. Les architectures logicielles peuvent être utilisées pour représenter la structure du système et raisonner sur les niveaux d'adaptation. En revanche, les architectures logicielles sont utilisées pour la construction des SAA et la définition des responsabilités. Dans cette section, nous présentons des approches basées sur l'architecture [11].

8.2. Approches basées sur les composants

Dans une architecture basée sur des composants, la description du système est composée de composants qui encapsulent les fonctionnalités du système et de connecteurs qui dictent l'interaction entre les composants. Les connecteurs relient un composant à un autre généralement via des relations telles que le flux de données ou le flux de contrôle.

Une description d'architecture de composant et de connecteur peut aider à la construction d'auto-adaptative en permettant au système de garder une trace de sa structure.

En d'autres termes, il suscite une conscience de soi structurelle, ce qui est particulièrement important pour capturer le comportement d'auto-configuration [11].

8.3. Approches basées sur les modèles

Un modèle est une représentation du système à un certain niveau d'abstractions. Un modèle peut représenter les exigences, l'architecture, l'implémentation ou le développement du système, selon la préoccupation en question, le modèle ne capture que les informations pertinentes par rapport à la préoccupation du modèle. D'autres types de modèles encapsulent les propriétés non fonctionnelles d'un système telles que les performances, la tolérance et la sécurité, etc. Un modèle est décrit à l'aide d'un langage de modélisation, qui est généralement composé d'une syntaxe abstraite, d'une syntaxe concrète et d'une sémantique. La syntaxe abstraite décrit les concepts du langage et leur composition pour créer un modèle. La syntaxe concrète est une notation textuelle ou graphique utilisée pour décrire un modèle. La sémantique utilise le sens du langage, c'est-à-dire l'interprétation d'un modèle écrit dans la langue correspondante. La sémantique peut être définie formellement en utilisant des notations mathématiques ou de manière informelle en utilisant le langage naturel [11].

8.4. Comparaison

Il existe dans la littérature plusieurs approches qui ont essayé de modéliser les systèmes auto-adaptatifs. Dans le tableau 1.1 nous faisons une comparaison entre ces approches en utilisant des critères. Ces critères sont :

- Type : le type que sur lequel se base l'approche étudiée (Basée sur l'architecture, Basé sur les composants, Basée sur le modèle et Basée sur le scénario)
- Modèle utilisé : le modèle que l'approche a utilisé.
- Langage généré : les langages générés par l'approche, pour spécifier ou bien valider l'approche.
- Boucle MAPE : l'utilisation de la boucle selon le besoin, M/A/P/E = Monitoring, Analyzing, Planning, Executing.
- Validation : le cas d'étude choisis pour valider l'approche, on s'intéresse à la capacité de l'approche pour la spécification de :
 - L'adaptation structurelle : qui permet l'échange de structure ou les composants système de manière dynamique au moment de l'exécution.

- L'adaptation comportementale : modification du comportement du système en ajustant les paramètres du système et de ses composants.
- Les outils : les outils utilisés par l'approche pour la spécification de l'adaptation.

Approche	Type	Modèle Utilisé	Langage généré	Boucle MAPE	Adaptation structurelle	Adaptation comportementale	Les outils
Rainbow [13]	- Basée sur l'architecture - Basé sur les composants	- Modèle d'architecture - Modèle d'environnement	- Stitch	- Toute	- Oui	- Oui	- Utilisation le langage de spécification 'Stitch' pour spécifier les deux types d'adaptation.
Event-based [14]	- Basée sur le modèle	- Modèle d'architecture	- Event-B	- Toute	- Oui	- Oui	- Utilisation de 'Event-B contexts' pour spécifier l'adaptation structurelle. - Utilisation de 'Event-B machines' pour spécifier l'adaptation comportementale.
PobSAM [15]	- Basée sur les modèles	- Modèle formel	- PobSAM	- Non	- Non	- Oui	- Utilisation de son propre langage PobSAM pour spécifier l'adaptation comportementale.
Patterns-based [16]	- Basée sur le modèle	- Modèle de conception		- Toute	- Oui	- Oui	- Utilisation la couche technique de model de conception pour spécifier l'adaptation.
An Aspect Oriented and modeling driven [17]	- Basée sur le modèle	- Modèle de fonctionnalité - Modèle d'architecture		- P/E	- Oui	- Non	- Utilisation de AOM pour la représentation de la variabilité.
Fusion [18]	- Basée sur le modèle	- Modèle de fonctionnalité	- FSP	- A, P, E, K	- Oui	- Oui	- Utilisation le langage 'FSP' pour spécifier l'adaptation structurelle et comportementale.

Table 1. 1 : Comparaison entre quelques approches qui ont essayé de modéliser les systèmes auto-adaptatifs

8.5. Discussion

Après avoir étudié différents travaux dans la littérature (Tableau 1.1) qui ont spécifié les systèmes auto-adaptatifs, nous avons constaté que la plupart des approches soit proposent une adaptation comportementale soit structurelle, et spécifient rarement les deux types. Les approches décrivant l'aspect comportementale spécifient seulement quelques types de variabilités dans le cas d'étude utilisée, à notre connaissance il n'existe pas encore une approche générique qui spécifie d'une manière générale les différents types de variabilités qu'on peut trouver dans les SAA.

Par contre dans notre travail nous allons proposer une approche générique qui permet de prendre en considération les deux types d'auto-adaptation : structurelle et comportementale. L'idée est d'utiliser un métamodèle combinant entre l'approche à composants pour spécifier l'auto-adaptation structurelle et les modèles de variabilité pour pouvoir spécifier l'auto-adaptation comportementale pour les différents types des systèmes auto-adaptatifs. Les différentes variabilités peuvent être spécifiées comme des propriétés non fonctionnelles au niveau de chaque composant du système.

9. Conclusion

Dans ce premier chapitre, nous avons présenté le concept générale au quel s'intègre notre travail et plus spécifiquement les éléments de bases, quelques approches de spécification des systèmes logiciels auto-adaptatifs. Malgré les progrès récents dans ce domaine, un aspect clé des systèmes auto-adaptatifs qui reste à être abordé en profondeur est la spécification des propriétés fonctionnelles et non fonctionnelles lors du fonctionnement en présence d'auto-adaptation. Le chapitre suivant, aura pour objectif de présenter les modèles existants dans la littérature pour la spécification de la variabilité. Une compréhension de ce nouveau domaine nous aidera certainement à bien construire notre approche.

CHAPITRE 2

LES MODELES DE SPECIFICATION DE LA VARIABILITE DANS SPL

1. Introduction

L'ingénierie des lignes de produits logiciels optimise le développement des systèmes individuels en tirant parti de leurs caractéristiques communes et en gérant leurs différences de manière systématique. Ces différences sont appelées variabilités.

Il existe dans la littérature plusieurs modèles de variabilité, parmi ces modèles ce qu'on appelle feature modèle (modèle de fonctionnalité).

Ce chapitre a pour objectif de présenter les notions de bases de lignes de produits logiciels et modèles de la variabilité notamment les feature models que nous allons utiliser pour notre approche.

2. Ligne de produits logiciels

L'objectif principal des lignes de produits (LdP) est qu'un groupe de systèmes similaires puisse être construit plus efficacement si ces systèmes sont développés ensemble, plutôt qu'indépendamment les uns des autres, tout en tirant parti des points communs entre les systèmes de manière planifiée.

La définition dominante de ligne de produits logiciels est celle-ci donnée par Paul Clements et Linda Northrop : « *Un ensemble de systèmes à forte intensité de logiciels qui partagent un ensemble commun et géré de fonctionnalités répondant aux besoins spécifiques d'un segment de marché ou d'une mission particulière et qui sont développés à partir d'un ensemble commun d'actifs de base d'une manière prescrite* » [19].

La Figure 2.1 montre les deux niveaux d'ingénierie qui caractérisent le Software Product Line Engineering (SPLE), qu'ils sont : Domain Engineering et Application Engineering.

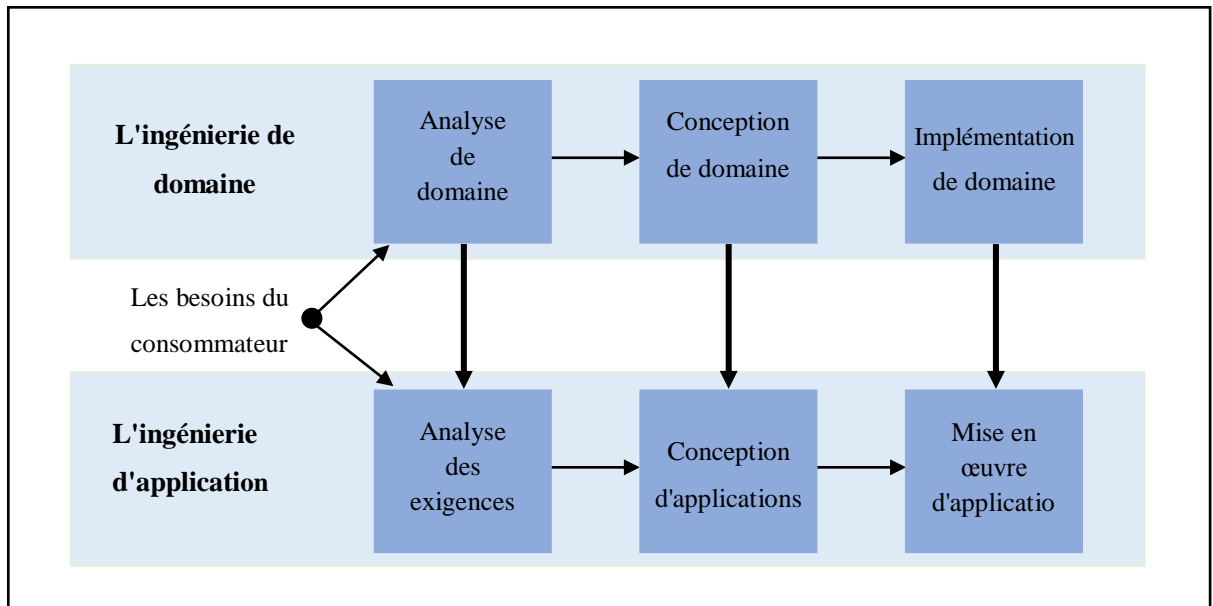


Figure 2.1 : Niveaux d'ingénierie de LdP[20]

L'ingénierie de domaine correspond à l'étude du domaine de la ligne de produits, l'identification des points communs et des variabilités entre les produits, la mise en place d'une architecture logicielle générique et la mise en œuvre de cette architecture. Consiste en la construction de composants réutilisables dits actifs qui seront réutilisés pour la construction des produits[20].

L'ingénierie d'application est utilisée pour trouver l'utilisation optimale pour le développement d'un nouveau produit à partir d'une ligne de produits en réduisant les coûts et le temps de développement et en améliorant la qualité. À ce niveau, les résultats de l'ingénierie du domaine sont utilisés pour la dérivation d'un produit particulier. Cette dérivation correspond à la prise de décision vers les points de variation[20].

3. Variabilité

3.1. Définition

La variabilité est la capacité d'un système à être efficacement étendu, modifié, personnalisé ou configuré pour une utilisation dans un contexte particulier [21]. Une autre définition présente la variabilité comme la capacité d'un système, d'un actif ou d'un environnement de développement à prendre en charge la production d'un ensemble d'artefacts qui diffèrent les uns des autres de manière pré-planifiée [22].

3.2. Classification de la variabilité

Plusieurs travaux dans la littérature ont classifié la variabilité:

3.2.1. Classification selon le niveau de l'exigence

3.2.1.1. La variabilité essentielle

La variabilité essentielle correspond au point de vue du client, définissant ce qu'il faut mettre en œuvre [23].

3.2.1.2. La variabilité technique

La variabilité technique concerne l'ingénierie de la famille de produits, définissant comment la mettre en œuvre [23].

3.2.2. Classification basée sur les dimensions de la variabilité

3.2.2.1. La variabilité dans le temps

La variabilité dans le temps concerne l'existence de différentes versions d'un artefact, valables à des moments différents [24].

3.2.2.2. La variabilité dans l'espace

La variabilité dans l'espace définit l'existence d'un artefact sous différentes formes en même temps [24].

3.2.3. Classification selon le niveau de la visibilité

3.2.3.1. La variabilité externe

La variabilité externe est visible pour les clients [24].

3.2.3.2. La variabilité interne

La variabilité interne comme celle du domaine, et qui par conséquent lui est masquée [24].

4. Les modèles de spécification de la variabilité

Dans l'ingénierie des lignes produit existent différentes approches utilisées pour modéliser la variabilité, Dans cette partie nous présentons ces différentes approches.

4.1. FM (Feature Model)

Un modèle de fonctionnalité (Feature Model) est un moyen de représenter l'espace des configurations possibles de tous les produits dans un SPL[25, 26].

4.2. OVM (Orthogonal Variability Model)

Le modèle de variabilité orthogonale (OVM) est l'autre concept pour exprimer la variabilité sur le processus SPL à côté de la FM. [27].

4.3. CVL (Common Variability Language)

Le langage de variabilité commun (CVL) est un langage pour le domaine indépendant, utilisé pour définir et résoudre les problèmes de variabilité dans Langage dédié (Domain-Specific language) ou (DSL) [28].

4.4. DM (Décision Model)

Un modèle de décision est un ensemble des décisions qui permettent de distinguer les membres d'une famille de produits d'ingénierie d'application et pour guider l'adaptation des produits de travail d'ingénierie d'application [29].

5. Feature Model

Tous les features models représentent les features (caractéristiques) sous la forme d'un graphe dont la structure élémentaire est un arbre où les features sont décrites hiérarchiquement selon une relation d'affinement. Dans cette section nous représentons les types de features models ainsi que leurs structures.

5.1. Feature Models basiques

Les modèles de caractéristiques basiques—à la FODA (voir section 5.1.1)—permettent de modéliser quatre grandes catégories de relations entre les features :

obligatoire, optionnelle, à choix alternatif, à choix multiple, requiert et exclut, qu'elles sont présentées dans la Figure 2. 2.

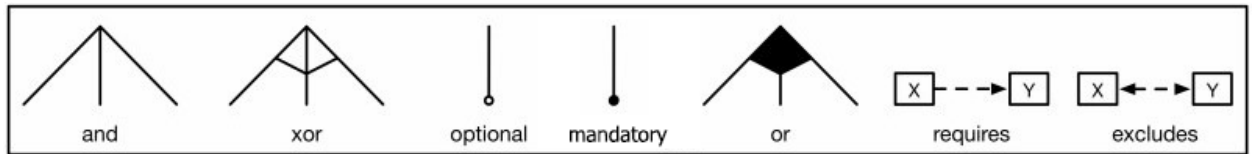


Figure 2.2 : Les éléments du diagramme de fonctions de base [30, 31].

- **Obligatoire** : lorsqu'une sous-feature possède une relation obligatoire avec son parent, le produit ne peut présenter la feature fille si et seulement si elle présente aussi la feature parent. Cette relation va donc dans les deux sens : un produit qui aurait la feature fille doit aussi présenter la feature parent [32].
- **Optionnelle** : lorsqu'une sous-feature possède une relation optionnelle avec son parent cela signifie que le produit ne peut présenter cette feature que s'il présente aussi la feature parent, mais le produit peut présenter la feature parent sans la sous-feature [32].
- **Choix alternatif (XOR)** un ensemble de sous-features est relié à une feature parent par une relation de choix alternatif si et seulement si le produit ne peut présenter qu'une et une seule des sous-features lorsqu'il possède la feature parent. À l'inverse, aucune des sous-features n'apparaît lorsque le produit ne présente pas la feature parent [32].
- **Choix multiple (OR)** : un ensemble de sous-features est relié à une feature parent par une relation à choix multiple si une ou plusieurs sous-features peuvent être incluses dans un produit qui présente la feature parent [32].
- **Requiert** : lorsqu'une feature A requiert une feature B, cela signifie que lorsqu'un produit présente la feature A, alors il doit aussi présenter la feature B [32].
- **Exclut** : une feature A exclut une feature B signifie que l'inclusion de A dans un produit exclut B du même produit réduit et dynamique, cela sera spécifié au moyen d'une dépendance « exclut » entre les deux features [32].

5.1.1. Approche FODA (Feature Oriented Domain Analysis)

FODA est un concept qui est utilisé pour analyser le problème de domaine sur SPL[30]. À partir de l'échantillon FODA de la Figure 2.3, on montre que l'arbre est

composé par le nom de la feature, la feature obligatoire, la feature optionnelle, la feature alternative et la règle de composition.

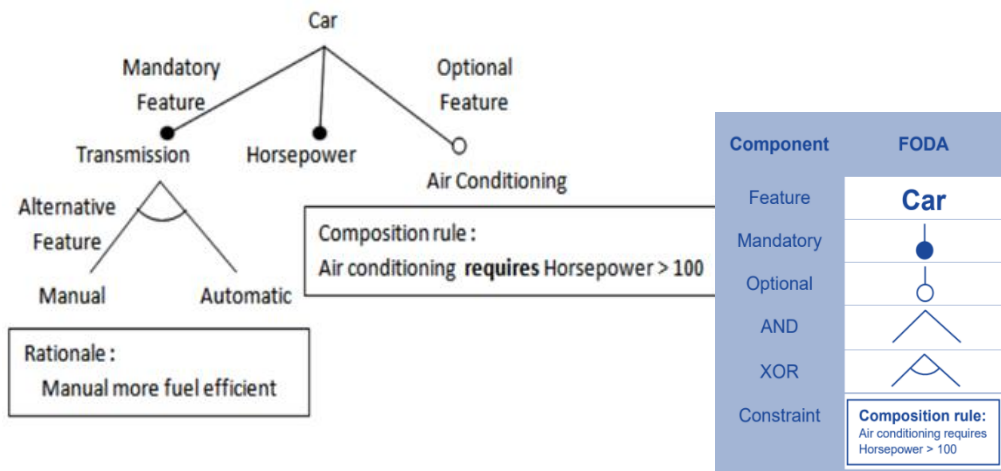


Figure 2.3 : Exemple de Feature Model FODA[30]

5.2. Feature Model avec cardinalités

Certains auteurs proposent d'étendre les modèles de caractéristiques FODA par des cardinalités UML [26, 33]. Deux types de cardinalités peuvent en fait être distingués : les cardinalités de caractéristiques et les cardinalités de groupes de caractéristiques comme illustré dans la Figure 2.4.

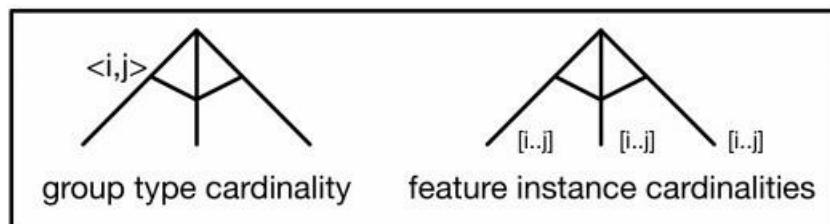


Figure 2.4 : La notation de Feature Model avec cardinalités [33, 34]

- **Une cardinalité de feature** est un intervalle noté $[n..m]$ associé à une feature, dans lequel n est la borne inférieure et m la borne supérieure de l'intervalle, qui détermine le nombre d'instances de la feature dans un produit [32].
- **Une cardinalité de groupe de feature** est un intervalle noté sous la forme, m étant la borne inférieure de l'intervalle et n la borne supérieure, limitant le nombre des feature de ce groupe que peut présenter un produit. Toutes les features du groupe auquel la cardinalité s'applique doivent avoir le même père. En

conséquence, une relation à choix alternatifs entre sous-features est sémantiquement équivalente à une cardinalité du groupe des features. De la même manière, une relation à choix multiples correspond à une cardinalité du groupe des features, n'étant le nombre total des features du groupe [32].

5.2.1. Approche CBFM (Cardinality-Based Feature Model)

CBFM a été intégré à plusieurs extensions FODA originales. Il s'agit d'un feature model hiérarchique où chaque caractéristique a une cardinalité [35].

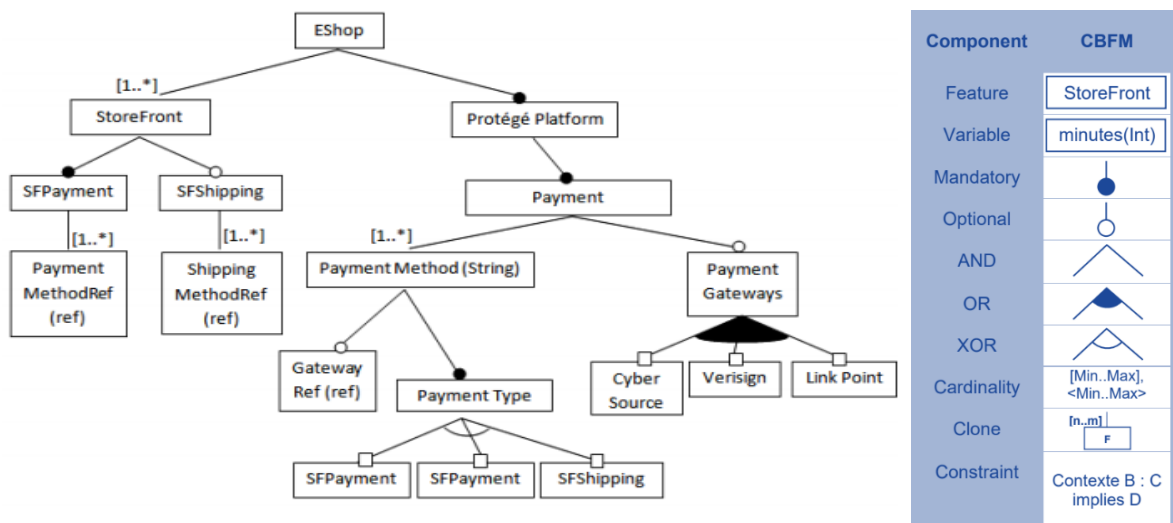


Figure 2.5 : Exemple d'extrait de Feature Model basé sur la cardinalité (CBFM) [35].

La Figure 2.5 montre un exemple de modélisation EShop par le modèle CBFM dont la cardinalité de feature est présente.

6. Conclusion

Les Features Models sont devenus les modèles les plus importants et les plus utilisés dans l'ingénierie des lignes de produits logiciels.

Dans ce chapitre nous avons présenté les concepts fondamentaux de lignes de produits ainsi que les modèles de variabilité, plus précisément, les Features Models. Nous avons acquis les connaissances sur les Features Models que nous allons utiliser dans notre métamodèle de spécification des systèmes auto-adaptatifs en couvrant les différents types de variabilité, nous avons compris leur structure à partir des approches existantes dans la littérature.

Dans le chapitre suivant nous allons décrire notre approche servant à modéliser les systèmes auto-adaptatifs.

CHAPITRE 3

NOTRE APPROCHE DE LA SPECIFICATION DES SYSTEMES AUTO-ADAPTATIFS

1. Introduction

Après avoir étudié les modèles de variabilités trouvées dans les lignes des produits logiciels, plus précisément, Feature Model : ses différents principes et ses types, nous présentons notre approche comme une nouvelle approche de spécification des systèmes auto-adaptatifs. Dans cette approche, nous nous concentrons sur la modélisation de la variabilité trouvée au niveau de l'architecture et la variabilité trouvée au niveau de la configuration du système en utilisant des différentes caractéristiques extraites du Feature Model.

Dans ce chapitre, comme montre la Figure 3.1, nous présentons d'abord le détail du métamodèle proposé. Ensuite nous détaillons le graphisme, puis nous présentons l'ADL que nous proposons ainsi que transformation de modèle pour obtenir notre ADL à partir d'un modèle de spécification. Enfin nous proposons une vérification pour la validation de modèle de spécification obtenu.

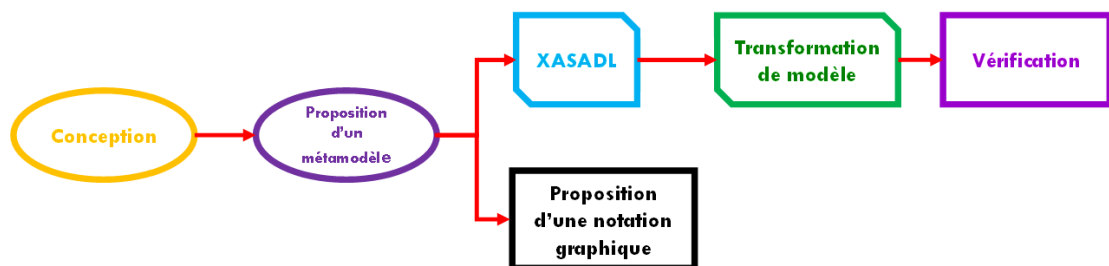


Figure 3. 1 : Plan de conception notre approche

2. Variabilité trouvée dans les systèmes auto-adaptatifs

Les systèmes auto-adaptatifs changent leur structure et leur comportement en fonction des variations trouvées dans leur environnement, pour cela, on distingue deux

types de variabilité : variabilité au niveau d'architecture et variabilité au niveau de configuration.

Dans cette section nous présentons le métamodèle proposé pour la spécification des systèmes auto-adaptatifs en mettant en évidence les deux types de variabilité.

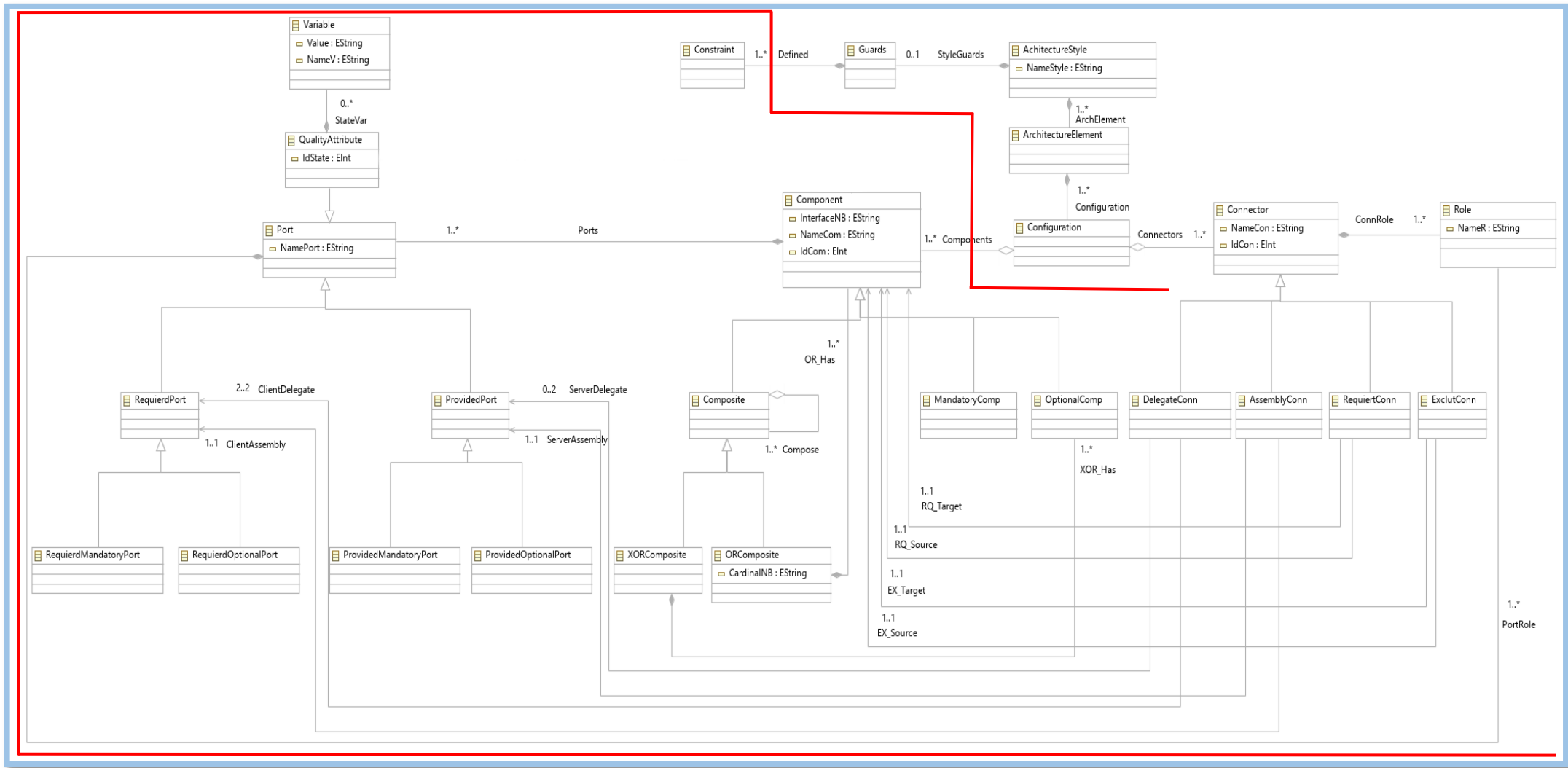


Figure 3. 2 : Métamodèle proposé

La Figure 3.2 présente le métamodèle proposé, nous commençons d'abord par le niveau structurel ensuite le niveau comportemental (configuratif).

2.1. Variabilité au niveau d'architecture

La variabilité au niveau d'architecture consiste aux changements structurels du système, tel que l'ajout ou suppression d'un élément structurel (composant, port, connecteur).

2.1.1. Composant

Un composant représente une unité de calcul ou de stockage de données à laquelle est associée une unité d'implémentation[36].

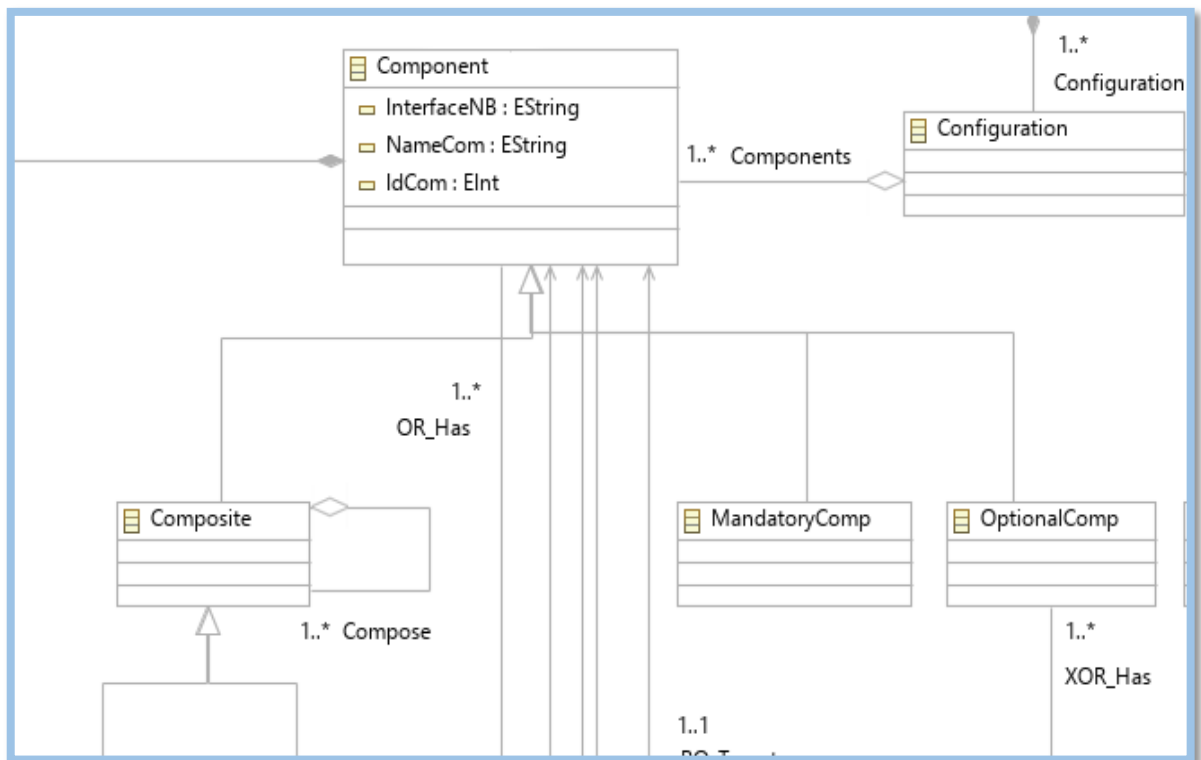


Figure 3.3 : La partie '*composants*' du métamodèle

La Figure 3.3 représente la partie des composants que nous avons défini par les deux types suivants :

- **MandatoryComponent** : un composant obligatoire est un composant qui soit présent (actif) obligatoirement dans une architecture.

- **OptionalComponent** : un composant optionnel est un composant qui peut être présent ou pas (actif/inactif) dans une architecture selon les besoins du système auto-adaptatif.

D'autre part, un composite est un composant qui possède une structure hiérarchique de sous-composants avec des composants primitifs comme feuilles[37]. Nous avons défini deux autres types du composite :

- **XOR** : un composant XOR est un composite qui comporte plusieurs autres sous-composants et qui n'exécute qu'un et qu'un seul sous-composant (1 seul sous-composant peut être actif à un moment donné, les autres sous-composants doivent être inactifs). Le composant XOR ne doit pas contenir des composants de type MandatoryComponent, car la présence de ce dernier forcera son exécution.
- **OR** : un composant OR est un composite qui exécute un ou plusieurs sous-composants, selon le nombre de cardinalité donné par l'utilisateur. C'est-à-dire plusieurs de ses sous-composants peuvent être actifs à un moment donné.

2.1.2. Port

Les points d'interaction d'un composant et d'une configuration sont appelés *ports* [38]. Deux types de ports peuvent être distingués dans la Figure 3.4 :

- **ProvidedPort** : Les ports fournis: ce sont des ports qui exportent les services des composants et des configurations à son environnement[38].
- **RequiredPort**: ce sont des ports qui importent les besoins des composants et des configurations[38].

Dans notre approche, nous définissons deux types qui appartiennent aux deux types de port précédents :

- **MandatoryPort** : est un port qui soit obligatoirement présent (actif) dans tout le cycle de vie du composant.
- **OptionalPort** : est un port qui peut être actif ou bien inactif dans le cycle de vie du composant.

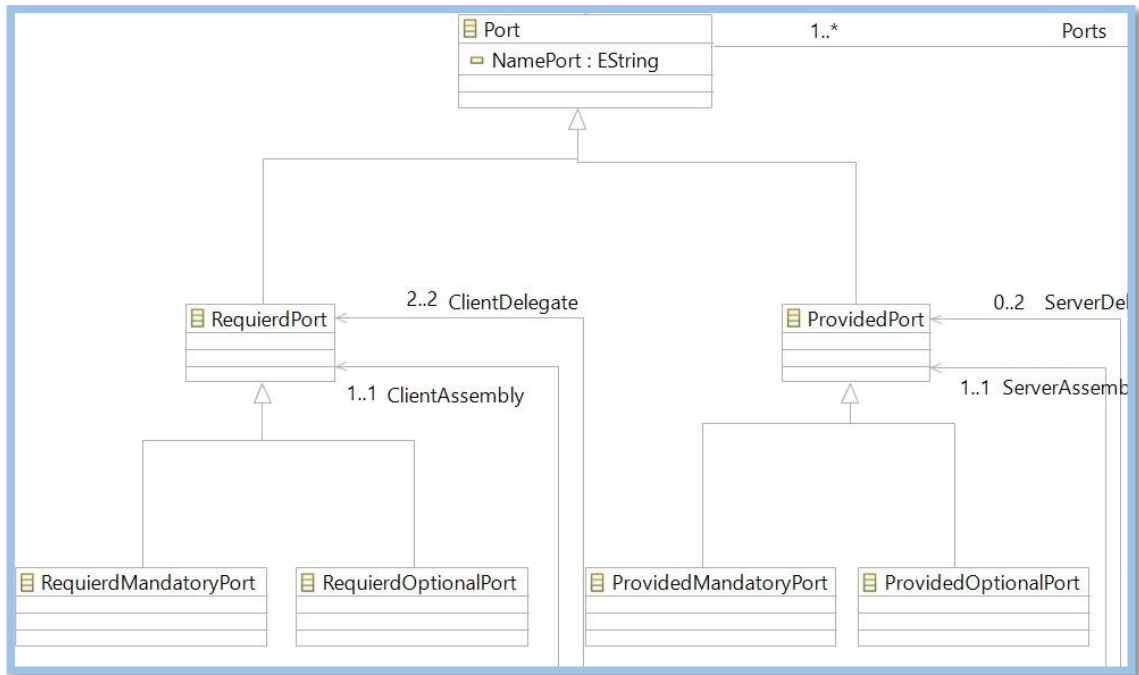


Figure 3.4 : La partie *'ports'* du métamodèle

2.1.3. Connecteur

Un connecteur est un bloc de construction utilisé pour exprimer les interactions entre **'composants'** ainsi que les règles qui gouvernent cette interaction. Les connecteurs sont des entités architecturales qui relient des ensembles de composants et agissent en tant que médiateurs entre eux [39].

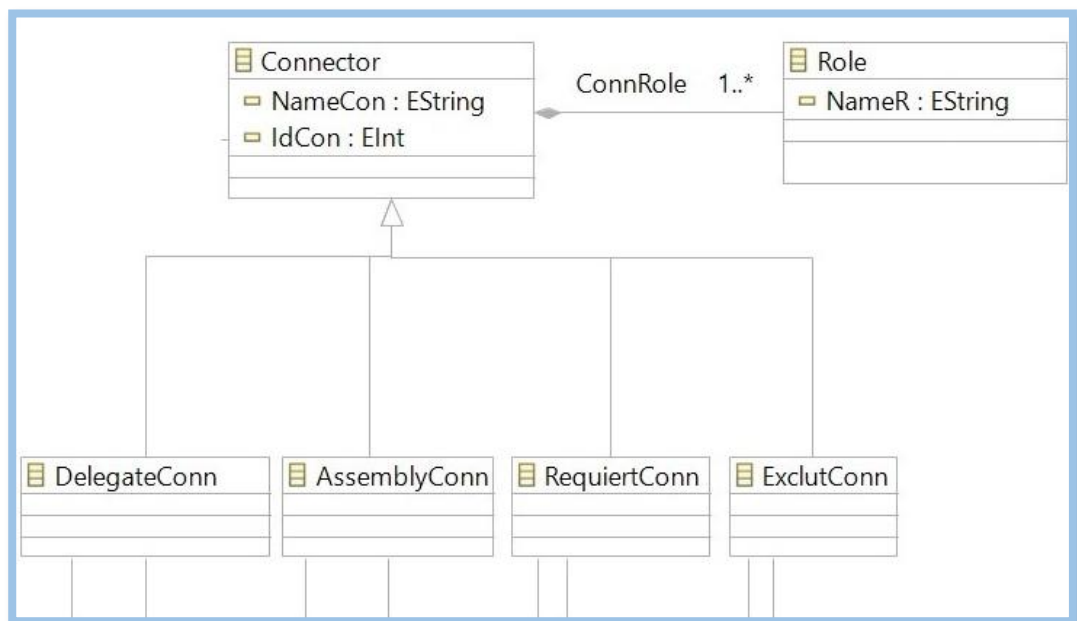


Figure 3.5 : La partie *'connecteurs'* du métamodèle

Les types de connecteurs illustrés dans la Figure 3.5, sont détaillés comme suit :

- **AssemblyConnector** : Un connecteur d'assemblage permet de relier deux ports de deux composants, il définit qu'un composant fournit les services dont un autre composant a besoin. Un connecteur d'assemblage est un connecteur défini à partir d'un port requis vers un port fourni.
- **DelegationConnector** : Un connecteur de délégation ne doit être défini qu'entre des Ports de même nature (par exemple, entre deux Ports fournis ou entre deux Ports requis). Il spécifie une déclaration selon laquelle le comportement disponible sur une instance de composant n'est pas réellement réalisé par ce composant lui-même, mais par une autre instance dotée de fonctionnalités compatibles. Un connecteur de délégation permet de relier les ports d'un composite avec ses sous-composants.

De plus de ces deux connecteurs nous avons définis deux autres types de control :

- **RequiertConnector** : lorsqu'un composant A est relié à un composant B par 'RequiertConnector', cela signifie que lorsque le composant A est actif, alors le composant B doit aussi être actif au même moment que A.
- **ExclutConnector** : lorsqu'un composant A est relié à un composant B par 'ExclutConnector', cela signifie que lorsque le composant A est actif, alors le composant B doit être inactif et vice versa.

2.2. Variabilité au niveau de la configuration

Contrairement la variabilité architecturale, la variabilité configurative consiste aux changements au niveau du composant, c'est à dire la modification de ces paramètres et de ces attributs. Dans cette partie nous définissons :

QualityAttribute : c'est un port qui est composé de plusieurs variables non-fonctionnels de l'environnement et leurs valeurs. Le changement de ces valeurs entraîne un changement au niveau des paramètres du composant et/ou un changement structurel de tout le système.

3. Représentation graphique et textuel des concepts proposés

3.1. Représentation textuelle

Avant de passer à la représentation graphique du métamodèle, nous mettons le point sur la représentation textuel de l'architecture.

Nous avons proposé un ADL (Architecture Dynamic Language) basé sur XML (eXtensible Markup Language) pour la spécification de l'architecture logiciel nommé XASADL (eXtensible Adaptive System Architecture Dynamic Language).

L'ADL est décrit par une DTD (Document Type Déclaration). La DTD est l'ensemble des règles et des propriétés que doit suivre le document XML. Ces règles définissent généralement le nom et le contenu de chaque balise et le contexte dans lequel elles doivent exister.

Et ci-dessous, le DTD du métamodèle proposé :

```
<? xml version="1.0" encoding="iso-8859-1" standalone="no"?>
< ! ELEMENT Component (Port+, QualityAttribute+)>
< ! ELEMENT Composite (Component*, XORComposite*, ORComposite*, Port+)>
< ! ELEMENT XORComposite (OptionalComponent*, Composite*)>
< ! ELEMENT ORComposite (Component*, Composite*)>
< ! ELEMENT Port (Role+)>
< ! ELEMENT Connector (Role+)>
< ! ELEMENT QualityAttribute (Variable*)>
< ! ELEMENT Variable Empty>
< ! ELEMENT Constraint Empty>
< ! ELEMENT Role Empty>
< ! ATTLIST Component
    InterfaceNb CDATA #REQUIRED
    Type ("Optional" | "Mandatory") #REQUIRED
    idCom ID #REQUIRED >
```

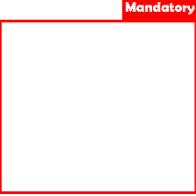
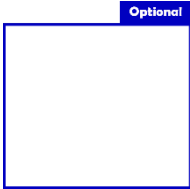
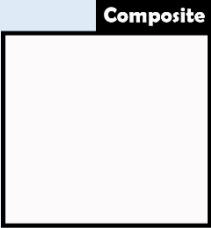
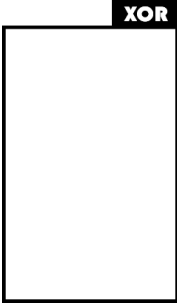
```

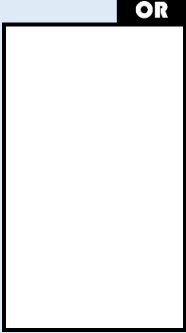





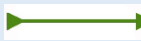

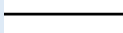

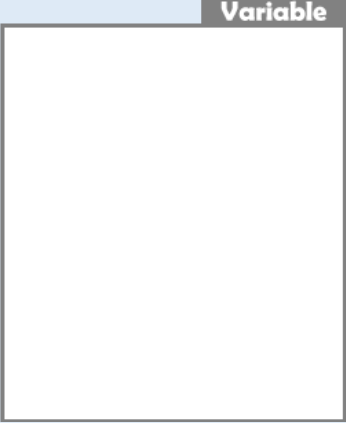
< !ATTLIST ORComposite
    CardinalNb CDATA #REQUIRED>
< !ATTLIST Port
    NamePort CDATA #REQUIRED
    Type ("RequiredMandatoryPort" | "RequiredOptionalPort" |
"ProvidedMandatoryPort" | "ProvidedOptionalPort")
#REQUIRED >
< !ATTLIST Connector
    NameCon CDATA #REQUIRED
    Type ("DelegateConn" | "AssemblyConn" | "RequiertConn" | "ExclutConn")
#REQUIRED
    idCon ID #REQUIRED >
< !ATTLIST QualityAttribute
    idState ID #REQUIRED>
< !ATTLIST Variable
    NameV CDATA #REQUIRED
    Value CDATA #REQUIRED>
< !ATTLIST Role
    NameR CDATA #REQUIRED >

```

3.2. Représentation graphique

Les notations graphiques proposées pour notre métamodèle sont représentées dans le Tableau 3.1 suivant :

Concept	Notation
<p>Component</p>	 <p>A square box with a red border. A small red rectangular label with the word "Mandatory" in white text is positioned at the top right corner of the box.</p>
	 <p>A square box with a blue border. A small blue rectangular label with the word "Optional" in white text is positioned at the top right corner of the box.</p>
<p>Composite</p>	 <p>A square box with a black border and a light pink fill. A small black rectangular label with the word "Composite" in white text is positioned at the top right corner of the box.</p>
<p>XORComposite</p>	 <p>A vertical rectangular box with a black border. A small black rectangular label with the word "XOR" in white text is positioned at the top right corner of the box.</p>

ORComposite		
Port	Required	
	Mandatory	Optional
		
	Provided	
	Mandatory	Optional
		
Connector	Exclut	Requiert
		
	Delegate	Assembly
		
QualityAttribute		
Variable		

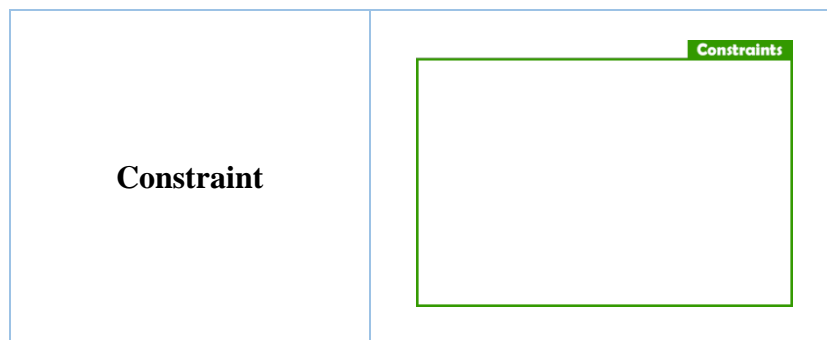


Table 3.1 : Représentation graphique des concepts

3.3. Transformation de modèle graphique au textuel

Il existe dans la littérature plusieurs approches de transformation de modèles qui appartenant à deux familles : La famille modèle vers texte (Model To Text (M2T)), et la famille modèle à Modèle (Model To Model (M2M)). Dans le Tableau 3.2 une illustration de ces deux familles.

Familles	M2M	M2T
Approches de transformation de modèles	Dirigée par la structure	Parcours de modèles (Programmation)
	Manipulation directe	
	Approche relationnelle	Template
	Transformation de graphes	
	Hybride	

Table 3.2 : Approches des transformations de modèles[40]

Pour notre approche nous nous intéressons à la transformation du modèle de spécification d'un SAA vers une représentation textuelle XASADL (voir section 3.4), nous utilisons donc l'approche Template de la famille Model To Text (M2T). Cette approche nous permettra de générer automatiquement le code XASADL à partir de notre métamodèle. L'entrée de cette transformation est le modèle obtenu à partir l'éditeur graphique que nous avons développé et le résultat et la sortie de cette transformation est le fichier XASADL. La réalisation de cette transformation sera faite dans le chapitre suivant.

3.4. Le code généré XASADL

Nous proposons de transformer le modèle obtenu vers un XASADL, nous proposons d'enrichir le fichier x3ADL (eXtensible Architecture, Aspect and Action Description Language) [41] pour obtenir notre XASADL comme indiqué ci-dessous :

```
<Configuration>
  <System name="System" Type="Composite">
    <!-- System Definition -->
    <Composite name="" Type="" InterfaceNB="" CardinalNB="" >
      <!-- Composite Definition -->
      <Ports>
        <!-- Port Definition -->
        <Port name="" Type="">
          <!-- Port Definition -->
        </Port>
      </Ports>
      <Connectors>
        <!-- Connectors Definition -->
        <Connector source="" Target="" type="">
          <!-- Connector Definition -->
        </Connector>
      </Connectors>
      <Variables>
        <!-- Variables Definition -->
        <Variable name="" value="">
          <!-- Variable Definition -->
        </ Variable >
      </ Variables >
    <Component name="" Type="" InterfaceNB="" >
```

```

<!-- Component Definition -->

<Ports>

<!-- Port Definition -->

  <Port name="" Type="">

<!-- Port Definition -->

  </Port>

</Ports>

</Component>

</Composite>

</System>

</Configuration>

```

- **La balise <Composite name="" Type="" InterfaceNB="" CardinalNB="" > :**
 Cette balise indique le nom du composite, le type (Composite, ORComposite ou XORComposite), le nombre d'instance du composite (InterfaceNB) et le nombre de cardinalité (CardinalNB) si le type du composite est 'ORComposite'.
- **La balise <Component name="" Type="" InterfaceNB="" > :**
 Cette balise indique le nom du composant, le type (Optional ou Mandatory), le nombre d'instance du composant (InterfaceNB). La balise **Component** peut être inclut dans la balise **Composite** comme il est possible d'être indépendante.
- **La balise <Ports> :**
 La balise **Ports** est une descendante de la balise **Component** ou bien la balise **Composite**. Cette balise regroupe un ensemble de balises **Port** de la forme :
<Port name="" Type="">.
 La balise **Port** indique le nom du port, le type (ProvidedMandatoryPort, ProvidedOptionalPort, RequiredMandatoryPort ou RequiredOptionalPort).
- **La balise <Connectors> :**
 La balise **Connectors** est une descendante de la balise **Composite**. Cette balise regroupe un ensemble de balises **Connector** de la forme :
La balise <Connector source="" Target="" type="">.

La balise **Connector** indique le nom du Connector, La source et la destination (Target), et le type (Assembly, Delegate).

- **La balise<Variables> :**

La balise **Variables** est une descendante de la balise **Composite**, elle contient un ensemble de balises **Variable** de la forme :

`<Variable name="" value="">`. Cette dernière indique le nom et la valeur de la variable.

4. Vérification Structurelle

La vérification c'est la 2^{ème} phase de notre approche, il s'agit de vérifier le modèle obtenu structurellement, c'est-à-dire de vérifier si les éléments sont bien structurés et de vérifier la compatibilité des valeurs de leurs attributs avec la structure du système.

4.1. Vérification automatique

Nous proposons de faire une vérification automatique, c'est-à-dire de limiter les actions faites par l'utilisateur. Dans cette section nous présentons les actions autorisées et non-autorisées que nous avons proposé pour la validation de notre modèle.

4.1.1. Les actions non-autorisées

Ce sont des actions qui peuvent éliminer ou bien changer le concept d'un ou plusieurs éléments architecturaux.

- **Exclut** : le connecteur 'Exclut' n'est pas permis dans le 'XORComposite', car son existence dans 'XORComposite' n'a aucun sens, c'est-à-dire le 'XORComposite' va choisir un et un seul sous-composite, donc le connecteur 'Exclut' n'a aucun rôle.

De plus, le 'MandatoryComponent' n'est pas permis d'être ni source ni target de ce connecteur, car le concept de 'MandatoryComponent' est d'être obligatoirement actif dans le système, et le connecteur 'Exclut' va le mettre inactif si le 2^{ème} composant lié avec ce connecteur est actif.

- **Requiert** : le concept du connecteur 'Requiert' n'est pas permis dans le 'XORComposite', car le 'XORComposite' exécute forcément un et un seul sous-composite, connecteur 'Requiert' entraîne exécution au moins de deux sous-composites.

D'autre part, le système va exécuter le 'MandatoryComponent' obligatoirement, donc le connecteur 'Requiert' n'a aucun rôle si la source et le target sont des 'MandatoryComponents'. De plus, Si nous mettons la source 'MandatoryComponent' et le target 'OptionalComponent', ce dernier va prendre le même concept que 'MandatoryComponent' et va perdre donc son propre concept.

- **MandatoryComponent** : le 'XORComposite' va choisir un seul sous-composite à exécuter, et la présence de 'MandatoryComponent' forcera son exécution en éliminant les autres sous-composites. Par conséquent le 'MandatoryComponent' n'est pas permis dans le 'XORComposite'.
- **AssemblyConnector** : le connecteur 'Assembly' peut connecter seulement deux ports de types différents (ProvidedPort avec RequiredPort), et ces deux ports appartiennent à deux composants de même classe, c'est-à-dire soit ces deux composants sont de classe 'Component' (OptionalComponent ou MandatoryComponent), soit sont de la classe 'Composite' (Composite, XORComposite ou ORComposite).
- **DelegationConnector** : le connecteur 'Delegation' peut connecter seulement deux ports de même type (ProvidedPort avec ProvidedPort ou bien RequiredPort avec RequiredPort), l'un de ces ports connectés doit appartenir à un composite X, cependant le 2^{ème} port doit appartenir à un sous-composite de X.
- **RequiredMandatoryPort** et **ProvidedMandatoryPort** : les deux types du port 'RequiredMandatoryPort' et 'ProvidedMandatoryPort' doit être actif à tout moment donné, donc ils peuvent appartenir au 'MandatoryComponent' car son mode est toujours actif. Cependant, ils ne peuvent pas appartenir à d'autres types de composants, car leur mode passe de l'état actif à l'état inactif.

4.1.2. Les actions autorisées

A part les actions non-autorisées toutes les autres actions sont autorisées, et nous allons les traiter dans la section suivante.

4.2. Vérification Programmée

Dans cette partie nous utilisons le fichier XASADL obtenu après l'exécution du code AQL de la section 3.3, pour vérifier la sémantique des actions autorisées. Pour cela, nous proposons les contraintes suivantes :

- Le nom 'Name Attribut' du composant ou bien composite doit être unique et non pas vide.
- Le nombre d'instance 'InterfaceNB' doit être strictement supérieur à 0.
- ORComposite doit respecter les contraintes suivantes :
 - Il doit contenir au moins un sous-composite à exécuter.
 - Le nombre de cardinalité 'CardinalNB' doit être compris entre 0 et le nombre de sous-composites de l'ORComposite.
- Le 'MandatoryComponent' doit être actif (propriété mode='Actif').
- Les deux types du port 'RequiredMandatoryPort' et 'ProvidedMandatoryPort' doivent être actifs (propriété mode='Actif').
- XORComposite doit contenir un et un seul composant actif à un moment donné.
- Les deux types 'RequiredMandatoryPort' et 'RequiredOptionalPort' doivent être relié au moins avec un ProvidedPort.

Pour réaliser cette vérification nous développons un programme JAVA (voir chapitre 4) qui prend en entrée le fichier XASADL obtenu d'une spécification en utilisant notre modèle de spécification des systèmes auto-adaptatif, et va par la suite faire une évaluation de ce fichier en respectant les contraintes de vérification. Enfin, le résultat de cette évaluation sera des messages d'erreurs et des messages de validation.

5. Diagramme de cas d'utilisation

Les principales fonctionnalités de l'application à concevoir sont régies autour des besoins du concepteur des systèmes auto-adaptatifs. Elles sont illustrées dans le diagramme de cas d'utilisation suivant (la figure 3.6) :

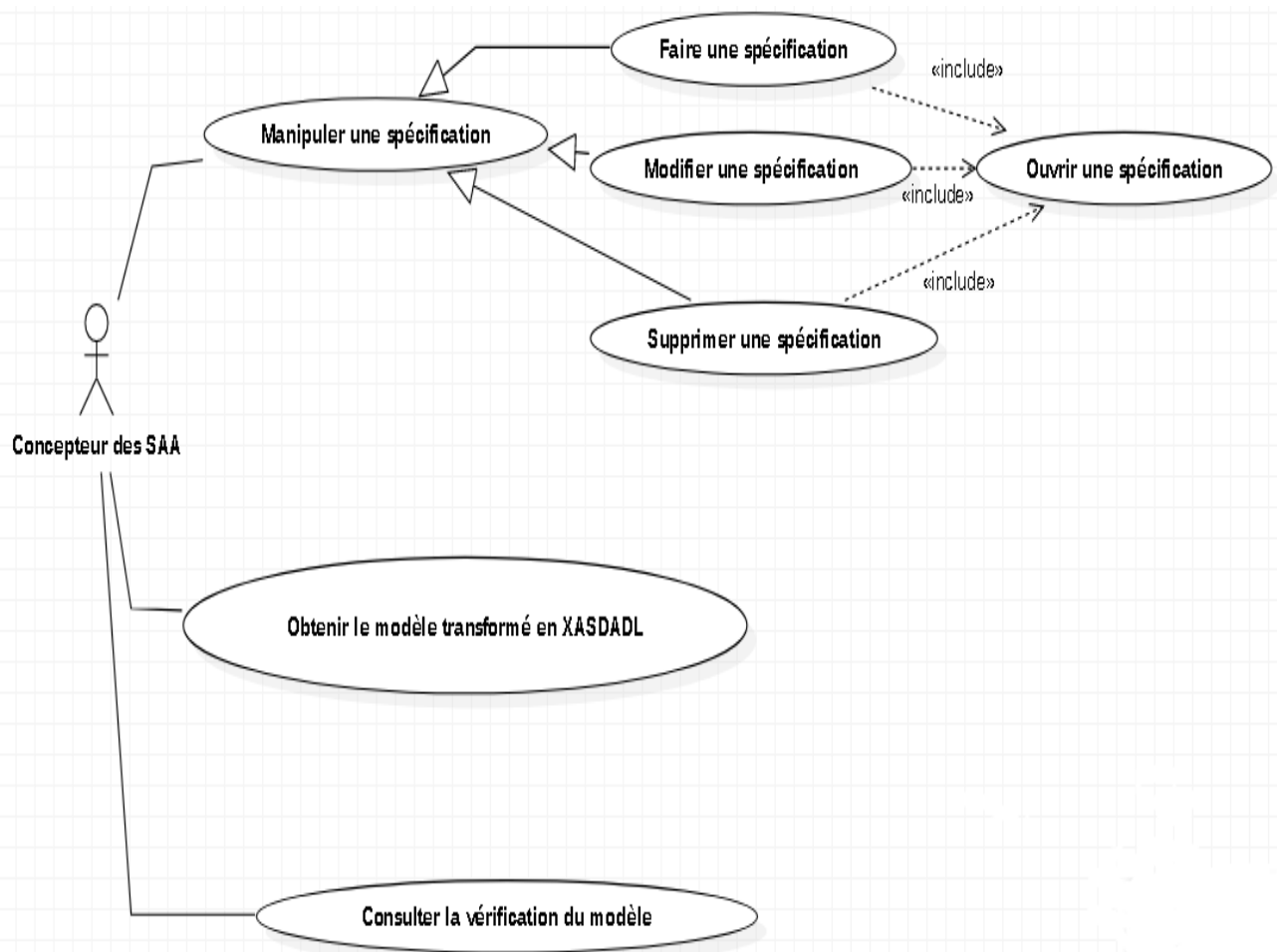


Figure 3. 6 : Diagramme de cas d'utilisation.

5.1. Description des différents Cas d'Utilisation

Manipuler une spécification : Permet au concepteur de manipuler une spécification des systèmes auto-adaptatifs. C'est-à-dire soit:

- Faire une nouvelle spécification d'un système auto-adaptatif en utilisant notre application.
- Modifier une spécification déjà faite par l'ajout, modification ou suppression des éléments en utilisant notre éditeur.
- Supprimer une spécification faite à travers notre application.

Obtenir le modèle de spécification en XASADL : Permet au concepteur de voir et obtenir la transformation de modèle obtenu après une spécification en modèle XASADL.

Consulter la vérification du modèle : Permet au concepteur de vérifier la validité de son modèle de spécification des systèmes auto-adaptatifs.

6. Conclusion

Dans ce chapitre nous avons présenté les concepts fondamentaux de notre approche en spécifiant les différents concepts du métamodèle de spécification des systèmes auto-adaptatifs, le graphisme adopté pour représenter ces concepts, nous avons par la suite détaillé la vérification du modèle obtenu de notre modèle de spécification.

Le chapitre qui suit sera consacré à l'implémentation de notre éditeur graphique et le programme de vérification du modèle obtenu. Nous présentons aussi l'environnement de notre travail, ainsi que les outils de développement utilisés.

CHAPITRE 4

L'IMPLEMENTATION DE L'EDITEUR GRAPHIQUE

1. Introduction

Après avoir décrit notre approche dans le chapitre précédent, nous allons dans ce chapitre présenter les étapes d'implémentation de notre éditeur graphique, ainsi que la description de l'environnement et les outils de travail utilisés pour atteindre nos objectifs.

2. L'environnement et outils de travail

Le choix des outils dans le cadre de notre travail de fin d'étude n'été pas facile vue le manque de documentations. Cette section montre l'environnement que nous avons choisi ainsi que sa description.

2.1. L'éditeur de logiciels Obeo

Obeo¹ est un éditeur de logiciels, spécialisé dans l'approche model-driven, et un key-player de la plateforme Eclipse. C'est un acteur de l'écosystème Open Source et un membre stratégique de la Fondation Eclipse.

Sa gamme de produits comprend : Sirius (Graphical Designers), Acceleo (générateur de code), Obeo Designer (environnement de modélisation graphique personnalisé), add-ons collaboratifs pour Eclipse Capella, Obeo SmartEA (Enterprise Architecture), consulting (formation, expertise et coaching), et Eclipse Modeling en tant que pile de base de tout environnement de modélisation. Dans notre cas on utilise **Obeo Designer** la version 11.4 avec **Sirius** et **Acceleo**.

2.2. Sirius

Sirius² est un projet open-source Eclipse qui permet de créer facilement un outil de modélisation graphique. Il s'appuie sur les technologies de modélisation Eclipse,

¹ https://www.eclipse.org/membership/showMember.php?member_id=863

² <https://marketplace.eclipse.org/node/1318261>

notamment **EMF** (**Eclipse Modeling Framework**) pour la gestion des modèles et **GMF** (**Graphic Modeling Framework**) pour la représentation graphique.

Sirius permet d'équiper les équipes qui doivent traiter des architectures complexes sur des domaines spécifiques. Il est particulièrement adapté aux utilisateurs ayant défini un DSL (**Domain Specific Language**) et ayant besoin de représentations graphiques pour mieux élaborer et analyser un système et améliorer la communication avec les autres membres de l'équipe, partenaires ou clients, la version que nous avons utilisée est 6.4.

2.3. Acceleo

Acceleo³ est un générateur de code source de la fondation Eclipse permettant de mettre en œuvre l'approche **MDA** (**Model Driven Architecture**) pour réaliser des applications à partir de modèles basés sur **EMF**. Il s'agit d'une implémentation de la norme **MOF**⁴ (**Models to Text Transformation Language**) **Models to Text (MOFM2T)** de l'**Object Management Group (OMG)** pour les transformations de modèles à texte, la version que nous avons utilisée est 3.7.9.

2.4. Eclipse IDE

Eclipse⁵ **IDE**, **I**ntegrated **D**evelopment **E**nvironment (**EDI** environnement de développement intégré en français), c'est-à-dire un logiciel qui simplifie la programmation en proposant un certain nombre de raccourcis et d'aide à la programmation. Il est développé par **IBM** (**I**nternational **B**usiness **M**achines), est gratuit et disponible pour la plupart des systèmes d'exploitation, nous avons utilisé la version 2021-06 l et la version du **JDK** est 11.8.0.

2.5. Langage d'implémentation JAVA

Java⁶ est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy, présenté officiellement le 23 mai 1995 au SunWorld.

Une particularité de Java est que les logiciels écrits dans ce langage sont compilés vers une représentation binaire intermédiaire qui peut être exécutée dans une machine virtuelle Java (**JVM**) en faisant abstraction du système d'exploitation.

³ <https://fr.wikipedia.org/wiki/Acceleo>

⁴ https://fr.wikipedia.org/wiki/MOF_Models_to_Text_Transformation_Language

⁵ <https://dept-info.labri.fr/ENSEIGNEMENT/programmation2/intro-eclipse/>

⁶ [https://fr.wikipedia.org/wiki/Java_\(langage\)](https://fr.wikipedia.org/wiki/Java_(langage))

2.6. XML

L'Extensible Markup Language, généralement appelé XML⁷, « langage de balisage extensible1 » en français, est un métalangage informatique de balisage générique qui est un sous-ensemble du Standard Generalized Markup Language (SGML).

Sa syntaxe est dite « extensible » car elle permet de définir différents langages avec pour chacun son vocabulaire et sa grammaire, comme XHTML, XSLT, RSS, SVG... Elle est reconnaissable par son usage des chevrons (<, >) encadrant les noms des balises. L'objectif initial de XML est de faciliter l'échange automatisé de contenus complexes (arbres, texte enrichi, etc.) entre systèmes d'informations hétérogènes (interopérabilité), pour notre projet nous avons choisi la version 1.0.

2.7. JDOM

JDOM⁸ (acronyme de l'anglais Java Document Object Model), est une bibliothèque open source pour manipulation des fichiers XML en Java. Elle intègre DOM et SAX, et supporte XPath et XSLT. Elle utilise des analyses syntaxiques externes pour construire les documents, pour notre projet nous avons choisi la version 2.0.6.

2.8. Langage AQL

Le langage de requête Acceleo (AQL⁹) est un langage utilisé pour naviguer et interroger un EMF. AQL en tant que moteur de requête est petit, simple, rapide, extensible et apporte une validation plus riche.

3. Implémentation

Nous avons choisi Sirius pour implémenter notre éditeur graphique. Sirius nécessite trois modèles de base en entrée décrivant les différents aspects d'un éditeur graphique. Pour réaliser notre éditeur graphique nous avons passé par deux phases :

3.1. Création d'un projet EMF

Cette phase consiste à produire notre métamodèle ce qui permet par la suite d'implémenter l'éditeur graphique. Dans cette section nous détaillons les étapes de cette phase.

⁷ https://fr.wikipedia.org/wiki/Extensible_Markup_Language

⁸ <https://fr.wikipedia.org/wiki/JDOM>

⁹ <https://www.eclipse.org/acceleo/documentation/>

3.1.1. Création Le model de domaine (*.ecore)

C'est le métamodèle décrivant le domaine de spécification des systèmes auto-adaptatifs (Figure 4.1). Il a été importé à partir de notre métamodèle que nous avons défini dans le chapitre précédent. Il présente les concepts, les attributs, et les relations entre les concepts. Le métamodèle est présenté de façon hiérarchique.

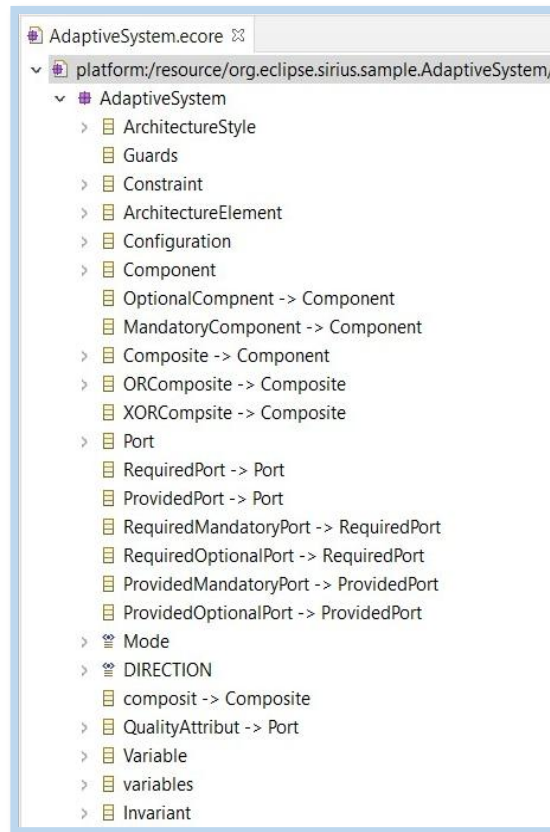


Figure 4.1 : Le modèle de domaine de l'éditeur graphique

3.1.2. Le model de génération (*.genmodel)

Contient des informations sur la génération de code.

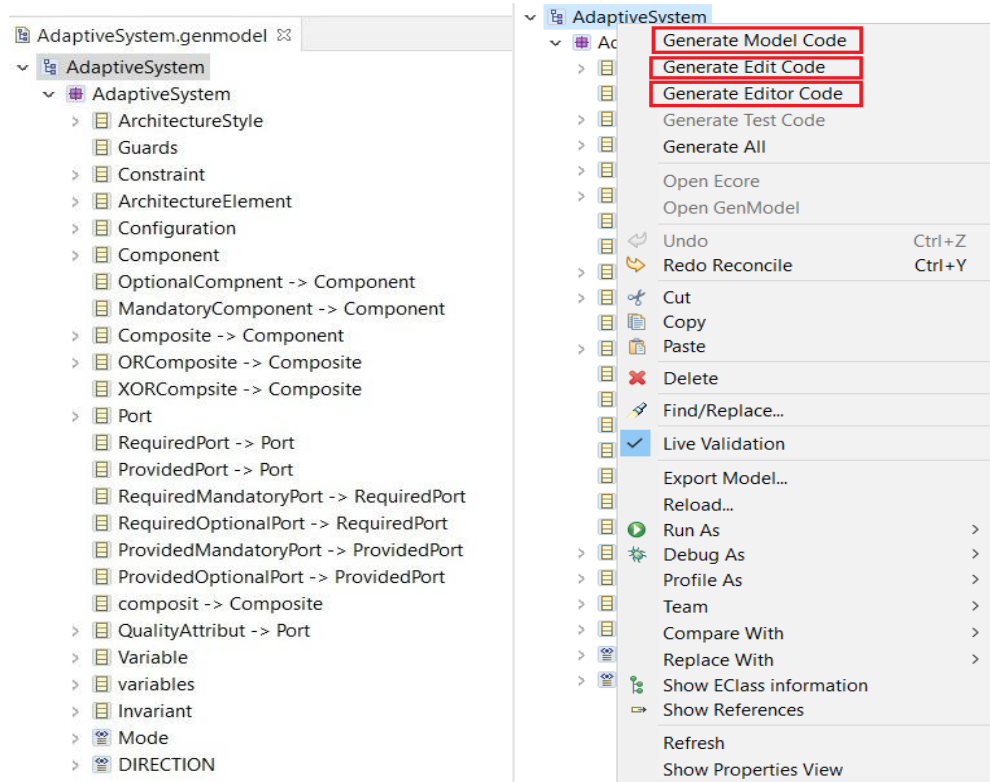


Figure 4.2 : Le modèle de génération de l'éditeur

Les actions du menu contextuel generate code (Figure 4.2) :

- L'action Generate Model Code produit du code Java pour implémenter le modèle
- L'action Generate Edit Code produit un code adaptateur pour prendre en charge les visionneuses
- L'action Generate Editor Code produit un éditeur Eclipse complet
- L'action Generate All produit les trois.

Après avoir générer les trois types du code, nous exécutons le projet pour pouvoir entamer à la 2^{ème} phase de l'implémentation.

3.1.3. Representation Data (*.aird)

Les fichiers (*.aird) (Figure 4.3) qui contient des données de représentations Sirius sont également stockés en tant que modèles.

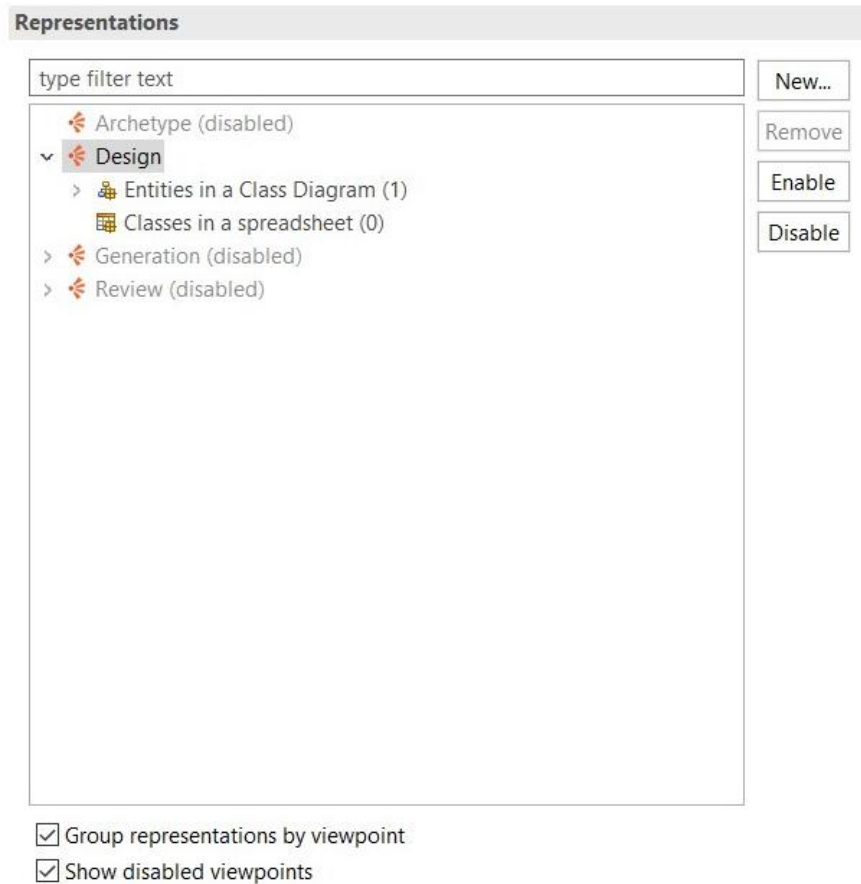


Figure 4.3 : Le modèle Representation Data

3.2. Implémentation de l'éditeur

Après l'exécution du projet Sirius dans la partie Runtime nous créons notre éditeur graphique à travers fichier odesign. Dans cette section nous détaillons l'implémentation de l'éditeur.

3.2.1. Modèles de spécification de point de vue (*.odesign)

Ce fichier décrit le plan de travail de la modélisation que nous avons créé déjà dans projet Sirius. Ce fichier est créé dans Viewpoint Specification Project dans la partie Runtime, il contient toute la description de l'éditeur graphique de notre approche (voir la Figure 4.4), ce fichier est créé dans un nouveau projet ViewPoint Specification Project.

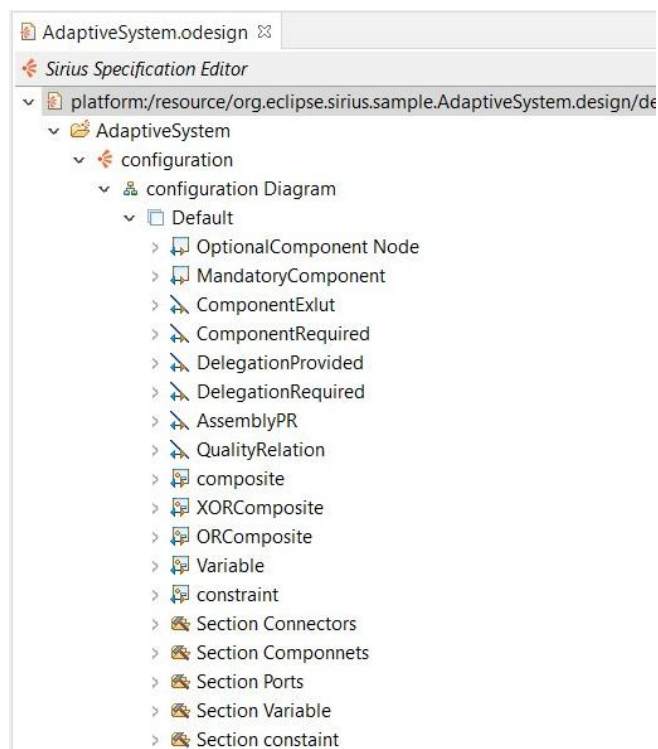


Figure 4.4 : Le fichier Odesign qui représente le plan de travail de la modélisation

3.2.2. Modeling Project

Il faut créer un projet Modeling Project, que doit contenir au moins un métamodèle et une représentation pour créer une extension de notre diagramme. La Figure 4.5 montre comment le projet est créé.

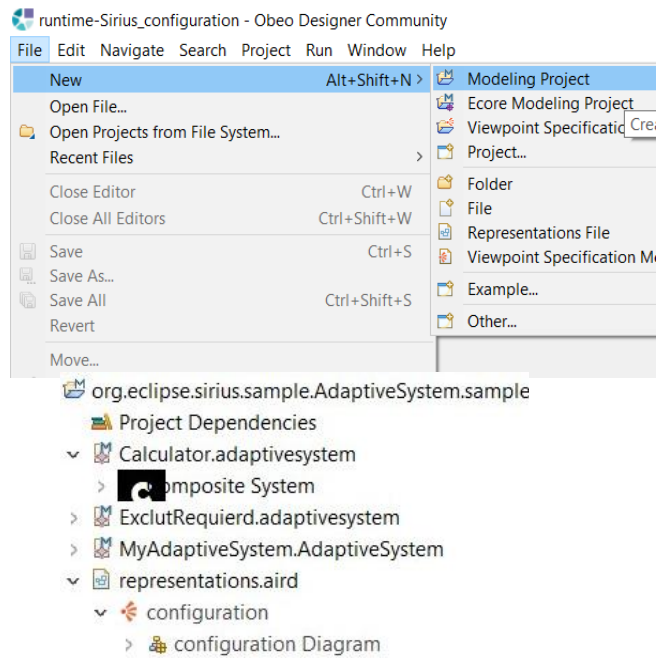


Figure 4.5 : La création d'un projet Modeling Project

4. Présentation de l'éditeur graphique

L'idée principale derrière le développement de cet éditeur graphique est de donner une aide à la spécification des systèmes auto-adaptatifs selon notre approche.

Notre éditeur graphique est composé de 5 parties (voir la Figure 4.6) :

- La 1^{ère} partie Model Explorer ou nous pouvons créer un nouveau diagramme instancié du notre model.
- La 2^{ème} partie c'est un espace vide permettent de la représentation et la modification du diagramme c'est-à-dire la modification graphique du modèle.
- La 3^{ème} partie c'est la palette qui contient les éléments graphiques de création des concepts.
- La 4^{ème} partie c'est la vue Outline : donne un aperçu général du modèle (lecture seule)
- La 5^{ème} partie c'est la vue de propriétés : permet le renseignement des propriétés des éléments du modèle également celles non représentées sur le diagramme.

The image displays the Eclipse Sirius editor interface, divided into several key areas:

- 1. Type Filter:** Located at the top left, it shows a search bar with the text "type filter text" and a list of project paths:
 - > org.eclipse.acceleo.module.sample1
 - > org.eclipse.sirius.sample.AdaptiveSystem
 - > org.eclipse.sirius.sample.AdaptiveSystem
- 2. Main Diagram:** The central workspace contains a complex diagram of a composite system. It features three main containers:
 - Composite:** Contains three mandatory components labeled A, B, and C.
 - XOR Composite:** Contains two optional components, each represented by a box with a "+" sign.
 - OR Composite:** Contains three optional components: "Textuel", "graphical", and "Audit".
 The components are interconnected using various connector types, such as "Exclut" (red arrow) and "Requiert" (green arrow).
- 3. Palette:** Located on the right side, it provides a library of elements for the diagram:
 - Connectors:** Exclut (red arrow), Requiert (green arrow), and DelegationProvided.
 - Componnets:** OptionalComponent (blue square with 'O'), MandatoryComponent (red square with 'M').
 - Ports:** ProvidedOptionalPort, ProvidedMandatoryport, RequiredOptionalport.
 - Variable:** Variable (blue square with 'V'), Attribute (grey triangle).
 - constant:** Invariant (green diamond), Constraint (green square with 'C').
- 4. Outline:** Located in the bottom left, it shows a hierarchical tree view of the diagram's structure.
- 5. Properties Panel:** Located at the bottom, it displays the configuration for the selected "Composite System":
 - Name:** System
 - Mode Component:** Inactif (radio button) / **Actif** (radio button)
 - Interface NB:** 0

Figure 4.6 : Vue d'ensemble de l'éditeur

5. La transformation du modèle

Pour réaliser la transformation M2T, Nous avons utilisé l'outil Acceleo pour développer le code AQL (Acceleo Query Language) qui est un langage utilisé pour naviguer et interroger un métamodèle, ce langage utilise une approche par Template.

5.1. Les règles de transformation

Nous avons défini des règles de transformations pour pouvoir transformer le modèle après une spécification à un fichier XASADL. Le tableau 4.1 montre ses règles de transformation. Nous avons implémenté ses règles avec le langage AQL.

Concept	Transformation
Composite	<code><Composite Name="" Type="composite" InterfaceNB="" Mode=""></code>
ORComposite	<code><Composite Name="" Type="ORComposite" InterfaceNB="" CardinalNB="" Mode=""></code>
XORComposite	<code><Composite Name="" Type="XORCompsite" InterfaceNB="" Mode=""></code>
MandatoryComponent	<code><Component Name="" Type="MandatoryComponent" InterfaceNB="" Mode=""></code>
OptionalComponent	<code><Component Name="" Type="OptionalComponent" InterfaceNB="" Mode=""></code>
Port	<code><Port Name="" Type="" Mode="" /></code>
Connector	<code><Connector Source="" Target="" Type="" /></code>
Variable	<code><Variable Name="" Type="Variable" /></code>
Constraint	<code><Constraint Name="" Type="Constraint" /></code>

Table 4.1 : Les règles de transformation pour transformer le modèle après une spécification à un fichier XASADL.

5.2. Le code AQL

La Figure 4.7 illustre le code AQL que nous avons développé pour générer un Code XML à partir de notre métamodèle afin de générer la représentation textuelle XASADL :

```

[comment encoding = UTF-8 /]
[**
 * The documentation of the module generate.
 */]
[module generate('http://www.example.org/AdaptiveSystem')]
[**
 * The documentation of the template GenerateSAA.
 * @param aComposite
 */]

[template public GenerateSAA(aComposite : Composite)]
[comment @main/]
[file (aComposite.name.concat('XMLGenerate.xml'), false, 'UTF-8')]
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
<System Name="[ aComposite.name/ ]" Type="[ aComposite.eClass().name/ ]" >

[comment BIG Simple composite /]
[ for ( cpnt : composi | aComposite.composites ) ]
<Composite Name="[ cpnt.name/ ]" Type="[cpnt.eClass().name/ ]"
InterfaceNB="[cpnt.InterfaceNB/ ]">
[ for ( port : Port | cpnt.ports ) ]
<Port Name="[ port.name/ ]" Type="[ port.eClass().name/ ]" />
[/for]

[ for ( port : Port | cpnt.ports ) ]
[ for ( port1 : Port | port.delegationPort ) ]
<Connector Source="[ port.name/ ]" Target="[ port1.name/ ]"
Type="Delegate" />
[/for] [/for]

[ for ( cpnt3 : Component | cpnt.components ) ]
<Component Name="[ cpnt3.name/ ]" Type="[cpnt3.eClass().name/ ]"
InterfaceNB="[cpnt.InterfaceNB/ ]">
[ for ( port : Port | cpnt3.ports ) ]
<Port Name="[ port.name/ ]" Type="[ port.eClass().name/ ]" />
[/for]
</Component>
[/for]

[ for ( cpnt2 : Component | cpnt.components ) ]
[ for ( port : Port | cpnt2.ports ) ]
[ for ( port1 : Port | port.AssemblyPR ) ]
<Connector Source="[ port.name/ ]" Target="[ port1.name/ ]"
Type="Assembly" />
[/for]
[/for]

```

A

```

[ for ( att1 : Variable | cpnt.variables ) ]
<Variables Name="[att1.name/ ]">
[ for ( att2 : variables | att1.variables ) ]
<Variable Name="[ att2.name/ ]" Value="[ att2.Value/ ]" />
[/for]</Variables>[/for]

[comment Simple composite /]
[ for ( cpnt1 : composi | cpnt.composites ) ]
<Composite Name="[ cpnt1.name/ ]" Type="[cpnt1.eClass().name/ ]"
InterfaceNB="[cpnt.InterfaceNB/ ]">
[ for ( port : Port | cpnt1.ports ) ]
<Port Name="[ port.name/ ]" Type="[ port.eClass().name/ ]" />
[/for]

[ for ( port : Port | cpnt1.ports ) ]
[ for ( port1 : Port | port.delegationPort ) ]
<Connector Source="[ port.name/ ]" Target="[ port1.name/ ]"
Type="Delegate" />
[/for] [/for]

[ for ( cpnt2 : Component | cpnt1.components ) ]
[ for ( port : Port | cpnt2.ports ) ]
[ for ( port1 : Port | port.AssemblyPR ) ]
<Connector Source="[ port.name/ ]" Target="[ port1.name/ ]"
Type="Assembly" />
[/for]
[/for]

[ for ( cpnt2 : Component | cpnt1.components ) ]
<Component Name="[ cpnt2.name/ ]" Type="[cpnt2.eClass().name/ ]"
InterfaceNB="[cpnt.InterfaceNB/ ]">
[ for ( port : Port | cpnt2.ports ) ]
<Port Name="[ port.name/ ]" Type="[ port.eClass().name/ ]" />
[/for]
</Component>
[/for]
</Composite>
[/for]

[comment XOR composite /]
[ for ( cpnt1 : XORComposit | cpnt.composites ) ]
<Composite Name="[ cpnt1.name/ ]" Type="[cpnt1.eClass().name/ ]"
InterfaceNB="[cpnt.InterfaceNB/ ]">
[ for ( port : Port | cpnt1.ports ) ]
<Port Name="[ port.name/ ]" Type="[ port.eClass().name/ ]" />
[/for]

[ for ( port : Port | cpnt1.ports ) ]
[ for ( port1 : Port | port.delegationPort ) ]
<Connector Source="[ port.name/ ]" Target="[ port1.name/ ]"
Type="Delegate" />
[/for] [/for]

```

B

```

[ for ( cpnt2 : Component | cpnt1.components ) ]
[ for ( port : Port | cpnt2.ports ) ]
[ for ( port1 : Port | port.AssemblyPR ) ]
<Connector Source="[ port.name/ ]" Target="[ port1.name/ ]"
Type="Assembly" />
[/for]
[/for]
[/for]

[ for ( cpnt2 : Component | cpnt1.components ) ]
<Component Name="[ cpnt2.name/ ]" Type="[cpnt2.eClass().name/ ]"
InterfaceNB="[cpnt.InterfaceNB/ ]">
[ for ( port : Port | cpnt2.ports ) ]
<Port Name="[ port.name/ ]" Type="[ port.eClass().name/ ]" />
[/for]
</Component>
[/for]
</Composite>
[/for]

[comment OR composite /]
[ for ( cpnt1 : ORComposite | cpnt.composites ) ]
<Composite Name="[ cpnt1.name/ ]" Type="[cpnt1.eClass().name/ ]"
InterfaceNB="[cpnt.InterfaceNB/ ] CardinalNB="[cpnt1.CardinalNB/ ]">
[ for ( port : Port | cpnt1.ports ) ]
<Port Name="[ port.name/ ]" Type="[ port.eClass().name/ ]" />
[/for]
[ for ( port : Port | cpnt1.ports ) ]
[ for ( port1 : Port | port.delegationPort ) ]
<Connector Source="[ port.name/ ]" Target="[ port1.name/ ]"
Type="Delegate" />
[/for] [/for]

[ for ( cpnt2 : Component | cpnt1.components ) ]
[ for ( port : Port | cpnt2.ports ) ]
[ for ( port1 : Port | port.AssemblyPR ) ]
<Connector Source="[ port.name/ ]" Target="[ port1.name/ ]"
Type="Assembly" />
[/for]
[/for]
[/for]

[ for ( cpnt2 : Component | cpnt1.components ) ]
<Component Name="[ cpnt2.name/ ]" Type="[cpnt2.eClass().name/ ]"
InterfaceNB="[cpnt.InterfaceNB/ ]">
[ for ( port : Port | cpnt2.ports ) ]
<Port Name="[ port.name/ ]" Type="[ port.eClass().name/ ]" />
[/for]
</Component>
[/for]
</Composite>
[/for]

```

C

D	<pre> [comment END /] [for (port : Port aComposite.ports)] <Port Name="[port.name/] Type="[port.eClass().name/]" /> [/for] [for (cpnt : Composite aComposite.composites)] [for (port : Port cpnt.ports)] [for (port1 : Port port.AssemblyPR)] <Connector Source="[port.name/]" Target="[port1.name/]" Type="Assembly" /> [/for] [/for] [/for] </System> </Configuration> [/file] [/template] </pre>
---	--

Figure 4. 7 : Le Template de base proposée pour générer un code XML.

Le code AQL est structuré par les éléments suivants :

- **Module** : Il s'agit de la déclaration du métamodèle à utiliser pour interpréter le modèle source. Le script : **[module module_name ('Métamodel URI)]** montre que le module a un nom et un URI du métamodèle utilisé.
- **Template** : Contient le texte à générer et des instructions à remplacer par des données, il est délimité par les balises [template] [/template]. Le script : **[template public Template_name (C : Class)]** montre que le Template a un nom et une signature, où **C** est un attribut et **Classe** est le type de cet attribut. L'existence de scripte **[comment @main /]** dans le Template indique que ce dernier est Template principal.
- **File** : indique le fichier de sortie, qui doit être spécifié. Il est délimité par les balises [file] [/file]. Son script est : **[file (<uri_expression>, <append_mode>, '<output_encoding>')] (...)** [/file] dont :
 - **uri_expression** indique le nom du fichier de sortie.
 - **append_mode** (optionnel) indique si le texte de sortie doit être ajouté au fichier généré ou bien doit remplacer son contenu.

- **'output_encoding'** (optionnel) indique l'encodage à utiliser pour le fichier de sortie.
- **For** : Le bloc For est une boucle, il existe deux syntaxes dans AQL pour exprimer les boucles For :
 - **Full syntax** : `[for (i : E | e)] [/for]`, où **i** est une variable de type **E**, et la boucle va produire le texte entre `[for]` et `[/for]` pour chaque attribut de la collection **e**.
 - **Light syntax** : `[for (<iterable_expression>)] [/for]`, Lors de l'utilisation de Light syntax, une variable implicite **i** est créée, qui contient l'indice de l'itération en cours, en commençant à 1.

Dans notre code, nous avons utilisé la première syntaxe, Full syntax.

6. Le résultat du code java

Nous avons développé un code java sous Eclipse IDE qui permet la vérification du modèle obtenu après sa spécification. Cette vérification est faite grâce à la bibliothèque JDOM qui permet de parcourir et extraire les informations du fichier XASADL obtenu après la transformation du modèle qui est déjà expliqué dans la section précédente. L'exécution de ce code consiste à deux types de messages (messages d'erreurs et messages de validation). La Figure 4.8 illustre les messages d'erreurs qui peuvent se produire, d'autre part la Figure 4.9 illustre les messages de validation.

```

Error : The Mode of Textuel is unacceptable
Error : The Mode of P1P-OR-M PORT is unacceptable
Error : The Cardinal number doesn't respect the rule of cardinals numbers 3
Error : The Interface Number 0 is unacceptable
Error : The name Textuel is duplicate
Error : ORComposite should have at least one Component , Component = 0
Error : The XOR Composite Operation has more than one compo Actif

```

Figure 4.8 : Les messages d'erreurs

```

OK : The Mode of Textuel is acceptable
OK : The Mode of P1P-OR-M PORT is acceptable
OK : The Cardinal number respect the rule of cardinals numbers 3
OK : The Interface Number 2 is acceptable
OK : The ORComposite have at least one Component , Component = 3
OK : The name resultat is not duplicate
OK : The XOR Composite Operation has one compo Actif

```

Figure 4.9 : Les messages de validation

7. Exemples illustratifs

Pour illustrer les différents éléments proposés dans notre approche nous proposons les exemples suivants :

7.1. Système de calcul simple

C'est un système de calcul l'opération $(A+B) * C$ et afficher le résultat. Il est construit de trois composites :

- Le 1^{er} composite pour la lecture des entrés A, B et C.
- Le 2^{ème} composite pour faire le calcul simple.
- Le 3^{ème} composite pour afficher le résultat.

7.1.1. Spécification du système

La Figure 4.10 représente le système du calcul simple spécifié par notre modèle. Il est composé de trois composites :

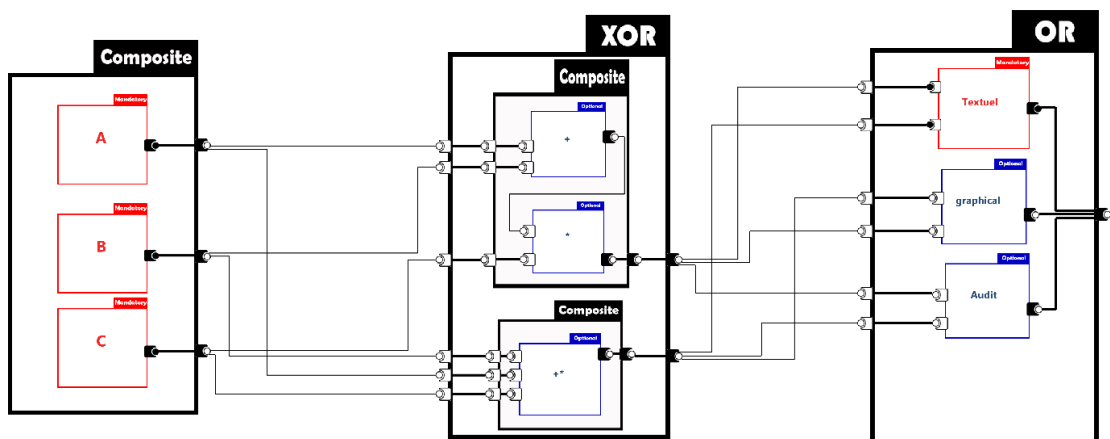


Figure 4.10 : Présentation de spécification 'Système du calcul simple'

Le premier composite représenté par la Figure 4.11, ce composite nommé ‘Lecture’ (voir Figure 4.15) permet de lire les entrées des variables A, B et C.

Nous avons mis les composants de lecture des entrées comme ‘MandatoryComponent’, cela veut dire que les 3 composants A, B et C sont actifs et seront exécutés par le système. Le résultat de la lecture sera envoyé au composite ‘Operation’ à travers les ‘ProvidedMandatoryPorts’ en utilisant les ‘AssemblyConnectors’. Les ‘ProvidedMandatoryPorts’ que nous avons utilisés sont actifs dans le système.

Les interfaces des composants et les interfaces du composite sont reliées entre eux par ‘DelegateConnector’.

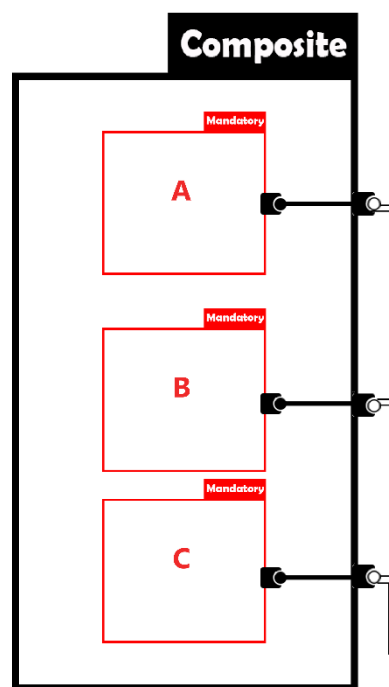


Figure 4.11 : Présentation du Composite ‘Lecture’

Le deuxième composite représenté par la Figure 4.12, ce composite nommé ‘Operation’ permet de faire une opération de calcul de deux manières. Chaque façon est représenté par un sous-composite.

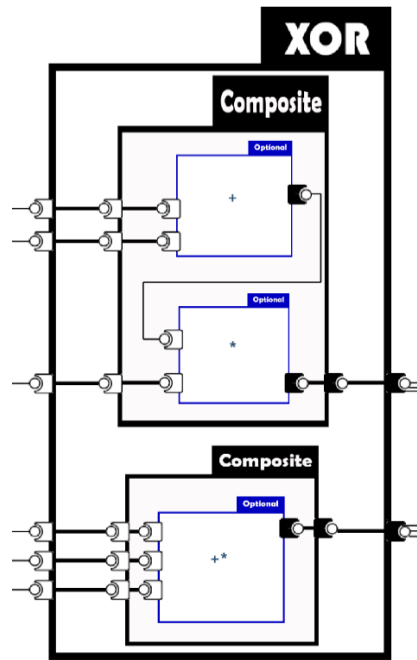


Figure 4.12 : Présentation du XORComposite 'Operation'

Nous avons choisi 'XORComposite' comme type de ce composite, c'est-à-dire un de ses sous-composite sera exécuté. et nous avons proposé la structure du composite 'Operation' comme suit :

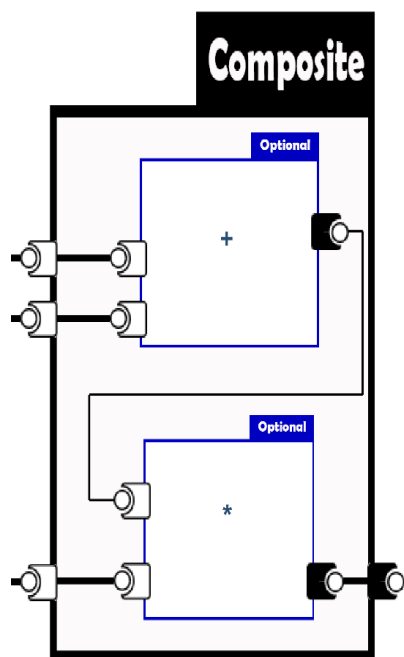


Figure 4.13 : Présentation du 1er Sous-Composite

- Le 1^{er} sous-composite (Figure 4.13) nommé 'comp1', il est composé de deux composants de type 'OptionalComponent' : le 1^{er} composant nommé '+' reçoit le résultat de la lecture A et B par 'RequiredOptionalPorts' et fait l'opération

d'addition $A+B$. le résultat par la suite sera envoyé au 2^{ème} composant nommé '*' à travers 'ProvidedOptionalPort' en utilisant 'AssemblyConnector', '*' reçoit le résultat de la lecture du C et complète le calcul par l'opération de multiplication $(A+B) * C$.

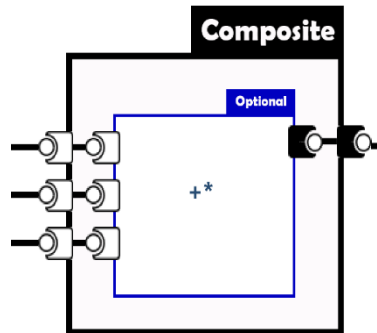


Figure 4.14 : Présentation du 2^{ème} Sous-Composite

- Le 2^{ème} composite nommé 'compo2' présenté par la Figure 4.14 composé un composant de type 'OptionalComponent', ce composant reçoit le résultat de lecture A, B et C, ensuite il fait l'opération d'addition $(A + B)$ et l'opération de multiplication $(A + B) * C$.

Le résultat du calcul fait par le composite 'Operation' est envoyé au 3^{ème} composite Affichage.

- Le troisième composite représenté par la Figure 4.15, ce composite nommé 'Affichage' permet d'afficher le résultat de calcul selon le désir de l'utilisateur. Il est composé de 3 sous-composite d'affichage (Textuel, Graphical et Audit).

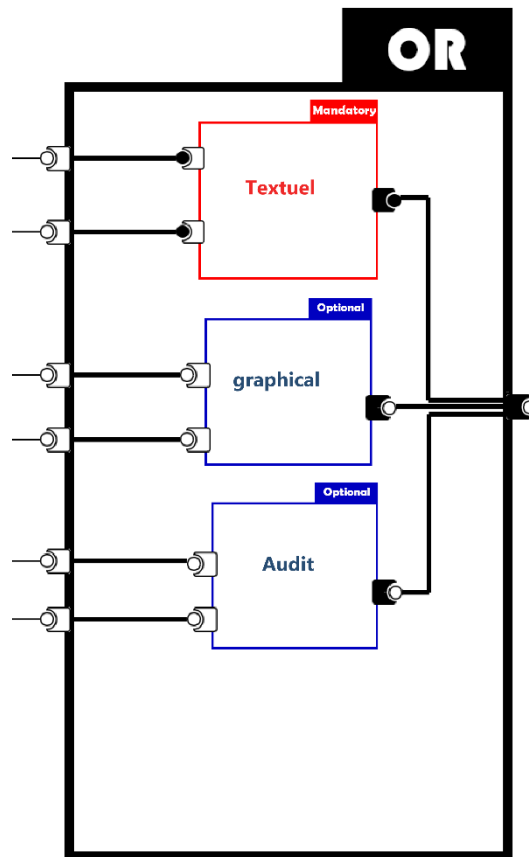


Figure 4.15 : Présentation du ORComposite 'Affichage'

- Nous avons choisi le 'ORComposite' comme un type de ce composite cela veut dire qu'il peut exécuter plusieurs sous-composite selon le 'NBCardinal' donné par l'utilisateur.

Nous avons met trois sous-composants dans le composant 'Affichage' :

- Composant Textuel : il reçoit le résultat et l'affiche sous forme un texte, nous avons mis son type 'MandatoryComponent', c'est-à-dire il sera exécuté obligatoirement par le système et nous avons choisi les ports de type obligatoire pour qu'ils soient actifs à tout moment.
- Composant Graphical : il reçoit le résultat et l'affiche sous forme un graphe.
- Composant Audit : il reçoit le résultat et l'affiche sous forme vocal.

Nous avons choisi le type des deux composant 'Graphical' et 'Audit' leur type est 'OptionalComponent', ils peuvent être exécutés par le système comme ils ne peuvent pas.

7.1.2. Fichier XASADL obtenu :

Dans cette partie nous présentons le fichier XASADL (voir la Figure 4.16) obtenu à partir la transformation de modèle que nous avons déjà détaillé dans les sections précédentes. Le fichier AQL utilisé pour obtenir ce code sera présenté dans le chapitre suivant.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configuration>
3
4 <System Name="System" Type="Composite" >
5
6 <Composite Name="Lecture" Type="composit" InterfaceNB="0" Mode="Actif">
7 <Port Name="P1P-CS" Type="ProvidedOptionalPort" Mode="Inactif" />
8 <Port Name="P3P-CS" Type="ProvidedOptionalPort" Mode="Inactif" />
9 <Port Name="P2P-CS" Type="ProvidedOptionalPort" Mode="Inactif" />
10
11 <Connector Source="P1P-CS" Target="P4P-CS-M" Type="Delegate" />
12 <Connector Source="P3P-CS" Target="ProvidedMandatoryPort" Type="Delegate" />
13 <Connector Source="P2P-CS" Target="ProvidedMandatoryPort" Type="Delegate" />
14
15 <Component Name="A" Type="MandatoryComponent" InterfaceNB="0" Mode="Actif">
16 <Port Name="P4P-CS-M" Type="ProvidedMandatoryPort" Mode="Inactif" />
17 </Component>
18 <Component Name="B" Type="MandatoryComponent" InterfaceNB="0" Mode="Actif">
19 <Port Name="ProvidedMandatoryPort" Type="ProvidedMandatoryPort" Mode="Inactif" />
20 </Component>
21 <Component Name="C" Type="MandatoryComponent" InterfaceNB="0" Mode="Actif">
22 <Port Name="ProvidedMandatoryPort" Type="ProvidedMandatoryPort" Mode="Inactif" />
23 </Component>
24 </Composite>
25
26
27 <Composite Name="Operation" Type="XORCompsite" InterfaceNB="0" Mode="Actif">
28 <Port Name="P1R-XOR" Type="RequiredOptionalPort" Mode="Inactif" />
29 <Port Name="P2R-XOR" Type="RequiredOptionalPort" Mode="Inactif" />
30 <Port Name="P4R-XOR" Type="RequiredOptionalPort" Mode="Inactif" />
31 <Port Name="P5R-XOR" Type="RequiredOptionalPort" Mode="Inactif" />
32 <Port Name="P6R-XOR" Type="RequiredOptionalPort" Mode="Inactif" />
33 <Port Name="P3R-XOR" Type="RequiredOptionalPort" Mode="Inactif" />
34 <Port Name="P1P-XOR" Type="ProvidedOptionalPort" Mode="Inactif" />
35 <Port Name="P2P-XOR" Type="ProvidedOptionalPort" Mode="Inactif" />
36
37 <Connector Source="P1R-XOR" Target="P1R-XOR-CS" Type="Delegate" />
38 <Connector Source="P2R-XOR" Target="P2R-XOR-CS" Type="Delegate" />
39 <Connector Source="P3R-XOR" Target="P3R-XOR-CS" Type="Delegate" />
40 <Connector Source="P4R-XOR" Target="P4R-XOR-CS" Type="Delegate" />
41 <Connector Source="P5R-XOR" Target="P5R-XOR-CS" Type="Delegate" />
42 <Connector Source="P6R-XOR" Target="P6R-XOR-CS" Type="Delegate" />
43
44 <Composite Name="Compo1" Type="composit" InterfaceNB="0" Mode="Actif">
45 <Port Name="P1R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif" />
46 <Port Name="P2R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif" />
47 <Port Name="P3R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif" />
48 <Port Name="P1P-XOR-CS" Type="ProvidedOptionalPort" Mode="Inactif" />
49
50 <Connector Source="P1R-XOR-CS" Target="P1R-XOR-CS-0" Type="Delegate" />
51 <Connector Source="P2R-XOR-CS" Target="P2R-XOR-CS-0" Type="Delegate" />
52 <Connector Source="P3R-XOR-CS" Target="P4R-XOR-CS-0" Type="Delegate" />
53 <Connector Source="P1P-XOR-CS" Target="P1P-XOR" Type="Delegate" />
54 <Connector Source="P1P-XOR-CS-0" Target="P3R-XOR-CS-0" Type="Assembly" />
55
56 <Component Name="+" Type="OptionalCompnent" InterfaceNB="0" Mode="Inactif">
57 <Port Name="P2R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif" />
58 <Port Name="P1R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif" />
59 <Port Name="P1P-XOR-CS-0" Type="ProvidedOptionalPort" Mode="Inactif" />
60 </Component>
61 <Component Name="*" Type="OptionalCompnent" InterfaceNB="0" Mode="Actif">
62 <Port Name="P3R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif" />
63 <Port Name="P4R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif" />
64 <Port Name="P2P-XOR-CS-0" Type="ProvidedOptionalPort" Mode="Inactif" />
65 </Component>
66 </Composite>
67 <Composite Name="Compo2" Type="composit" InterfaceNB="0" Mode="Inactif">
68 <Port Name="P4R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif" />
69 <Port Name="P5R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif" />
70 <Port Name="P6R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif" />
71 <Port Name="P3P-XOR" Type="ProvidedOptionalPort" Mode="Inactif" />
72
73 <Connector Source="P4R-XOR-CS" Target="P5R-XOR-CS-0" Type="Delegate" />
74 <Connector Source="P5R-XOR-CS" Target="P6R-XOR-CS-0" Type="Delegate" />
75 <Connector Source="P6R-XOR-CS" Target="P7R-XOR-CS-0" Type="Delegate" />
76 <Connector Source="P3P-XOR" Target="P2P-XOR" Type="Delegate" />
77
78
79 <Component Name="*" Type="OptionalCompnent" InterfaceNB="0" Mode="Actif">
80 <Port Name="P6R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif" />
81 <Port Name="P7R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif" />
82 <Port Name="P5R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif" />
83 <Port Name="P3P-XOR-CS-0" Type="ProvidedOptionalPort" Mode="Inactif" />
84 </Component>
85 </Composite>
86
87
88
89 <Composite Name="resultat" Type="ORComposite" InterfaceNB="0" CardinalNB="0" Mode="Actif">
90 <Port Name="P4R-OR" Type="RequiredOptionalPort" Mode="Inactif" />
91 <Port Name="P1R-OR" Type="RequiredOptionalPort" Mode="Inactif" />
92 <Port Name="P2R-OR" Type="RequiredOptionalPort" Mode="Inactif" />
93 <Port Name="P3R-OR" Type="RequiredOptionalPort" Mode="Inactif" />
94 <Port Name="P5R-OR" Type="RequiredOptionalPort" Mode="Inactif" />
95 <Port Name="P6R-OR" Type="RequiredOptionalPort" Mode="Inactif" />
96 <Port Name="P2P-OR" Type="ProvidedOptionalPort" Mode="Inactif" />
97
98 <Connector Source="P1R-OR" Target="P1R-OR-M" Type="Delegate" />
99 <Connector Source="P2R-OR" Target="P2R-OR-M" Type="Delegate" />
100 <Connector Source="P3R-OR" Target="P1R-OR-0" Type="Delegate" />
101
102 <Component Name="graphical" Type="OptionalCompnent" InterfaceNB="0" Mode="Actif">
103 <Port Name="P1R-OR-0" Type="RequiredOptionalPort" Mode="Inactif" />
104 <Port Name="P2R-OR-0" Type="RequiredOptionalPort" Mode="Inactif" />
105 <Port Name="P2P-OR-M" Type="ProvidedOptionalPort" Mode="Inactif" />
106 </Component>
107 <Component Name="Audit" Type="OptionalCompnent" InterfaceNB="0" Mode="Inactif">
108 <Port Name="P3R-OR-0" Type="RequiredOptionalPort" Mode="Inactif" />
109 <Port Name="P4R-OR-0" Type="RequiredOptionalPort" Mode="Inactif" />
110 <Port Name="P3P-OR-M" Type="ProvidedOptionalPort" Mode="Inactif" />
111 </Component>
112 <Component Name="Textuel" Type="MandatoryComponent" InterfaceNB="0" Mode="Actif">
113 <Port Name="P1R-OR-M" Type="RequiredMandatoryPort" Mode="Actif" />
114 <Port Name="P2R-OR-M" Type="RequiredMandatoryPort" Mode="Actif" />
115 <Port Name="P1P-OR-M" Type="ProvidedMandatoryPort" Mode="Actif" />
116 </Component>
117
118 </Composite>
119
120 <Connector Source="P1P-CS" Target="P1R-XOR" Type="Assembly" />
121 <Connector Source="P1P-CS" Target="P4R-XOR" Type="Assembly" />
122 <Connector Source="P3P-CS" Target="P5R-XOR" Type="Assembly" />
123 <Connector Source="P3P-CS" Target="P3R-XOR" Type="Assembly" />
124 <Connector Source="P2P-CS" Target="P2R-XOR" Type="Assembly" />
125 <Connector Source="P2P-CS" Target="P6R-XOR" Type="Assembly" />
126 <Connector Source="P1P-XOR" Target="P5R-OR" Type="Assembly" />
127 <Connector Source="P1P-XOR" Target="P1R-OR" Type="Assembly" />
128 <Connector Source="P1P-XOR" Target="P3R-OR" Type="Assembly" />
129 <Connector Source="P2P-XOR" Target="P6R-OR" Type="Assembly" />
130 <Connector Source="P2P-XOR" Target="P2R-OR" Type="Assembly" />
131 <Connector Source="P2P-XOR" Target="P4R-OR" Type="Assembly" />
132
133 </System>
134
135 </Configuration>
136

```

Figure 4.16 : Le fichier XASADL du 'système du calcul simple'

7.1.3. La vérification programmée :

La Figure 4.17 présente les erreurs qui peuvent se produire dans la spécification du 'système du calcul simple' dont les messages d'erreurs sont écrits en rouge et les messages de validation qui signifient que la spécification d'un élément ou ses paramètres est bonne sont écrits en vert :

- La partie A de la Figure représente l'erreur "le nom du composant est dupliqué " ainsi que son message d'erreur.
- La partie B de la Figure représente l'erreur " le nombre d'instance InterfaceNB est nul ainsi son le message d'erreur.
- La partie C de la Figure représente l'erreur "ORComposite n'a pas du sous-composite" ainsi que son message d'erreur.
- La partie D de la figure représente l'erreur "le nombre de cardinalité CardinalNB du ORComposite est supérieur au nombre de ses sous-composites ainsi que son message d'erreur.
- La partie E de la Figure représente l'erreur "le mode du MandatoryComponent est inactif ainsi que son message d'erreur.
- La partie F de la Figure représente l'erreur "le mode du RequiredMandatoryPort est Inactif" ainsi que son message d'erreur.
- La partie G de la Figure représente l'erreur "le XORComposite à plus qu'un sous-composite son mode est actif " et son message d'erreur.

<pre> 58<Composite Name="Compo" Type="composit" InterfaceNB="0" Mode="Actif"> 59 <Port Name="P1R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif"/> 60 <Port Name="P2R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif"/> 61 <Port Name="P3R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif"/> 62 <Port Name="P1P-XOR-CS" Type="ProvidedOptionalPort" Mode="Inactif"/> 63 64 <Connector Source="P1R-XOR-CS" Target="P1R-XOR-CS-0" Type="Delegate" /> 65 <Connector Source="P2R-XOR-CS" Target="P2R-XOR-CS-0" Type="Delegate" /> 66 <Connector Source="P3R-XOR-CS" Target="P4R-XOR-CS-0" Type="Delegate" /> 67 <Connector Source="P1P-XOR-CS" Target="P1P-XOR" Type="Delegate" /> 68 <Connector Source="P1P-XOR-CS-0" Target="P3R-XOR-CS-0" Type="Assembly" /> 69 70<Component Name="+" Type="OptionalCompent" InterfaceNB="0" Mode="Inactif"> 71 <Port Name="P2R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif"/> 72 <Port Name="P1R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif"/> 73 <Port Name="P1P-XOR-CS-0" Type="ProvidedOptionalPort" Mode="Inactif"/> 74 </Component> 75 76<Component Name="*" Type="OptionalCompent" InterfaceNB="0" Mode="Actif"> 77 <Port Name="P3R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif"/> 78 <Port Name="P4R-XOR-CS-0" Type="RequiredOptionalPort" Mode="Inactif"/> 79 <Port Name="P2P-XOR-CS-0" Type="ProvidedOptionalPort" Mode="Inactif"/> 80 </Component> 81 82 </Composite> 83 84<Composite Name="Compo" Type="composit" InterfaceNB="0" Mode="Inactif"> 85 <Port Name="P4R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif"/> 86 <Port Name="P5R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif"/> 87 <Port Name="P6R-XOR-CS" Type="RequiredOptionalPort" Mode="Inactif"/> 88 <Port Name="P3P-XOR" Type="ProvidedOptionalPort" Mode="Inactif"/> 89 90 <Connector Source="P4R-XOR-CS" Target="P5R-XOR-CS-0" Type="Delegate" /> 91 <Connector Source="P5R-XOR-CS" Target="P6R-XOR-CS-0" Type="Delegate" /> 92 <Connector Source="P6R-XOR-CS" Target="P7R-XOR-CS-0" Type="Delegate" /> 93 <Connector Source="P3P-XOR" Target="P2P-XOR" Type="Delegate" /> 94 </pre> <p>OK : The name Compois not duplicate Error : The name Compo is duplicate OK : The name +is not duplicate OK : The name *is not duplicate</p>	<pre> 102<Composite Name="resultat" Type="ORComposite" InterfaceNB="0" CardinalNB="7" Mode="Actif"> 103 <Port Name="P4R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 104 <Port Name="P1R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 105 <Port Name="P2R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 106 <Port Name="P3R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 107 <Port Name="P5R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 108 <Port Name="P6R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 109 <Port Name="P1P-OR" Type="ProvidedOptionalPort" Mode="Inactif"/> 110 <Port Name="P2P-OR" Type="ProvidedOptionalPort" Mode="Inactif"/> 111 <Port Name="P3P-OR" Type="ProvidedOptionalPort" Mode="Inactif"/> </pre> <p>Error : The Cardinal number doesn't respect the rule of cardinals numbers</p>
<pre> <Component Name="Textuel" Type="MandatoryComponent" InterfaceNB="0" Mode="Actif"> </pre> <p>Name : Textuel Type : MandatoryComponent InterfaceNB : 0 Mode : Actif</p> <p>OK : The name Textuel is not duplicate Error : The Interface Number 0 is unacceptable</p>	<pre> 21<Component Name="A" Type="MandatoryComponent" InterfaceNB="0" Mode="Inactif"> 22 <Port Name="P4R-CS-M" Type="RequiredMandatoryPort" Mode="Inactif" /> 23 <Port Name="P4P-CS-M" Type="ProvidedMandatoryPort" Mode="Inactif" /> 24 </Component> </pre> <p>Name : A Type : MandatoryComponent InterfaceNB : 0 Mode : Inactif</p> <p>Error : The Mode of A is unacceptable</p>
<pre> 102<Composite Name="resultat" Type="ORComposite" InterfaceNB="0" CardinalNB="0" Mode="Actif"> 103 <Port Name="P4R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 104 <Port Name="P1R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 105 <Port Name="P2R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 106 <Port Name="P3R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 107 <Port Name="P5R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 108 <Port Name="P6R-OR" Type="RequiredOptionalPort" Mode="Inactif"/> 109 <Port Name="P1P-OR" Type="ProvidedOptionalPort" Mode="Inactif"/> 110 <Port Name="P2P-OR" Type="ProvidedOptionalPort" Mode="Inactif"/> 111 <Port Name="P3P-OR" Type="ProvidedOptionalPort" Mode="Inactif"/> 112 113 <Connector Source="P1R-OR" Target="P1R-OR-M" Type="Delegate" /> 114 <Connector Source="P2R-OR" Target="P2R-OR-M" Type="Delegate" /> 115 <Connector Source="P3R-OR" Target="P1R-OR-0" Type="Delegate" /> 116 117 118 119 </Composite> </pre> <p>Error : ORComposite should have at least one Component , Component =</p>	<pre> 134<Component Name="Textuel" Type="MandatoryComponent" InterfaceNB="0" Mode="Actif"> 135 <Port Name="P1R-OR-M" Type="RequiredMandatoryPort" Mode="Inactif"/> 136 <Port Name="P2R-OR-M" Type="RequiredMandatoryPort" Mode="Actif"/> 137 <Port Name="P1P-OR-M" Type="ProvidedMandatoryPort" Mode="Actif"/> 138 </Component> 139 </pre> <p>Port Name : P1R-OR-M Type : RequiredMandatoryPort Mode : Inactif</p> <p>Error : The Mode of P1R-OR-M PORT is unacceptable</p>
<pre> 36<Composite Name="Operation" Type="XORCompsite" InterfaceNB="0" Mode="Actif"> 37 <Port Name="P1R-XOR" Type="RequiredOptionalPort" Mode="Inactif"/> 38 <Port Name="P2R-XOR" Type="RequiredOptionalPort" Mode="Inactif"/> 39 <Port Name="P3R-XOR" Type="RequiredOptionalPort" Mode="Inactif"/> 40 <Port Name="P4R-XOR" Type="RequiredOptionalPort" Mode="Inactif"/> 41 <Port Name="P5R-XOR" Type="RequiredOptionalPort" Mode="Inactif"/> 42 <Port Name="P6R-XOR" Type="RequiredOptionalPort" Mode="Inactif"/> 43 <Port Name="P1P-XOR" Type="ProvidedOptionalPort" Mode="Inactif"/> 44 <Port Name="P2P-XOR" Type="ProvidedOptionalPort" Mode="Inactif"/> 45 46 <Connector Source="P1R-XOR" Target="P1R-XOR-CS" Type="Delegate" /> 47 <Connector Source="P2R-XOR" Target="P2R-XOR-CS" Type="Delegate" /> 48 <Connector Source="P3R-XOR" Target="P3R-XOR-CS" Type="Delegate" /> 49 <Connector Source="P4R-XOR" Target="P4R-XOR-CS" Type="Delegate" /> 50 <Connector Source="P5R-XOR" Target="P5R-XOR-CS" Type="Delegate" /> 51 <Connector Source="P6R-XOR" Target="P6R-XOR-CS" Type="Delegate" /> 52 53<Composite Name="Compo1" Type="composit" InterfaceNB="0" Mode="Actif"> 77 </Composite> 78 79<Composite Name="Compo2" Type="composit" InterfaceNB="0" Mode="Actif"> 98 </Composite> 99 90 </Composite> </pre> <p>Error : The XOR Composite Operation has more than one compo Actif</p>	

Figure 4.17 : Représentation des erreurs de spécification du 'système de calcul simple'

7.2. Cours en ligne

C'est un système qui affiche les cours en ligne, et les rendre visionnés par l'intéressé, de fait il permet d'afficher les cours visés en trois formats : PDF, Lecteur Vidéo et PowerPoint.

En revanche, le type de format change automatiquement tout dépend la demande et le débit de l'internet. Par exemple quand les demandes des intéressés dépassent 1000 demandes, l'affichage se réduit automatiquement au format PDF, la même chose quand le débit de l'internet est moins de 1MB/s. Sinon dans les cas stables (bon débit, moins de demandes) l'affichage sera en vidéo et en Power point.

7.2.1. Spécification du système

Dans la Figure 4.18 nous proposons une spécification de ce système en utilisant notre modèle de spécification.

- ✓ Nous avons proposé que le système soit présenté par un composite et composé de 3 composants :
 - Le 1^{er} composant nommé 'Cours-PDF' pour l'affichage des cours en format PDF.
 - Le 2^{ème} composant nommé 'Cours-Vidéo' pour l'affichage des cours en format Vidéo.
 - Le 3^{ème} composant nommé 'Cours-PowerPoint' pour l'affichage des cours en format PowerPoint.

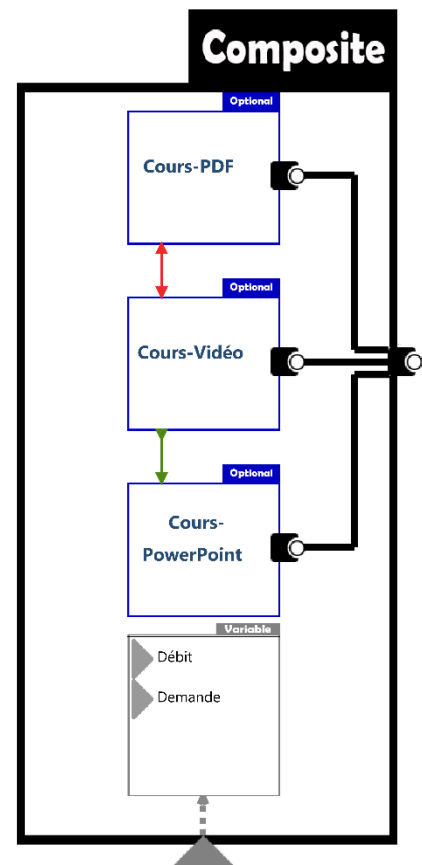
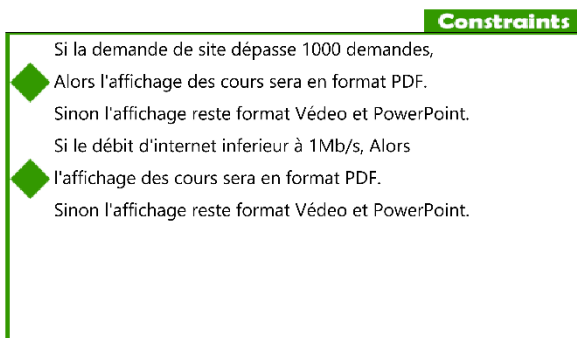


Figure 4.18 : Spécification du Système 'Cours en ligne'

- ✓ Nous avons choisi le type de ces composants 'OptionalComponent' car le changement de leur mode passe d'actif à inactif et vice versa.
- ✓ Nous avons mis 'ExclutConnector' entre le composant 'Cours-PDF' et le composant 'Cours-Vidéo', pour spécifier que l'activation de l'un de ces deux composants rend le 2eme inactif.
- ✓ Nous avons mis 'RequiredConnector' entre le composant 'Cours-Vidéo' et le composant 'Cours-PowerPoint', pour spécifier que lorsque le composant 'Cours-Vidéo' est actif le composant 'Cours-PowerPoint' sera obligatoirement actif au même temps.

L'activation et la désactivation de ces composants dépend au variables débit d'internet et demande du site, pour cela :

- ✓ Nous avons ajouté le port 'QualityAttribute' et 'Variables' pour spécifier ces variables.

- ✓ Nous avons ajouté 'Constraints' pour décrire les contraintes à respecter pour réaliser l'adaptation que nous souhaitons.

7.2.2. Le fichier XASADL obtenu

```

1 <?xml version="1.0" encoding="UTF-8"?>
2<Configuration>
3<System Name="Plateforme" Type="Composite" >
4
5<Composite Name="Affichage" Type="composit" InterfaceNB="0" Mode="Actif">
6 <Port Name="QuelityAttribut" Type="QualityAttribut" Mode="Inactif" />
7 <Port Name="ProvidedOptionalPort" Type="ProvidedOptionalPort" Mode="Inactif" />
8
9
10<Component Name="Cours-PDF " Type="OptionalCompnent" InterfaceNB="0" Mode="Inactif">
11 <Port Name="ProvidedOptionalPort" Type="ProvidedOptionalPort" Mode="Inactif" />
12 </Component>
13<Component Name="Cours-Vidéo" Type="OptionalCompnent" InterfaceNB="0" Mode="Actif">
14 <Port Name="ProvidedOptionalPort" Type="ProvidedOptionalPort" Mode="Inactif" />
15 </Component>
16<Component Name="Cours-PowerPoint" Type="OptionalCompnent" InterfaceNB="0" Mode="Actif">
17 <Port Name="ProvidedOptionalPort" Type="ProvidedOptionalPort" Mode="Inactif" />
18 </Component>
19
20 <ControlConnector Source="Cours-PDF " Target="Cours-Vidéo" Type="Exclut" />
21 <ControlConnector Source="Cours-Vidéo" Target="Cours-PowerPoint" Type="Required" />
22
23<Variables Type="Variables">
24 <Variable Name="Débit" Type="Variable" />
25 <Variable Name="Demande" Type="Variable" />
26 </Variables>
27
28 </Composite>
29
30<Constraints Name="" Type="Constraints">
31 <Constraint Name="Si la demande de site dépasse 1000 demandes, Alors l'affichage des cours sera en format PDF.
32 Sinon l'affichage reste format Vidéo et PowerPoint." Type="Constraint"/>
33 <Constraint Name="Si le débit d'internet inferieur à 1Mb/s, Alors l'affichage des cours sera en format PDF.
34 Sinon l'affichage reste format Vidéo et PowerPoint." Type="Constraint"/>
35 </Constraints>
36
37 </System>
38 </Configuration>
39

```

Figure 4.19 : Le fichier XASADL généré de Système 'Cours en ligne'

La Figure 4.19 présente le fichier généré de modèle de spécification du système 'Cours en ligne', ce fichier sera évalué dans la section suivante pour valider la spécification du système.

7.2.3. Vérification de la spécification du système

La Figure 4.20 présente les erreurs de spécification du système 'Cours en ligne' par notre modèle, la partie 1 de la figure présente le fichier XASADL généré avec des erreurs de spécification. D'autre part, la partie 2 de la figure désigne les messages d'erreurs et les messages de validation résultants de la vérification programmée.

Dans cette figure, on distingue deux types des erreurs :

- La 1^{ère} erreur est "le nombre d'instance InterfaceNB est nul", elle est illustrée dans la figure dont le composite nommé Affichage son InterfaceNB est égal à 0 (encadrement rose), la même chose pour les composants 'Cours-PDF' (encadrement bleu), 'Cours-Vidéo' (encadrement vert) et le composant nommé aussi 'Cours-PDF' (jaune).
- La 2^{ème} erreur est "le nom d'un composant est dupliqué". Cette erreur est illustrée par un encadrement rouge dont il y a deux composant du même nom 'Cours-PDF'.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configuration>
3
4 <System Name="Plateforme" Type="Composite" >
5
6 <Composite Name="Affichage" Type="composit" InterfaceNB="0" Mode="Actif">
7 <Port Name="QualityAttribut" Type="QualityAttribut" />
8
9 <Component Name="Cours-PDF" Type="OptionalCompnent" InterfaceNB="0" Mode="Inactif">
10 </Component>
11 <Component Name="Cours-Vidéo" Type="OptionalCompnent" InterfaceNB="0" Mode="Actif">
12 </Component>
13 <Component Name="Cours-PDF" Type="OptionalCompnent" InterfaceNB="0" Mode="Actif">
14 </Component>
15
16 <ControlConnector Source="Cours-PDF" Target="Cours-Vidéo" Type="Exclut" />
17 <ControlConnector Source="Cours-Vidéo" Target="Cours-PowerPoint" Type="Required" />
18
19 <Variables Type="Var">
20 <Variable Name="Débit" Type="var"/>
21 <Variable Name="Demande" Type="var"/>
22 </Variables>
23
24 </Composite>
25
26 <Constraints Type="composit" >
27 <Constraint Name="Si la demande de site dépasse 1000 demandes, Alors l'affichage des cours sera en format PDF.
28 Sinon l'affichage reste format Vidéo et PowerPoint." Type="composit"/>
29 <Constraint Name="Si le débit d'internet inferieur à 1Mb/s, Alors l'affichage des cours sera en format PDF.
30 Sinon l'affichage reste format Vidéo et PowerPoint." Type="composit"/>
31 </Constraints>
32
33 </System>
34
35 </Configuration>
36

```

1

```

Composite : [[Element: <Port/>], [Element: <Component/>], [Element: <Component/>], [Element:
Name : Affichage
Type : composit
InterfaceNB : 0
Mode : Actif
[
Component
Error : The Interface Number 0 is unacceptable
Component
Error : The Interface Number 0 is unacceptable
Component
Error : The Interface Number 0 is unacceptable
]
Error : The Interface Number 0 is unacceptable
Names List: [Cours-PDF, Cours-Vidéo, Cours-PDF, Affichage]
OK : The name Cours-PDF is not duplicate
Error : The name Cours-PDF is duplicate
OK : The name Cours-Vidéo is not duplicate
OK : The name Cours-PDF is not duplicate
OK : The name Affichage is not duplicate

```

2

Figure 4.20 : Représentation des erreurs de spécification du système 'Cours en ligne'

8. Temps d'exécution

Nous avons fait un test sur le temps d'exécution de code de vérification sur plusieurs composites, le Tableau 4.2 présente ce test.

Le test a été effectués sur un ordinateur portable HP sous Windows 10 professionnel, avec un processeur Intel(R) Core (TM) i7 CPU U650 @2.59GHz et une RAM de 8.00 Go.

Nombre	Temps
0	0
10	407
20	540
40	672
80	1045
160	2372
320	6892

Table 4.2 : Temps d'exécution pour Composites.

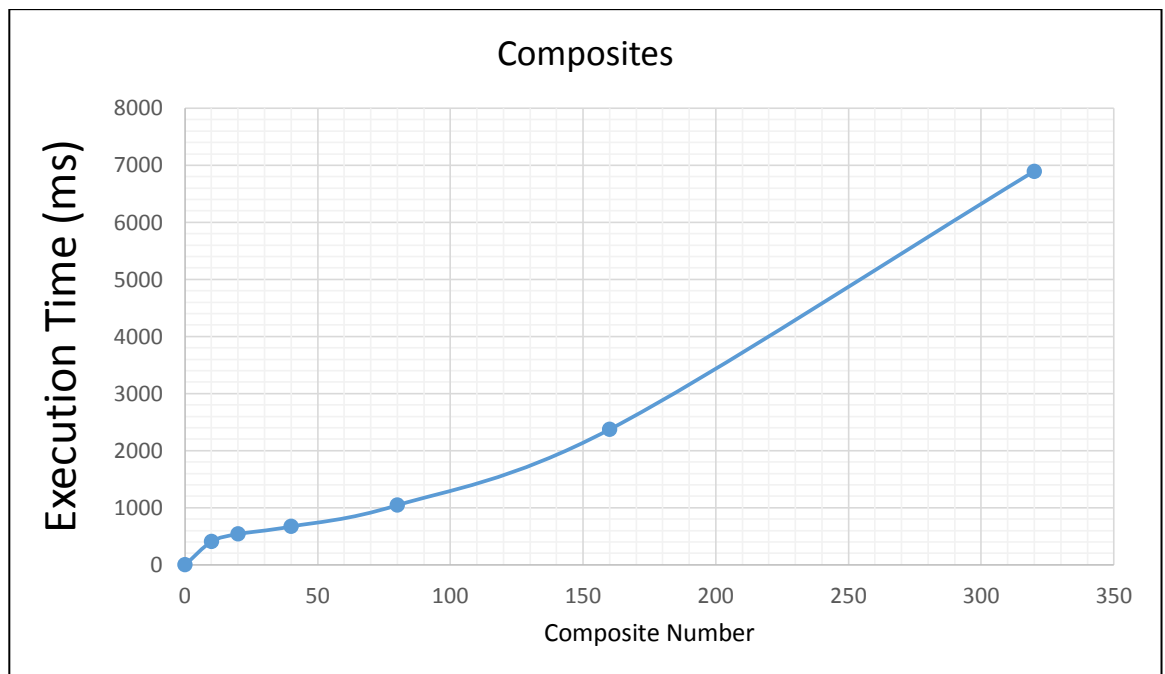


Figure 4.21 : Test de temps d'exécution

La courbe montre le test effectué sur un nombre croissant de composites (Figure 4.21). Le résultat de test montre que le temps d'exécution est une fonction de la forme

exponentielle, cependant pour une architecture logicielle de 320 composites qui contient 832 de connecteurs et avec un nombre moyen de 3 connecteurs pour chaque composite, le temps d'exécution qui est approximativement de 6 secondes reste très acceptable.

En conséquence, on peut dire que notre algorithme de vérification structurelle de l'architecture est réalisable et faisable pour des systèmes comptant un grand nombre de composants.

9. Conclusion

Ce chapitre constitue le dernier chapitre de ce mémoire, il a été dédié à la représentation des fonctionnalités de notre éditeur graphique qui aide à la spécification des systèmes auto-adaptatifs et le code de la vérification du modèle obtenu.

Nous avons présenté l'environnement et outils de travail utilisés, les étapes de l'implémentation de notre approche et nous avons clôturé ce chapitre par des exemples illustratifs.

CONCLUSION

Dans ce mémoire, nous avons proposé une nouvelle approche de spécification des systèmes auto-adaptatifs, l'approche proposée est générique elle est basée sur l'approche orienté composants en intégrant des concepts de variabilité trouvés dans les SPLs.

Le résultat obtenu est un éditeur graphique permettant de spécifier des systèmes auto-adaptatifs

Pour se faire, nous avons tout d'abord entamé une étude sur le domaine des systèmes auto-adaptatifs pour connaître les concepts et les bases fondamentaux de ce domaine.

Cette étude nous a aidé à extraire les informations nécessaires sur ces systèmes ainsi que les problèmes liés à leur spécification.

Ensuite, Nous avons étudié les modèles de spécifications de la variabilité dans les SPLs et plus précisément 'feature model', que nous avons intégrés dans notre métamodèle, parmi les notions utilisées : 'OR', 'XOR', 'Exclut', 'Requiert', ... etc.

Après l'acquisition des informations nécessaires sur les concepts de feature model nous avons pu spécifier notre métamodèle qui aide à la spécification des systèmes auto-adaptatifs en couvrant les deux types de variabilité, la variabilité au niveau de l'architecture et la variabilité au niveau de la configuration. Pour se faire, nous avons enrichi notre métamodèle générique basé sur les composants avec les notions de variabilités trouvés dans les SPL.

Une fois que notre métamodèle a été définis, nous avons développé un éditeur graphique simple, facile à utiliser pour spécifier les systèmes auto-adaptatifs.

La deuxième étape été de généré un code textuel que nous avons appelé XASADL. Ce code est obtenu grâce à une transformation de modèle, pour se faire, nous avons écrit un code AQL qui a permis de transformer la représentation graphique en un code textuel.

La troisième étape été d'identifier quelques règles structurelles pour réaliser une vérification du modèle obtenu après sa spécification à travers notre IDE. Nous avons donc implémenté un algorithme pour effectuer cette vérification sur le code XASADL obtenu.

Enfin, afin de valider notre approche, nous avons illustré les concepts développés à travers deux exemples.

Cette étape n'était pas facile pour nous car c'était la première fois que nous utilisons la plupart des outils, en commençant par Sirius, ainsi que pour faire la vérification nous étions obligés de passer par la transformation du modèle au fichier XML à travers l'outil Acceleo. En effet, il y a un grand manque de documentations sur les deux outils, ce qui nous a poussés à apprendre à travailler et manipuler en guise de notre expérience en l'utilisant.

Ce projet a fait l'objet d'une expérience intéressante, qui nous a permis d'améliorer nos connaissances et nos compétences dans plusieurs domaines à savoir le domaine de la modélisation et méta-modélisation dans le domaine de développement des éditeurs graphiques et le domaine de transformation de modèle.

La partie d'implémentation de l'éditeur graphique n'a pas été facile, la prise en main des outils tel que Sirius et GMF est très difficile à cause de la complexité de la tâche et un grand manque de documentation. C'est pour cette raison on ne trouve pas beaucoup d'outils proposés dans la littérature pour soutenir les Frameworks de spécification de systèmes.

Cependant, au cours de notre développement, nous avons pensé à plusieurs perspectives applicables, nous citons : la vérification sémantique du modèle que nous n'avons pas abordé dans ce travail, appliquer aussi notre approche sur des cas d'études réels plus complexes, côté implémentation l'amélioration du fichier Odesign de telle façon qu'il va faire plusieurs autres vérifications sur le modèle.

BIBLIOGRAPHIE

- [1] R. De Lemos *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*: Springer, 2013, pp. 1-32.
- [2] M. Salehie, L. J. A. t. o. a. Tahvildari, and a. systems, "Self-adaptive software: Landscape and research challenges," vol. 4, no. 2, pp. 1-42, 2009.
- [3] D. Weyns, M. U. Iftikhar, S. Malek, and J. Andersson, "Claims and supporting evidence for self-adaptive systems: a literature study," presented at the Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Zurich, Switzerland, 2012.
- [4] F. Chauvel, "Méthodes et outils pour la conception de systèmes logiciels auto-adaptatifs," Université de Bretagne Sud, 2008.
- [5] A. J. I. W. P. Computing, "An architectural blueprint for autonomic computing," vol. 31, no. 2006, pp. 1-6, 2006.
- [6] J. O. Kephart and D. M. J. C. Chess, "The vision of autonomic computing," vol. 36, no. 1, pp. 41-50, 2003.
- [7] D. Weyns, "Software engineering of self-adaptive systems," in *Handbook of Software Engineering*: Springer, 2019, pp. 399-443.
- [8] B. S. Biswal and A. Mohapatra, "Analogy of autonomic computing: An AIS (artificial immune systems) overview," in *2014 IEEE International Conference on Computational Intelligence and Computing Research*, 2014, pp. 1-4: IEEE.
- [9] M. Trapp and B. Schürmann, "On the modeling of adaptive systems," in *International Workshop on Dependable Embedded Systems*, 2003: Citeseer.
- [10] L. Sabatucci, V. Seidita, and M. Cossentino, "The four types of self-adaptive systems: a metamodel," in *International Conference on Intelligent Interactive Multimedia Systems and Services*, 2018, pp. 440-450: Springer.
- [11] R. El Ballouli, "Modeling self-configuration in Architecture-based self-adaptive systems," Université Grenoble Alpes, 2019.
- [12] N. Abbas, J. Andersson, and W. Löwe, "Autonomic software product lines (ASPL)," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, 2010, pp. 324-331.
- [13] S.-W. Cheng, D. Garlan, and B. Schmerl, "Evaluating the effectiveness of the rainbow self-adaptive system," in *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2009, pp. 132-141: IEEE.
- [14] N. Fredj, Y. H. Kacem, and M. J. T. J. o. S. Abid, "An event-based approach for formally verifying runtime adaptive real-time systems," vol. 77, no. 3, pp. 3110-3143, 2021.
- [15] N. Khakpour, S. Jalili, C. Talcott, M. Sirjani, and M. J. S. o. C. P. Mousavi, "Formal modeling of evolving self-adaptive systems," vol. 78, no. 1, pp. 3-26, 2012.
- [16] M. L. Berkane, M. Boufaïda, and L. J. L. N. o. S. E. Seinturier, "A modular approach dedicated to self-adaptive system," vol. 3, no. 3, p. 183, 2015.
- [17] B. Morin *et al.*, "An aspect-oriented and model-driven approach for managing dynamic variability," in *international conference on Model Driven Engineering Languages and Systems*, 2008, pp. 782-796: Springer.
- [18] A. Elkhodary, N. Esfahani, and S. Malek, "FUSION: a framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 7-16.
- [19] P. Clements and L. Northrop, "Software product lines - practices and patterns," in *SEI series in software engineering*, 2002.

- [20] S. OUALI, N. KRAIEM, H. B. J. G. J. o. C. S. GHEZALA, and Technology, "Security requirement engineering for Intentional Software Product Line," 2012.
- [21] J. van Gorp, *Variability in software systems: the key to software reuse*. Department of Software Engineering & Computer Science, Blekinge Institute of ..., 2000.
- [22] F. Bachmann and P. C. Clements, "Variability in software product lines," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST2005.
- [23] G. Halmans and K. J. I. F. U. E. Pohl, "Communicating the variability of a software-product family to customers," vol. 18, no. 3-4, pp. 113-131, 2004.
- [24] K. Pohl, G. Böckle, and F. Van Der Linden, *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.
- [25] E. Bagheri, T. Di Noia, A. Ragone, and D. Gasevic, "Configuring software product line feature models based on stakeholders' soft and hard requirements," in *International Conference on Software Product Lines*, 2010, pp. 16-31: Springer.
- [26] K. Czarnecki, S. Helsen, U. J. S. p. I. Eisenecker, and practice, "Formalizing cardinality-based feature models and their specialization," vol. 10, no. 1, pp. 7-29, 2005.
- [27] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.
- [28] Ø. Haugen, A. Wąsowski, and K. Czarnecki, "CVL: common variability language," in *Proceedings of the 16th International Software Product Line Conference-Volume 2*, 2012, pp. 266-267.
- [29] S. Consortium, "Synthesis guidebook," Technical report, SPC-91122-MC. Herndon, Virginia: Software Productivity ...1991.
- [30] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst1990.
- [31] M. L. Griss, J. Favaro, and M. d'Alessandro, "Integrating feature modeling with the RSEB," in *Proceedings. Fifth International Conference on Software Reuse (Cat. No. 98TB100203)*, 1998, pp. 76-85: IEEE.
- [32] R. J. J. G. L. Mazo, "Avantages et limites des modèles de caractéristiques dans la modélisation des exigences de variabilité," no. 111, pp. 42-48, 2014.
- [33] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow, "Extending feature diagrams with UML multiplicities," in *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, 2002, vol. 23, pp. 1-7.
- [34] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration using feature models," in *International conference on software product lines*, 2004, pp. 266-283: Springer.
- [35] K. Czarnecki and C. H. P. Kim, "Cardinality-based feature modeling and constraints: A progress report," in *International Workshop on Software Factories*, 2005, pp. 16-20: ACM San Diego, California, USA.
- [36] F. Barbier, C. Cauvet, M. Oussalah, D. Rieu, and S. J. L. O. Bennisri, "Concepts clés et techniques de réutilisation dans l'ingénierie des systèmes d'information," vol. 10, no. 1, pp. 11-35, 2004.
- [37] L. Yessad, "Sélection sémantique de composants logiciels basée sur des critères de qualité de service."
- [38] N. Sadou-Harireche, "Evolution Structurale dans les Architecture Logicielles à base de Composants," PhD thesis, Université de Nantes, 2007.
- [39] D. Garlan, R. Monroe, and D. Wile, "Acme: An architecture description interchange language," in *CASCON First Decade High Impact Papers*, 2010, pp. 159-173.
- [40] M. NACERA, "UNE APPROCHE HYBRIDE POUR TRANSFORMER LES MODELES."
- [41] K. Bentlemcen, "Une Approche architecture logicielle pour la composition du service web," Blida, 2010.