TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

MTF	Mesure de la Taille Fonctionnelle
COSMIC	Common Software Measurement International Consortium
FUR	Functional User Requirements
UML	Unified Modeling Language
IFPUG	International Function Point Users Group
LOC	Length Of Code
UCP	Use Case Point
FPA	Function Point Analysis
ISO	International Organization for Standardization
FISMA	Finish Software Measurement Association
CFP	Cosmic Function Point
OMG	Object Management Group
MDA	Model Driven Architecture
MOF	Meta Object Facility
CWM	Common Warehouse Meta-model
XMI	XML Metadata Interchange
CIM	Computation Independent Model
PIM	Platform Independent Model
PSM	Platform Specific Model
OCL	Object Constraint Language
CCM	CORBA Component Model

INTRODUCTION

Au 18ème siècle, le célèbre physicien Lord Kelvin a declaré: "*When you can measure what you are speaking about, and can express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, then your knowledge is of a meager and unsatisfactory kind*" (Kelvin, 1891).

Depuis des siècles, la nature humaine a connu différents standards (étalons) ou unités de mesure qui permettent de comprendre des choses réelles en leur donnant une valeur significative. Parmi ces étalons de mesure, on cite le mètre pour mesurer les distances et la seconde pour mesurer le temps.

Comme pour tout autre domaine scientifique, les mesures devraient être nécessaires pour le domaine de génie logiciel. Cependant, elles sont encore peu utilisées et déployées dans cette discipline malgré la croissance de ce domaine et le rôle qu'elles jouent dans les autres disciplines scientifiques.

Un projet logiciel doit être de bonne qualité pour être bien réussi. Par conséquent, un ingénieur logiciel doit disposer de métriques pour mesurer ce critère. Ainsi, on peut avoir un contrôle plus précis, prévisible et reproductible sur le processus de développement de logiciels.

Le logiciel étant un ensemble de programmes et de données, la mesure de sa taille n'est pas une chose évidente. En effet, elle sert principalement à contrôler la qualité du produit et à mieux gérer les projets de développement afin de réduire le coût de production. De plus, les mesures constituent des paramètres en entrée pour estimer l'effort de développement d'un projet ainsi que pour comparer la productivité des projets informatiques.

Plusieurs méthodes de mesure de la taille d'un logiciel ont été déployées, telles que les mesures de longueur (en nombre de lignes de code, nombre de pages dans un cahier de

charges) et les mesures de fonctionnalités (nombre de points de fonction dans une spécification).

Les mesures de la taille fonctionnelle (MTF) sont dérivées des spécifications et peuvent être obtenues dès le début du cycle de développement du logiciel, ce qui les favorise par rapport aux mesures de longueur. En effet, les mesures basées sur le nombre des lignes de code ne peuvent être obtenues qu'après la phase de développement.

Dans ce contexte, le groupe COSMIC (Common Software Measurement International Consortium) regroupant un ensemble d'experts dans le domaine des mesures du logiciel, a présenté la nouvelle génération de méthode de mesure de la taille fonctionnelle d'un logiciel nommée COSMIC – ISO 19761. Cette méthode est basée sur l'application d'un ensemble de principes et de règles pour un logiciel tel qu'il est perçu par ses utilisateurs c'est-à-dire à travers les fonctionnalités utilisateurs requises (FUR).

La méthode COSMIC est théoriquement bien définie. De plus elle est appuyée par des cas d'utilisations qui servent comme référentiels pour la mesure de la taille fonctionnelle. Cependant, pour être appliquée en industrie, cela nécessite l'intervention d'experts dans le domaine de mesure. Le mesureur prend du temps pour lire les spécifications afin d'obtenir l'information qu'il lui faut pour établir la mesure. Éventuellement, cette tâche lui demande de l'effort manuel.

De point de vue modélisation, UML est la notation standard en industrie pour la spécification et la modélisation des systèmes logiciels à différents aspects partant de la spécification des exigences au déploiement. C'est pour cette raison que plusieurs initiatives de recherche ont porté sur la mesure d'un logiciel partant de ses modèles UML (Bévo *et al.*, 1999 ; Jenner, 2001 ; Azzouz *et al.*, 2004 ; vilus *et al.*, 2004 ; Habela *et al.*, 2004 ; Levesque *et al.*, 2008 ; Sellami *et al.*, 2009 ; Van Den Burg *et al.*,2005 et Lavazza *et al.*,2009).
Cette idée a notamment été introduite dans notre article intitulé « *COSMIC Functional Size Measurement Using UML Models* » (Barkallah *et al.,* 2011). L'objectif de l'article est de

présenter une revue de la littérature et une analyse des travaux de recherche antérieurs portant sur l'application de la mesure de la taille fonctionnelle avec COSMIC à partir des modèles UML. De plus, nous avons introduit notre motivation à proposer une nouvelle approche qui consiste à étendre UML pour la mesure avec COSMIC à l'aide d'un profil.

Étendre UML pour supporter de manière adéquate et spécifique la méthode COSMIC est l'objectif essentiel de ce projet de recherche. Cette nouvelle extension d'UML consiste à définir un profil UML pour COSMIC.

Avant de concevoir le profil, il est primordial de capturer les concepts pertinents de la méthode COSMIC et leurs relations dans un modèle de domaine qui servira comme méta-modèle de notre profil. Dans ce méta-modèle, on représentera les principaux concepts de COSMIC et leurs différentes relations. En outre, il est indispensable de respecter les règles du domaine, ce qui nécessite l'identification des différentes contraintes sur ce méta-modèle et les décrire formellement (Barkallah *et al.,* 2011). Le méta-modèle sera la base de nouveaux stéréotypes, valeurs marquées et différentes contraintes exprimées souvent avec le langage OCL. Ce profil UML sera utilisé pour annoter des modèles UML afin de capturer l'information pertinente pour la mesure de taille fonctionnelle avec COSMIC.

Ultimement, ce profil UML permettra d'automatiser partiellement la procédure de mesure de la taille fonctionnelle avec COSMIC. Cette procédure est souvent manuelle; par conséquent, l'expert en mesure prend beaucoup de temps à analyser les spécifications des exigences pour extraire les informations pertinentes et nécessaires pour établir sa mesure. Notamment, il serait intéressant de concevoir des modèles orientés mesure.

Avec le profil UML pour COSMIC, il sera plus facile à un architecte de modéliser ces informations en utilisant des diagrammes UML déjà existants et des nouveaux stéréotypes spécifiques à la méthode COSMIC (qu'on ne peut pas capturer à partir d'autres diagrammes). Ce travail facilitera la tâche du mesureur qui pourra établir sa mesure à l'aide de l'information capturée à partir du profil.

Le reste du rapport est composé de cinq chapitres organisés de la façon suivante :

Le chapitre 2 présente l'état de l'art des mesures de la taille fonctionnelle. Nous présentons d'abord la notion de mesure en génie logiciel. Ensuite nous décrivons d'une façon plus détaillée les mesures fonctionnelles. Nous présentons à la fin du chapitre le processus de mesure COSMIC.

Le chapitre 3 traite l'approche dirigée par les modèles. Nous donnons en premier un aperçu sur l'approche MDA et le langage UML. Ensuite nous traitons de plus près la notion de méta-modélisation et les profils UML.

Le chapitre 4 présente une revue de littérature des approches traitant l'idée de mesurer la taille fonctionnelle avec COSMIC partant des fonctionnalités décrites à l'aide des diagrammes UML. Nous commençons par décrire ces différents travaux. Ensuite, nous procédons à une analyse des travaux à l'aide d'une comparaison. Nous concluons le chapitre par une présentation d'autres travaux reliés.

Le chapitre 5 expose la conception du profil UML proposé pour supporter la méthode COSMIC. Nous commençons par dévoiler le méta-modèle du domaine et les règles qui lui sont liées. Ensuite, nous présentons l'architecture du profil, le mapping du méta-modèle vers UML incluant la définition des stéréotypes et nous terminons par présenter l'implémentation du profil avec un outil de modélisation.

Le chapitre 6 présente une étude de cas qui permet de mettre en valeur l'application du profil UML proposé.

CHAPITRE 1

LA MESURE DE LA TAILLE FONCTIONNELLE

1.1 Introduction

La mesure de la taille d'un logiciel n'est pas une chose évidente. Toutefois elle joue un rôle important dans la discipline du génie logiciel. Plusieurs méthodes de mesure ont été proposées successivement à l'industrie, parmi lesquelles, les mesures de la taille fonctionnelle ont connu un intérêt assez important. Dans ce chapitre, nous exposons une revue de littérature sur les mesures en génie logiciel en particulier les mesures de la taille fonctionnelle. Nous présentons d'abord la place de la mesure en génie logiciel. Par la suite, nous discutons l'évolution des différentes générations de méthodes de mesure. Nous nous intéressons plus spécifiquement aux méthodes IFPUG et COSMIC.

1.2 La mesure en génie logiciel

La mesure est un moyen indispensable pour toutes les disciplines scientifiques, par exemple, pour la médecine, l'industrie, etc. En effet, appliquer une mesure, nous permet d'évaluer la qualité ou faire des estimations afin d'établir des comparaisons. Ci-dessous, quelques exemples d'application de la mesure :

- Un médecin mesure la tension d'un patient afin de cerner son état de santé.
- Une entreprise mesure la qualité de son produit pour faire des négociations avec les acheteurs ou pour faire des améliorations.
- On mesure la quantité de farine pour faire un gâteau.
- On mesure la surface d'un terrain pour construire une maison.
- Etc.

Actuellement, le système international de mesure comporte sept unités de base :[1]

- Longueur (Mètre)
- Masse (Kilogramme)
- Temps (Seconde)
- Courant électrique (Ampère)
- Température (Kelvin)
- Quantité de matière (Mole)
- Intensité lumineuse (Candela)

Pour le domaine du génie logiciel, la mesure est fondamentale. En effet, elle permet d'établir des estimations pour avoir une idée sur l'effort requis en termes de ressources financières, matérielles et humaines pour le développement des projets. De plus, elle permet de prévoir les délais nécessaires pour leur réalisation et gérer l'avancement.

Au fil des temps, plusieurs définitions de la mesure des logiciels ont été proposées par des chercheurs et des praticiens. Chacun des auteurs a sa propre vision pour la mesure. Selon Barry Boehm*, « La mesure de logiciel est un processus continu de définir, de collecter et d'analyser des données sur le processus de développement de logiciel et ses produits pour comprendre et contrôler le processus et ses produits, et pour fournir des informations significatives pour les améliorer».* (Boehm, 2000)

Oman et Pfleeger ont mentionné dans leur livre (Oman *et* Pfleeger, 1997) que les raisons principales pour la mesure sont :

- Mesurer pour la compréhension
- Mesurer pour l'expérimentation
- Mesurer pour le contrôle des projets

[1] www.wikipedia.org

- Mesurer pour l'amélioration des processus
- Mesurer pour l'amélioration des produits
- Mesurer pour la prédiction

Selon eux, les trois raisons majeures pour mesurer sont:

a. **La compréhension** : La mesure d'un morceau logiciel ou d'un produit nous permet de comprendre les fonctionnements lors des phases de développement et de maintenance des logiciels. Cela permet de mieux évaluer le succès du projet en termes de temps et budget pour améliorer les processus de développement et de maintenance afin d'avoir des résultats satisfaisants en termes de qualité et coût.

b. **La prédiction** : Mesurer pour des fins de prévisions consiste à prévoir l'effort requis pour le projet en termes de ressources matérielles, humaines et financières à une étape assez précoce dans son cycle de vie et prévoir de nouvelles activités pour des projets futurs.

c. **Le contrôle :** Les mesures permettent de contrôler l'état d'avancement d'un projet grâce aux prédictions pour éviter des erreurs produites lors du développement ou de la maintenance afin de les rectifier tôt dans le cycle de vie du logiciel et de les éviter dans des projets futurs.

1.3 Historique des mesures en Génie Logiciel

Selon Oman et Pfleeger (Oman *et* Pfleeger, 1997), les mesures peuvent être appliquées sur les projets, pour savoir par exemple si un projet a été réalisé sur de bonnes pistes en termes de qualité, coûts et délais. Elles peuvent être également utilisées pour mesurer l'efficacité des processus, de même pour les produits.

Au fil du temps, le domaine du logiciel a connu plusieurs types de mesures. Parmi lesquelles, la méthode LOC (nombre de lignes de code - Length Of Code), UCP (le point de cas d'utilisation – Use Case Point), FPA (analyse de point de fonction – Function Point Analysis), COSMIC, MkII - ISO/IEC 20968, NESMA - ISO/IEC 24570, etc. Nous allons donner un bref aperçu sur quelques méthodes ci-dessous :

a. **Mesures de longueur ou lignes de code (LOC)** : Ces méthodes servent à mesurer la taille d'un programme en dénombrant le nombre de lignes de son code source par exemple ou le nombre de pages dans son cahier de charge. Elles sont simples, faciles à compter, et très faciles à comprendre. Mais plusieurs auteurs ont affirmé que les mesures avec les méthodes de longueur ne sont pas très pertinentes. En effet, le nombre de lignes peut varier d'un programmeur à un autre et d'un langage de programmation à un autre. Ce qui cause l'obtention de mesures différentes.

b. **Mesures de fonctionnalités (MTF)** : Ces méthodes servent à mesurer la taille d'un programme en quantifiant les besoins fonctionnels de ses utilisateurs, qui sont un sous-ensemble des besoins des utilisateurs. Les mesures de taille fonctionnelle sont utilisées pour comparer la productivité des projets informatiques, pour l'estimation de l'effort du projet et pour le contrôle et le suivi des changements fonctionnels tout au long du cycle de vie du projet (Sellami *et al.*, 2006).

- Exemple : IFPUG, COSMIC, MkII , NESMA

1.4 Les mesures de la taille fonctionnelle

1.4.1 Évolution des méthodes de mesure de la taille fonctionnelle

Depuis les années 80, plusieurs méthodes de mesure de la taille fonctionnelle sont apparues. La figure 1.1 montre l'évolution de ces méthodes à travers le temps.

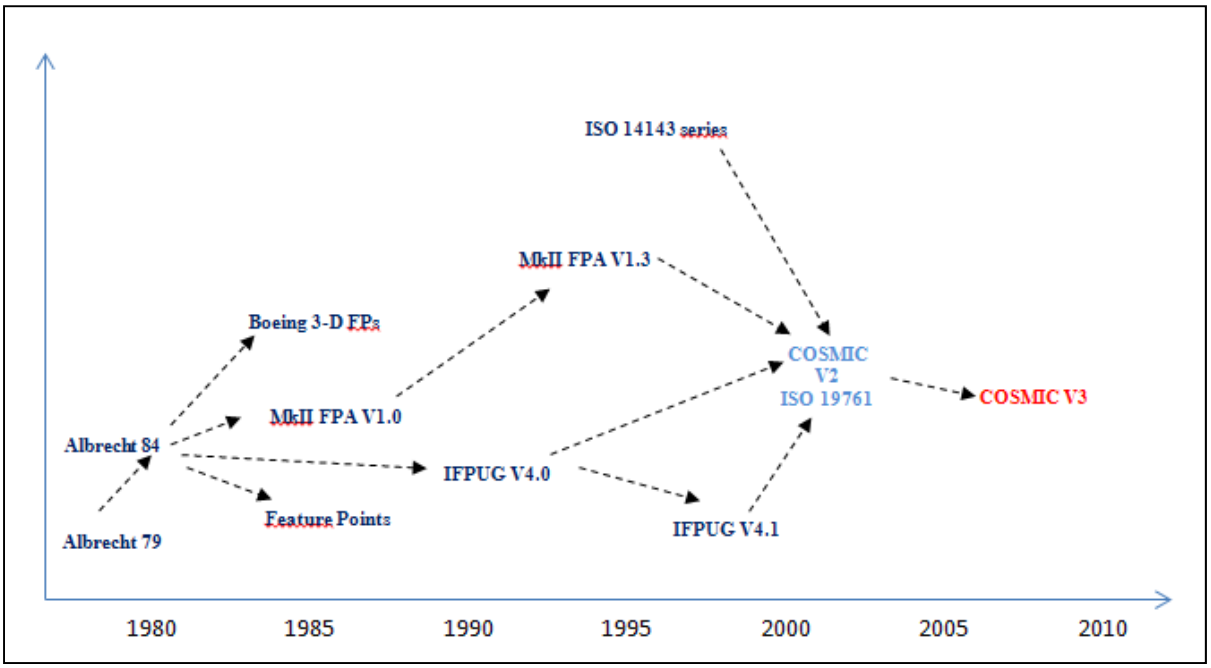


Figure 1.1 Évolution des méthodes de mesures de la taille fonctionnelle tirée de (Abran, 2010)

Parmi les méthodes citées ci-dessus, cinq ont franchi la normalisation ISO (Abran, 2010):

- ISO 29881: FISMA - Finish Software Measurement Association.
- ISO 24570: NESMA.
- ISO 20968: Mk II.
- ISO 20926: FPA (exp : IFPUG)
- ISO 19761: COSMIC

1.4.2 La méthode des points de fonction IFPUG

1.4.2.1 Principes

Dans les années 70, Allan Albrecht a introduit une méthode de mesure de la taille fonctionnelle des logiciels appelée Analyse des Points de Fonction. Au fil du temps, l'utilisation de cette méthode a augmenté et a été le sujet de plusieurs recherches. En 1986, l'International Function Point Users Group (IFPUG, 2012) a été créé et a normalisé la méthode originale d'Albrecht pour la mesure fonctionnelle des logiciels. Par la suite, la méthode FPA a franchi la normalisation ISO (International Organization for Standardization) en 2003, c'est la norme ISO 20926.

Qu'est-ce qu'un point de fonction ? C'est la première interrogation qu'un non connaisseur du domaine de mesure peut se poser. Souvent, on définit les points de fonction comme étant une unité de mesure standard qui représente la taille fonctionnelle d'une application logicielle.

"*A way of measuring the function value derived to our customer*". C'est de cette manière qu'a défini Albercht au années 70 la méthode FPA (Lavazza *et al.,* 2010). Evidemment, c'est à ce point-là que réside la nouveauté de la méthode FPA car en effet, on ne mesure plus le programme par son aspect technique (par exemple en calculant le nombre de lignes de code) mais plutôt par son aspect fonctionnel, plus précisément en calculant la taille fonctionnelle, exprimée en points de fonction, et indépendamment du langage utilisé ou du degré d'expérimentation du programmeur.

En effet, la méthode FPA permet de mesurer la taille du logiciel en quantifiant les fonctionnalités fournies à l'utilisateur à partir des spécifications fonctionnelles. Notons bien que sur la base des points de fonction nous pouvons déterminer entre autres les points suivants:

- **Productivité-performances du projet** : nombre de jours/ hommes dépensés.
- **Coût** : nombre de jours/ hommes en termes de coût.
- **Réactivité** : durée du projet.
- **Qualité** : nombre d'anomalies en production

Le modèle FPA distingue trois types de transactions (Entrée externe, Sortie externe, enquête externe) tel qu'illustré dans le schéma ci-dessous.

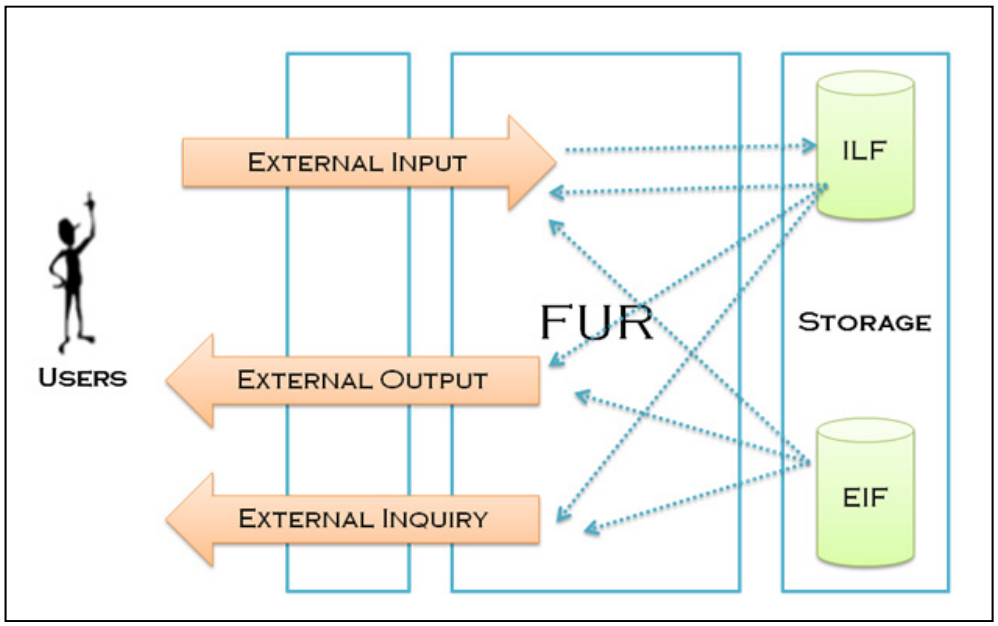


Figure 1.2 Les types de transactions et leurs relations dans le modèle FPA

- **Entrée externe (EI)** : un processus dans lequel les données élémentaires traversent la frontière de l'extérieur vers l'intérieur.
- **Sortie externe (EO)** : un processus élémentaire dans lequel des données dérivées traversent la frontière de l'intérieur vers l'extérieur.
- **Enquête externe (EQ)** : un processus élémentaire dans lequel les données récupérées traversent la frontière de l'intérieur vers l'extérieur.

1.4.2.2 Avantages et inconvénients

La méthode des points de fonction présente plusieurs avantages. En effet, elle a franchi la normalisation internationale, pas comme d'autres méthodes de mesure (LOC par exemple). Elle permet également d'effectuer les mesures indépendamment des technologies et des méthodologies employées lors du processus de développement, ce qui n'est pas le cas de la méthode du nombre de lignes de codes par exemple. FPA peut être employée pour déterminer le degré de productivité d'un outil, un environnement ou un langage de programmation par rapport à d'autres dans une même organisation ou d'autres organisations (Khelifi, 2005).

D'un autre côté, la méthode des points de fonction peut être vue insatisfaisante par rapport à d'autres points. Effectivement, la méthode présente une faiblesse en ce qui concerne la procédure de comptage, car elle nécessite un comptage manuel ce qui rend son automatisation chose difficile. De plus, elle exige de l'expérience pour avoir des mesures précises (Khelifi, 2005). Enfin, la méthode d'analyse des points de fonction n'est pas applicable à tous les domaines tels que le domaine des logiciels en temps réel, le domaine des applications web, le domaine des systèmes embarqué, etc. Elle n'est disponible que dans le domaine d'informatique de gestion et elle permet de calculer les fonctionnalités des systèmes d'information.

Plusieurs tentatives d'utilisation de la méthode FPA ont été effectuées sur les logiciels à temps réel mais elles ont donné des résultats insatisfaisants. Assurément, pour ce type de logiciels, FPA n'est pas capable d'effectuer un nombre très élevé de calculs internes et/ou des fonctions de contrôle (Maya *et al.*, 1998), ce qui génère des mesures non convenables.

Selon les auteurs (Maya *et al.,* 1998), la méthode FPA ne prend pas en compte les caractéristiques spécifiques aux applications de types temps réel. En effet, les applications de type temps réel produisent un grand nombre de sous-processus alors que la méthode ne prend pas en compte ces sous-processus ce qui cause la génération des petites tailles de points de

fonction dans ces types d'environnements. Donc, il a été indispensable de proposer une extension pour la méthode afin de remédier ce problème.

1.4.3 La méthode COSMIC

1.4.3.1 Principes

Plusieurs chercheurs dans le domaine de logiciels ainsi que différentes organisations en industrie ont constaté que les premières générations des méthodes de mesure de la taille fonctionnelle, basées sur les points de fonction, n'offrent pas une mesure conforme aux fonctionnalités des systèmes embarqués et à temps réel. De plus, plusieurs auteurs ont convenu que la structure de la méthode des points de fonction n'été pas adéquate pour mesurer des logiciels de traitement intensif avec un grand nombre de fonctions de contrôle et des calculs internes (Desharnais *et al.*, 1998) et c'est à partir de ces raisons qu'est survenue l'idée de COSMIC.

En effet, le prototype conçu pour la nouvelle génération de mesure de la taille fonctionnelle COSMIC a gardé les critères de qualité de la méthode IFPUG tels que la pertinence. La méthode COSMIC prend en compte les sous-processus intégrés au sein d'un processus de contrôle unique, elle permet évidemment d'identifier les différents groupes de données reçus, envoyés, lus et écrits et donc de générer plus de points de fonction. De plus, partant du fait que le degré de granularité est important pour les logiciels à temps réel, la méthode COSMIC considère un degré de granularité très fin, celui des sous processus.

COSMIC est une méthode de mesure de la taille fonctionnelle des logiciels maintenue par le groupe COSMIC regroupant un ensemble d'experts du domaine des métriques à travers le monde et dirigé par le Professeur Alain Abran (École de Technologie Supérieure,) et Charles Symons (Software Measurement Services Ltd, UK). Elle est devenue depuis mars 2003, un standard ISO pour la mesure de la taille fonctionnelle des logiciels: c'est la norme ISO 19761.

La méthode de mesure COSMIC applique un ensemble de procédures, modèles et règles pour un logiciel donné à travers ses fonctionnalités utilisateurs requises. Selon la perspective proposée par COSMIC, le logiciel est une partie d'un produit ou service dédiée pour satisfaire les exigences fonctionnelles des utilisateurs (Abran, 2010).

Les FUR peuvent être allouées pour la partie matérielle ou logicielle ou les deux en même temps comme le montre la figure ci-dessous.

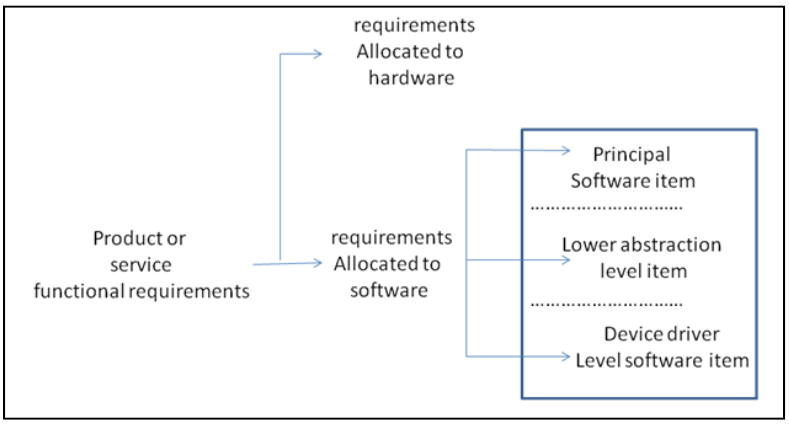


Figure 1.3 Allocation des Fonctionnalités Utilisateurs Requises tirée de (Abran, 2010)

En pratique, les FUR peuvent être dérivés parfois à partir de la documentation avant l'existence du logiciel. Ainsi, la taille fonctionnelle du logiciel peut être mesurée avant son implémentation (Abran, 2010). D'autre part, si une partie d'un logiciel déjà existant doit être mesurée, les FUR peuvent être dérivés des artefacts installés déjà sur le système. Ces deux principes sont illustrés dans le schéma ci-dessous.

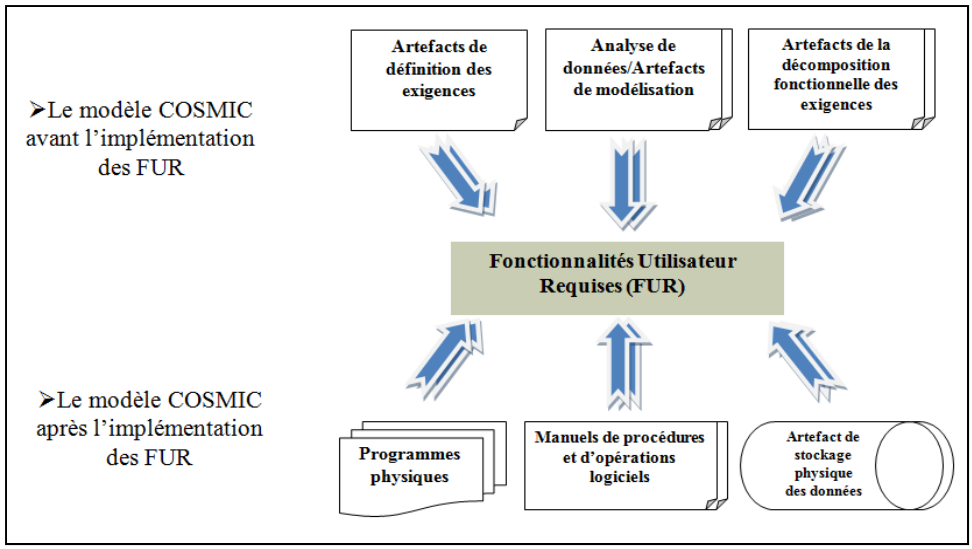


Figure 1.4 Le modèle COSMIC avant et après l'implémentation des FUR

1.4.3.2 Caractéristiques de la méthode COSMIC

La méthode COSMIC est venue pallier aux différentes faiblesses des autres méthodes de mesure. Son principal avantage est qu'elle est applicable à n'importe composant ou logiciel que ce soit de type affaire, temps réel ou autre. De plus, elle peut être utilisée pour différentes fins à n'importe quel moment dans le cycle de vie du logiciel. Plus spécifiquement, elle peut être appliquée sur des nouveaux projets ou sur des projets existants en maintenance ou encore sur des projets déjà achevés. De plus, le principe de la méthode COSMIC est simple: elle est basée sur le point de vue de l'utilisateur fonctionnel, celui qui demande la fonctionnalité mesurée. Le principe de cette méthode est de calculer le nombre de mouvements de données ou encore sous processus fonctionnels échangés entre l'utilisateur et le logiciel et entre le logiciel et la partie stockage. Le calcul de la taille fonctionnelle est un simple calcul.

1.4.4 Comparaison entre les méthodes COSMIC et IFPUG

Comme mentionné au début de cette section, parmi les raisons d'apparition de la nouvelle génération de méthode de mesures des logiciels, COSMIC, est son applicabilité aux logiciels autres que les MIS, tel que les logiciels à temps réel et les logiciel hybrides (Temps réel & d'affaire).C 'est l'une des différences qui existent entre les méthodes COSMIC et IFPUG. La méthode COSMIC diffère aussi de la méthode IFPUG en termes de granularité, puisqu'elle a un niveau de granularité très fin celui des sous-processus fonctionnels. C'est évidemment une des faiblesses de la méthode FPA par rapport aux logiciels à temps réel.

De plus, IFPUG mesure selon la vue utilisateur « humaine » uniquement, alors que la méthode COSMIC mesure à partir de différents points de vue : humain, un autre logiciel ou tout autre appareil interagissant avec le logiciel à mesurer (Khelifi, 2005).

1.5 Processus de mesure COSMIC

La méthode COSMIC prend en considération les fonctionnalités du logiciel de point de vue de ses utilisateurs. Son avantage d'être compatible à nombreux types de logiciels indépendamment des technologies utilisées pour le développement, lui a permis de franchir la normalisation internationale pour être la norme ISO 19761.

Le processus de mesure sur lequel est basée la méthode COSMIC se déroule sur trois phases, à savoir, la phase de stratégie de mesure, la phase d'arrimage et enfin la phase de mesure. Une phase reçoit en entrée des paramètres et fournit en sortie d'autres paramètres qui sont des entrées pour la phase qui la suit (Figure 1.5). Nous présenterons dans ce qui suit chacune des phases à part.

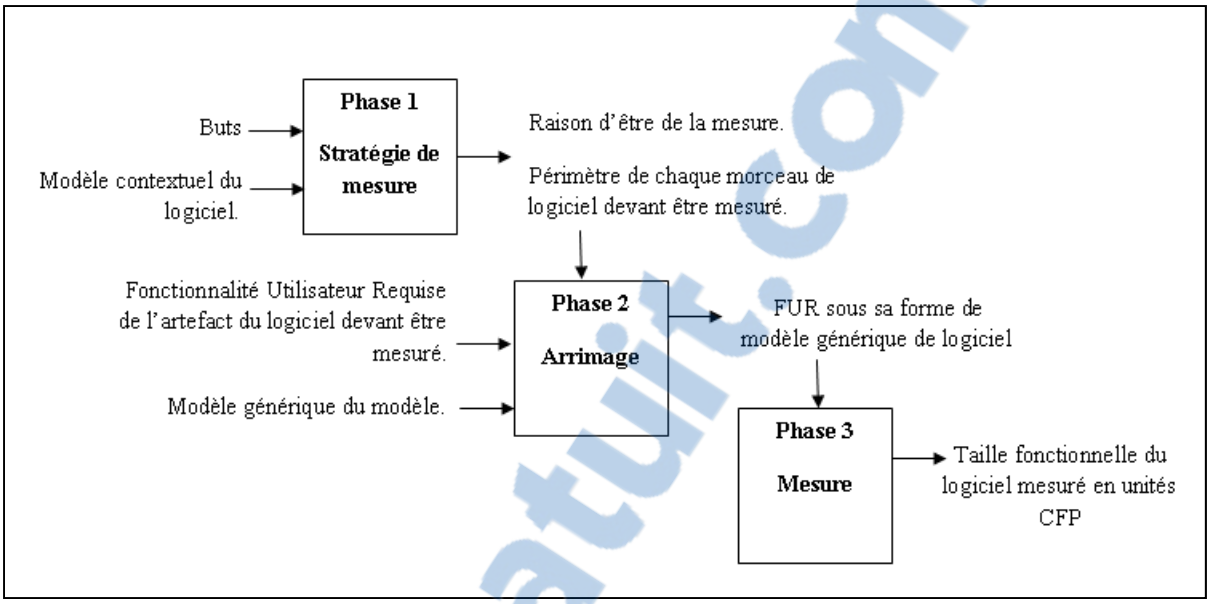


Figure 1.5 Processus de mesure de la méthode COSMIC tirée de Abran *et al.* (2009)

1.5.1 Définir la stratégie de mesure

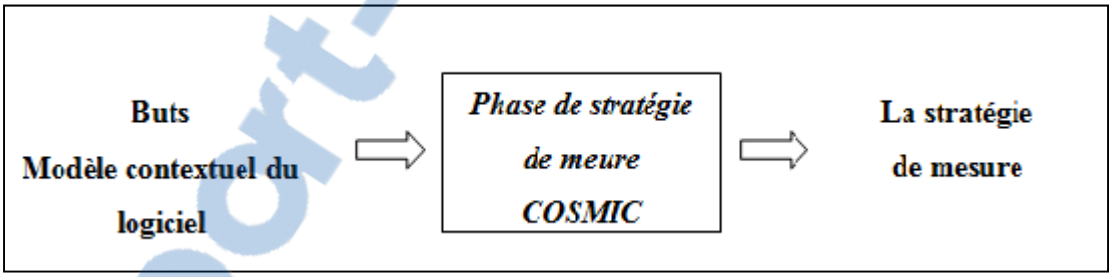


Figure 1.6 La phase de stratégie de mesure COSMIC

Comme tout processus ou projet, il est nécessaire de commencer par définir la stratégie de travail. En ce qui concerne les mesures de la taille fonctionnelle, il est obligatoire avant de commencer le processus de mesure de définir la stratégie de la mesure en précisant quatre paramètres essentiels :

- **La raison d'être de la mesure** : *« Un énoncé qui définit pourquoi une mesure est exigée, et comment le résultat sera employé. »* (Abran *et al*., 2009).

En effet, avant de débuter la construction d'un projet, il faut évidemment définir le « Pourquoi ? » de ce projet. Les raisons de mesure peuvent être différentes tout au long du cycle du logiciel. Et comme COSMIC nous permet de mesurer la taille d'un logiciel même avant son développement, alors nous pouvons par exemple mesurer pour estimer le coût d'un nouveau développement logiciel (Abran *et al*., 2009) comme nous pouvons mesurer après la mise en opération du logiciel pour d'autres raisons.

- **Le périmètre de la mesure** : « *C'est l'ensemble des Fonctionnalités Utilisateur Requises (FUR) qui doivent être incluses dans une occurrence spécifique de mesure de taille fonctionnelle. »* (Abran *et al*., 2009).

Après avoir déterminé la raison d'être de la mesure, il est obligatoire de définir le périmètre en classifiant les FUR. De plus, il est important de déterminer le périmètre de chaque morceau séparément si le logiciel est composé de plusieurs morceaux dans différentes couches. Le niveau de décomposition (Voir glossaire) doit être pris en compte dans certains cas aussi.

- **Les utilisateurs fonctionnels** *: « Un (type d') utilisateur qui envoie et/ou est un récepteur de données des Fonctionnalités Utilisateurs Requises d'une partie du logiciel. »* (Abran *et al*., 2009).

La taille fonctionnelle d'un morceau logiciel peut changer d'un utilisateur à un autre en employant des fonctionnalités différentes qu'ils expriment à travers leurs FUR.
Un utilisateur fonctionnel est tout utilisateur en relation directe avec le logiciel, il peut être un humain, un autre logiciel ou n'importe quelle machine qui interagit avec le logiciel à mesurer.

L'identification de la frontière du logiciel devient alors chose facile car c'est la partie qui sépare un logiciel ou un morceau de logiciel de ses utilisateurs. La frontière permet alors de distinguer les éléments faisant partie du logiciel à mesurer et ceux qui appartiennent aux utilisateurs du logiciel. Ce principe est illustré dans la figure ci-dessous.

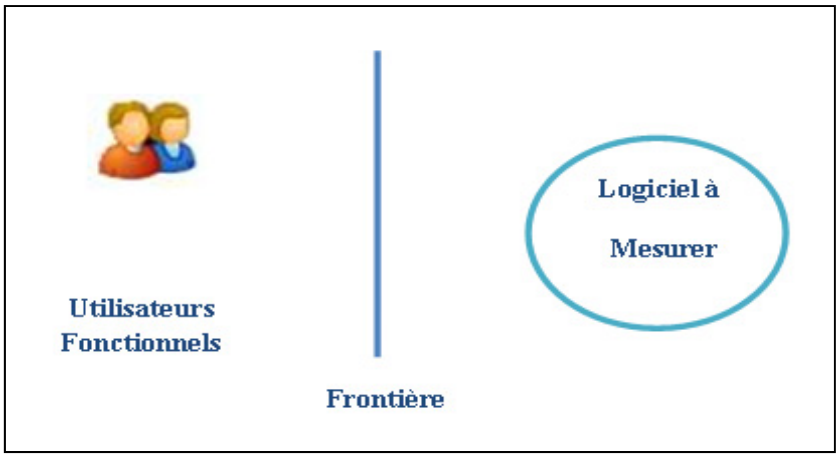


Figure 1.7 Frontière du logiciel à mesurer

- **Le niveau de granularité** : « *Tout niveau d'expansion de la description d'un seul morceau de logiciel (i.e. un énoncé d'une exigence, ou la description d'une structure d'un morceau de logiciel) tel que chaque niveau supérieur d'expansion, la description de la fonctionnalité du morceau du logiciel est à un niveau supérieur et uniforme de détails. »* (Abran *et al*., 2009).

COSMIC définit un niveau de granularité très fin, c'est celui des sous processus. En effet, le niveau de granularité nous permet de voir les FUR avec plus de détails. C'est à ce point-là que la méthode COSMIC a pu répondre à une des exigences des logiciels à temps réel, puisque ce genre de logiciels génère un grand nombre de sous processus fonctionnels qui doivent être pris en compte lors de la mesure pour donner les résultats satisfaisants.

En conclusion, il est indispensable de déterminer les quatre paramètres expliqués ci-dessus avant de commencer une procédure de mesure. La prochaine étape du processus de mesure COSMIC est la phase d'arrimage.

1.5.2 La phase d'arrimage (Mapping en anglais)

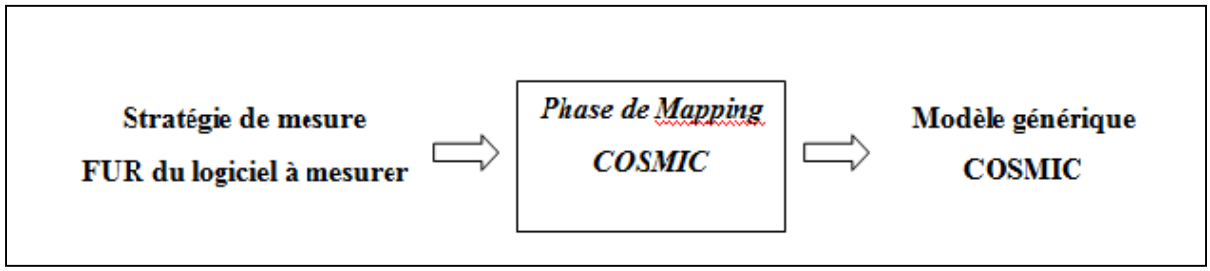


Figure 1.8 La phase de mapping COSMIC

Les résultats de la première phase du processus de mesure COSMIC, à savoir la raison d'être de la mesure, le périmètre, les utilisateurs fonctionnels et le niveau de granularité, tous ces éléments qui constituent la stratégie de la mesure, et les FURs provenant des artefacts du logiciel à mesurer sont les entrées de la phase d'arrimage (ou « mapping » en anglais). Cette dernière consiste à générer en sortie le modèle générique spécifique à la méthode COSMIC (Voir définition dans le glossaire). Cette étape, nécessite l'identification des éléments suivants :

- **Les Processus Fonctionnels : «** *Un composant élémentaire d'un ensemble des Fonctionnalités Utilisateur Requises (FUR), comprenant un ensemble de mouvements de données unique, cohésif et indépendamment exécutable. Il est déclenché par un mouvement de données d'un utilisateur fonctionnel qui informe le morceau de logiciel à mesurer que l'utilisateur fonctionnel a identifié un évènement déclencheur. Il est complet lorsqu'il a exécuté tout ce qui est requis en réponse au type d'évènement déclencheur»* (Abran *et al*., 2009).

En effet, un utilisateur communique avec le logiciel à travers la frontière, mais la question qui se pose est ' Comment l'utilisateur communique avec le logiciel ?' La réponse est simple: les processus fonctionnels sont en réalité initiés par les utilisateurs fonctionnels à travers les événements déclencheurs. Le manuel de mesure COSMIC définit un événement déclencheur

comme étant « *un type d'événement qui induit un utilisateur fonctionnel d'un morceau de logiciel à déclencher un ensemble de processus fonctionnels. »* (Abran *et al*., 2009).

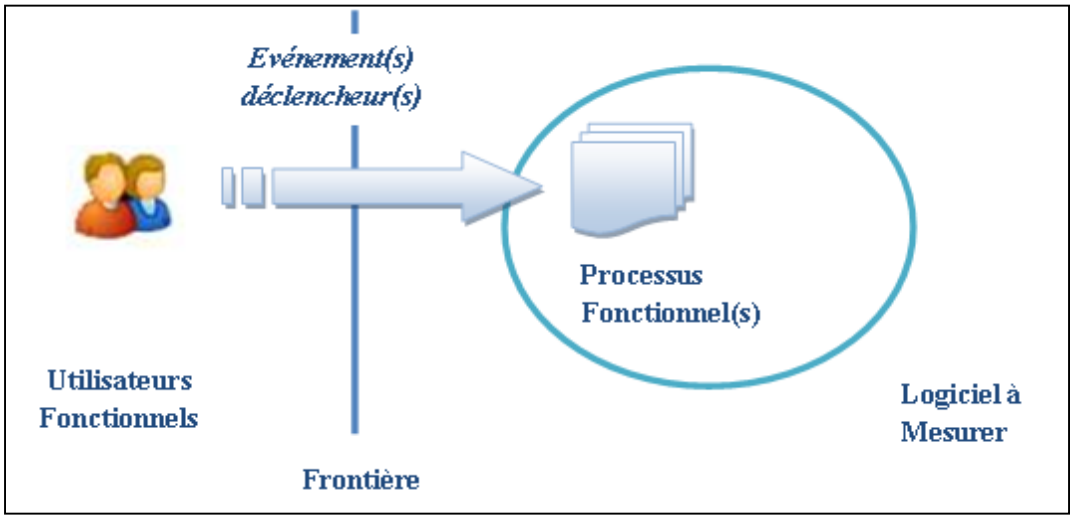


Figure 1.9 Relation entre processus fonctionnel et utilisateur

- **Les Groupes de données (DG):** *« Tout ensemble distinct, non vide, non ordonné et non redondant de types d'attributs de données où chaque type d'attribut de donnée décrit un aspect complémentaire du même objet d'intérêt.»* (Abran *et al*., 2009).

Pour être traitées, les FUR du logiciel à mesurer vont être transmises dans des mouvements de données. L'ensemble des FUR constitue un objet d'intérêt (voir glossaire). Cet objet d'intérêt est constitué d'un groupe de données (DG) qui lui est lié directement.

- **Les Attributs de données** : « *La plus petite parcelle d'information codée, dans un groupe de données, possédant une signification dans la perspective des Fonctionnalités Utilisateurs Requises (FUR) du logiciel. »* (Abran *et al*., 2009).

En effet un groupe de données est constitué d'un ensemble d'attributs non vide, non ordonné et non redondant tel que chaque attribut représente une petite parcelle d'information codée possédant une signification dans la perspective des FUR (Abran, 2010). Notons bien que

l'identification des attributs de données n'est pas une étape obligatoire. Le principe d'objet d'intérêt, groupe et attributs de données est illustré dans la figure 1.10.

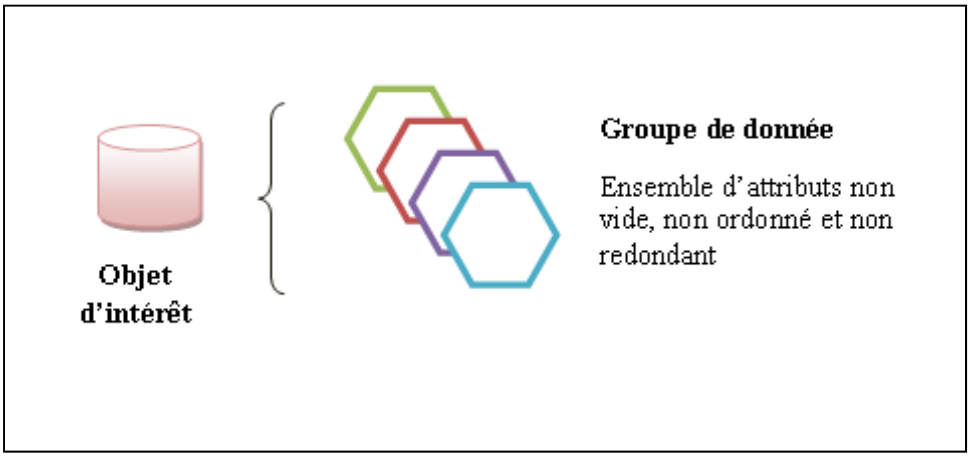


Figure 1.10 Groupe de données et attributs de données

1.5.3 La phase de mesure

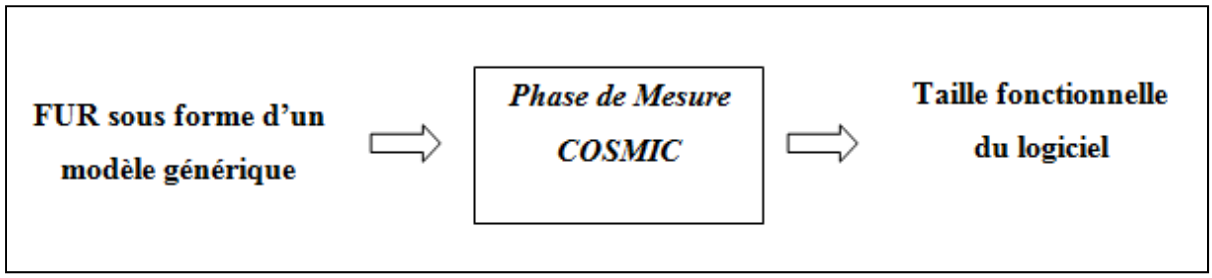


Figure 1.11 La phase de mesure

La dernière procédure du processus de mesure avec la méthode COSMIC est l'étape de mesure. Cette phase est composée de deux étapes, à savoir, l'identification des sous-processus fonctionnels et l'application de la fonction de mesure.

La phase de mesure reçoit en entrée les FUR sous la forme d'un modèle générique et donne en sortie le résultat final du processus de mesure qu'est la taille fonctionnelle du logiciel.

- **Identification des sous processus fonctionnels :**

Le principe COSMIC est de décomposer les fonctionnalités utilisateurs requises en un ensemble de processus fonctionnels où chacun représente un ensemble de sous-processus. Ces derniers peuvent être soit un mouvement de données soit une manipulation de données. Les mouvements de données sont classés en quatre catégories : entrée, sortie, lecture et écriture.

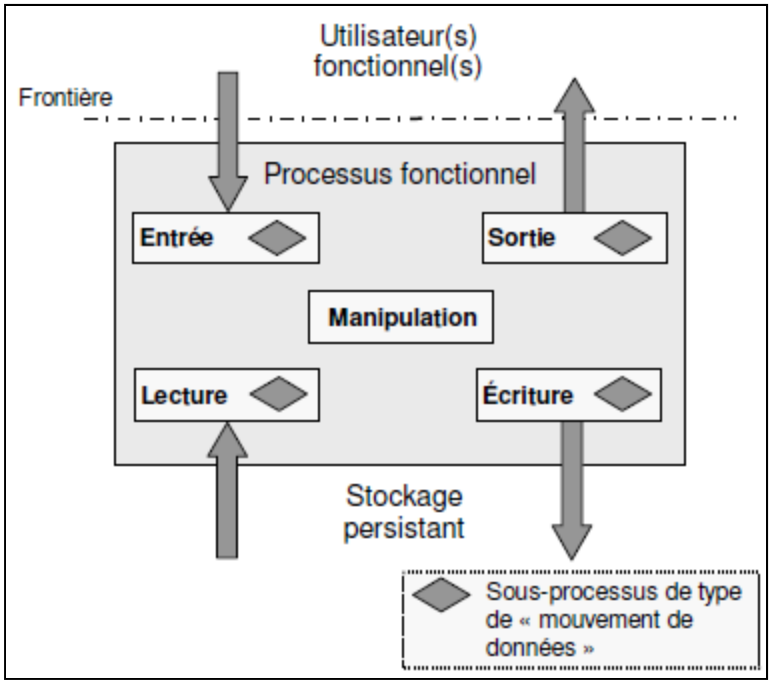


Figure 1.12 Les mouvements de données et leurs interrelations tirée de Abran *et al.* (2009)

Lorsqu'un utilisateur déclenche un événement, un processus fonctionnel sera déclenché par conséquent. Ce processus sera transmis sous la forme d'un groupe de données à travers le mouvement de donnée « entrée » à travers la frontière du logiciel. Dans ce cas, l'ensemble des attributs de données transférés vont informer le logiciel qu'un événement est survenu.
En contrepartie, un logiciel répond à un utilisateur par un mouvement de donnée « sortie » de même à travers la frontière. Une sortie représente surtout les présentations requises par les utilisateurs et les formatages. Les messages d'erreur générés par un logiciel par exemple sont de type « sortie ».

Une fois qu'une information est entrée au logiciel, elle sera traitée et enregistrée dans la partie de stockage persistant. Le déplacement du groupe de données du processus fonctionnel vers le stockage est appelé mouvement d' « écriture ». Par la suite, si un processus fonctionnel demande une information, il va la lire à partir de la partie de stockage où elle a été enregistrée. Ce mouvement de donnée est appelé « Lecture ».

Rappelons qu'un sous-processus fonctionnel peut être soit un mouvement de donnée soit une manipulation de données. En effet, les manipulations de données ne sont pas utilisées en tant que base de composants fonctionnels, COSMIC assume que leurs fonctionnalités sont déjà intégrées parmi les mouvements de données. En conclusion, nous pouvons présenter le processus général de COSMIC comme suit :

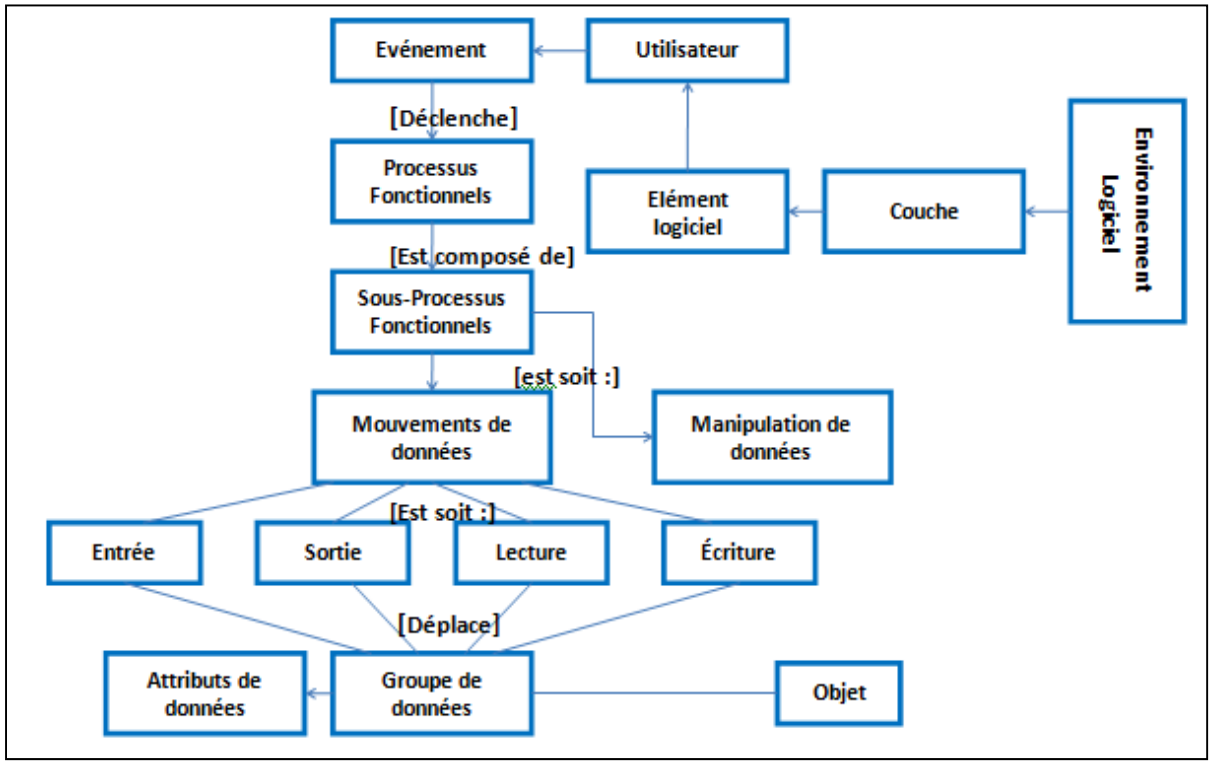


Figure 1.13 Méta-modèle COSMIC tirée de (Symons, 1999)

En effet, un environnement logiciel est composé d'une ou plusieurs couches, chacune de sa part peut contenir un ou plusieurs éléments logiciels. Ces derniers sont utilisés par un ou plusieurs utilisateurs. Un utilisateur va initier un ou plusieurs événements déclencheurs, qui de leurs part vont déclencher un ou plusieurs processus fonctionnels. Rappelons qu'un processus fonctionnel est décomposé en un ou plusieurs sous-processus fonctionnel. Chaque PF peut être soit une manipulation de données soit un mouvement de données.

Chacun des mouvements peut être soit une entrée de données soit une sortie de données. Ou bien, il peut être une lecture de données ou une écriture. De plus, un mouvement déplace un et un seul groupe de données qui est composé d'un ou plusieurs attributs de données et lié à un et un seul objet d'intérêt.

1.5.4 Application de la fonction de mesure

Lors de cette étape, une fonction de mesure est appliquée à tous les mouvements de données identifiés pour chaque processus fonctionnel. La norme de mesure COSMIC appelée unité de mesure est de 1 CFP (COSMIC Function Point). Cette unité de mesure est considéré comme la taille d'un mouvement de données.

Après avoir appliqué la fonction de mesure sur tous les processus fonctionnels, il est facile d'additionner toutes les valeurs associées aux mouvements de données pour avoir la taille fonctionnelle du logiciel à mesurer. Toutefois, un certain nombre de règles doit être appliqués à cette étape (Abran *et al*., 2009), à savoir :

Taille (processus fonctionnel) =

$$\sum \text{taille (Entrées)} + \sum \text{taille (Sorties)} + \sum \text{taille (Lectures)} + \sum \text{taille(Écritures)}$$

1.6 Conclusion

En conclusion, nous avons présenté dans ce chapitre la mesure en génie logiciel et en particulier les mesures de la taille fonctionnelle. Nous avons défini en particulier les méthodes IFPUG et COSMIC puisqu'elles sont les deux dernières méthodes inventées. Nous avons alors défini le principe de chacune à part, ainsi que ses avantages et ses inconvénients, puis nous avons établi une comparaison entre les deux. Le reste de ce mémoire est basé sur la méthode COSMIC.

CHAPITRE 2

L'APPROCHE DIRIGÉE PAR LES MODÈLES ET LES PROFILS UML

2.1 Introduction

À cause que les systèmes informatiques sont de plus en plus complexes, les ingénieurs doivent donner beaucoup d'attention à l'aspect modélisation plutôt que de mettre l'accent sur l'implémentation et la qualité du code. Cela permettra de faciliter le processus de développement de tels systèmes. Dans ce contexte, l'OMG (Object Management Group) (OMG, 2012) a défini l'approche dirigée par les modèles qui favorise l'utilisation des modèles permettant de générer automatiquement du code. L'avantage de MDA (Model Driven Architecture) (MDA, 2012) est que les modèles sont réutilisables, ce qui minimise beaucoup d'effort et de temps. L'une des spécificités de MDA est l'utilisation des langages de modélisation pour exprimer les modèles tels qu'UML et ses profils et les transformations de modèles. Dans ce chapitre, nous allons définir l'approche MDA, la notion de méta-modélisation. Dans ce chapitre, on présente plus spécifiquement les profils UML comme langage de modélisation des systèmes.

2.2 Aperçu de l'approche MDA

L'architecture dirigée par les modèles est une approche proposée par l'OMG (OMG, 2012) pour la conception et la modélisation des systèmes logiciels. L'objectif de MDA est l'élaboration des modèles indépendants des plates-formes d'exécution. L'idée est donc de séparer l'aspect logique (business) de l'aspect technique (code) en utilisant les modèles. On expose un peu plus bas la figure qui décrit le principe de MDA.

Cette architecture est composée de 4 couches. Nous remarquons que les standards de l'OMG à savoir UML, MOF (Meta Object Facility) et CWM (Common Warehouse Metamodel) sont au cœur de MDA. Dans la couche qui suit, se trouve le standard XMI (XML Metadata Interchange) qui permet la communication entre les plates-formes supportées par MDA

(JAVA, .Net, CORBA et web Services). Dans la couche suivante, on trouve les différents services couverts (événements, sécurité, transactions, répertoires). Enfin, dans la dernière couche, résident les différents domaines métiers (finance, transport, santé, ets.).

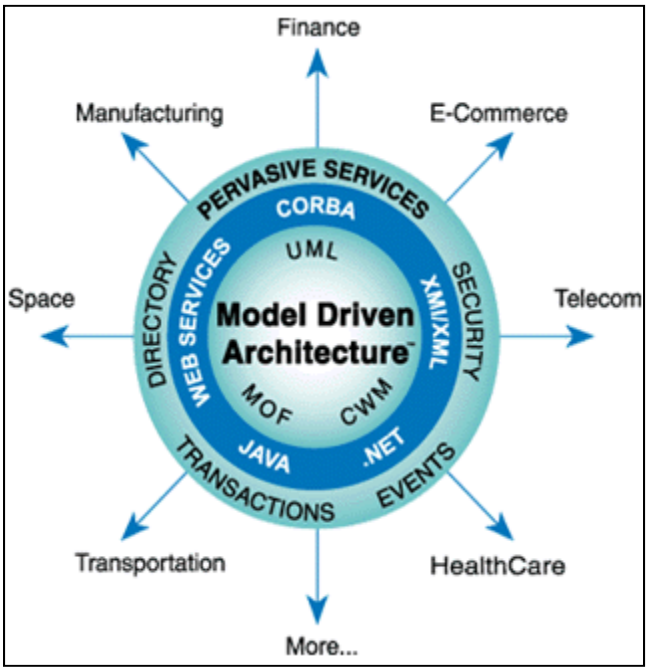


Figure 2.1 L'architecture dirigée par les modèles de l'OMG tirée de (OMG, 2012)

Cette architecture est structurée en quatre couches. Nous remarquons que les standards de l'OMG à savoir UML, MOF et CWM sont au cœur de MDA. Dans la couche qui suit, se trouve le standard XMI qui permet la communication entre les plates-formes supportées par MDA (JAVA, .Net, CORBA et web Services). Dans la couche suivante, on trouve les différents services couverts (événements, sécurité, transactions, répertoires). Enfin, dans la dernière couche, résident les différents domaines métiers (finance, transport, santé, etc.).

Le principe général de MDA est la transformation de modèles, à savoir les transformations CIM vers PIM et PIM vers PSM. Dans un premier niveau, l'approche MDA définit les modèles d'exigences ou CIM (Computation Independent Model) qui sont des modèles construits indépendamment de la programmation. Dans un deuxième plan, MDA définit les modèles d'analyse et de conception PIM (Platform Independent Model) qui sont

indépendants de la plate-forme utilisée. Enfin, les modèles définis dans le troisième niveau sont les modèles de code liés à une plate-forme d'exécution et appelés PSM (Platform Specific Model). La transformation entre les différents modèles est décrite dans la figure 2.2.

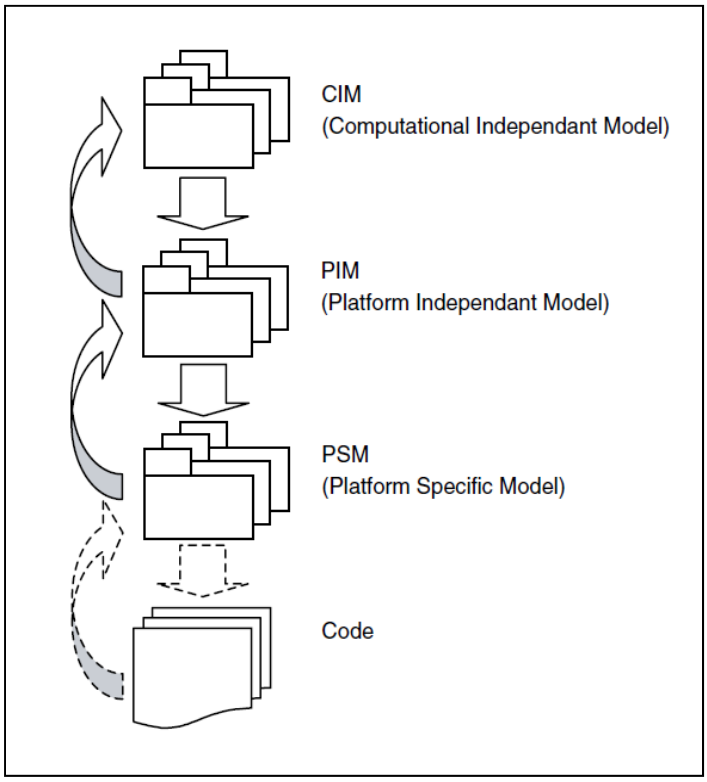


Figure 2.2 Les transformations en MDA tirée de Blanc et Salvatori (2005)

- **CIM** : Ce modèle est appelé modèle d'exigences. Il est indépendant de toute technique informatique. Il représente seulement le modèle métier ou le modèle du domaine. Autrement dit, il décrit les exigences du système.

- **PIM** : Ce modèle est appelé modèle d'analyse et de conception. Il met l'accent sur la fonctionnalité requise par l'utilisateur indépendamment de la plate-forme technique (CORBA, JAVA, .Net…). L'approche MDA préconise l'utilisation d'UML pour construire les PIM à l'aide des diagrammes de classe utilisés souvent pour décrire le domaine métier (Mannox, 2005).

- **PSM**: Ce modèle est appelé modèle de code. Celui-là dépend essentiellement de la plate-forme d'exécution. Le PSM est aligné avec le code de l'application. Il sert essentiellement à la génération du code à partir des modèles vers la plate-forme d'exécution.

En MDA, PIM et PSM peuvent être définis à l'aide des profils UML. Ce mécanisme, à savoir le profil UML, permet à UML d'être adapté à des plateformes d'exécutions différentes telles que CORBA, Java/EJB, XML/SOAP, etc. Par conséquent, selon l'approche MDA, les modèles réalisés à la base de ces profils sont des PSM car ils dépendent des plateformes.

2.3 Unified Modeling Language (UML)

UML (UML, 2012) est le langage de modélisation le plus utilisée pour l'analyse et la conception objet des systèmes logiciels. La première version d'UML a été adoptée par l'OMG en 1997 après avoir tenté de rapprocher les méthodes OMT, BOOCH et OOSE entre 1994 et 1996 (Gabay, Joseph et David, 2008). UML propose un ensemble de diagrammes qui permettent de spécifier et de documenter les systèmes.

UML 2 est une migration qui a apporté beaucoup de nouveautés à UML. Elle a été adoptée par l'OMG en 2003. Entre autres, cette nouvelle version supporte bien l'approche MDA ce qui a facilité le support des systèmes complexes. Depuis la migration vers cette version, UML est devenu composé de deux standards: l'infrastructure (UML Infrastructure, 2012) qui décrit le noyau d'UML (les diagrammes et les modèles UML), et la superstructure (UML Superstructure, 2012) qui spécifie l'architecture de méta-modélisation d'UML (concepts de modélisations qui constituent UML).

Un des aspects intéressants d'UML est qu'il définit des mécanismes d'extensions qui donnent la possibilité de l'étendre et l'adapter à n'importe quel domaine ou plate-forme d'exécution à l'aide de stéréotypes par exemple. Cette notion est fortement utilisée en MDA

puisqu'elle est utilisée pour les transformations de modèles. Nous détaillerons, les profils UML un peu plus loin dans ce chapitre.

2.4 La méta-modélisation en UML

Le standard UML est caractérisé par son approche de méta-modélisation. En effet, un méta-modèle est un modèle qui représente la structure et la sémantique d'un autre modèle. Par exemple, tout modèle UML est une instance du méta-modèle UML.

Selon MOF, un méta-modèle définit la structure que doit avoir tout modèle conforme à ce méta-modèle (Blanc et Salvatori, 2005). Cette approche de méta-modélisation est définie par l'OMG sous forme d'une architecture à quatre niveaux comme le montre la figure 2.3.

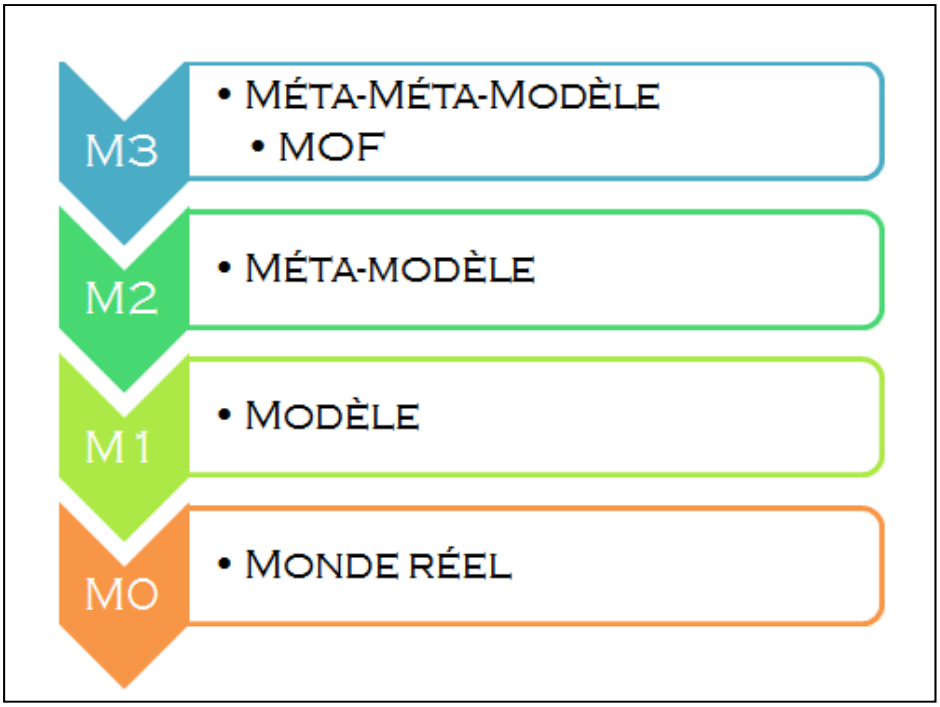


Figure 2.3 Architecture à 4 niveaux

- **Le niveau M0 :** Ce niveau représente le monde réel. Il est composé des instances du système qu'on veut modéliser. On l'appelle souvent modèle d'objet, car il contient les objets réels.

- **Le niveau M1 :** Ce niveau représente le modèle du système à concevoir. C'est à ce niveau là qu'on représente les informations du niveau M0. Les modèles UML sont utilisés souvent à ce niveau (comme on peut utiliser d'autres modèles).

- **Le niveau M2 :** Ce niveau représente la définition du modèle M1, appelé plus spécifiquement méta-modèle. Le méta-modèle UML appartient à ce niveau M2. Dans ce cas, M2 définit entre autre la notion de «Class», «Association», etc. (De même, on peut définir d'autres méta-modèles à ce niveau autres que le méta-modèle UML)

- **Le niveau M3 :** Dans ce niveau, on trouve la définition du MOF, c'est le méta-méta modèle de M1. C'est le dernier niveau de l'architecture à quatre niveaux. MOF est un langage qui définit les langages de modélisation. Le méta-modèle UML est une instance du MOF où chaque méta-classe UML est une instance d'un élément dans *InfrastructureLibrary* (UML Infrastructure, 2012).

En effet, MDA préconise le formalisme de méta-modélisation, c'est-à-dire modéliser les modèles eux mêmes. La figure 2.4 présente un exemple d'utilisation réelle de l'architecture expliquée précédemment.

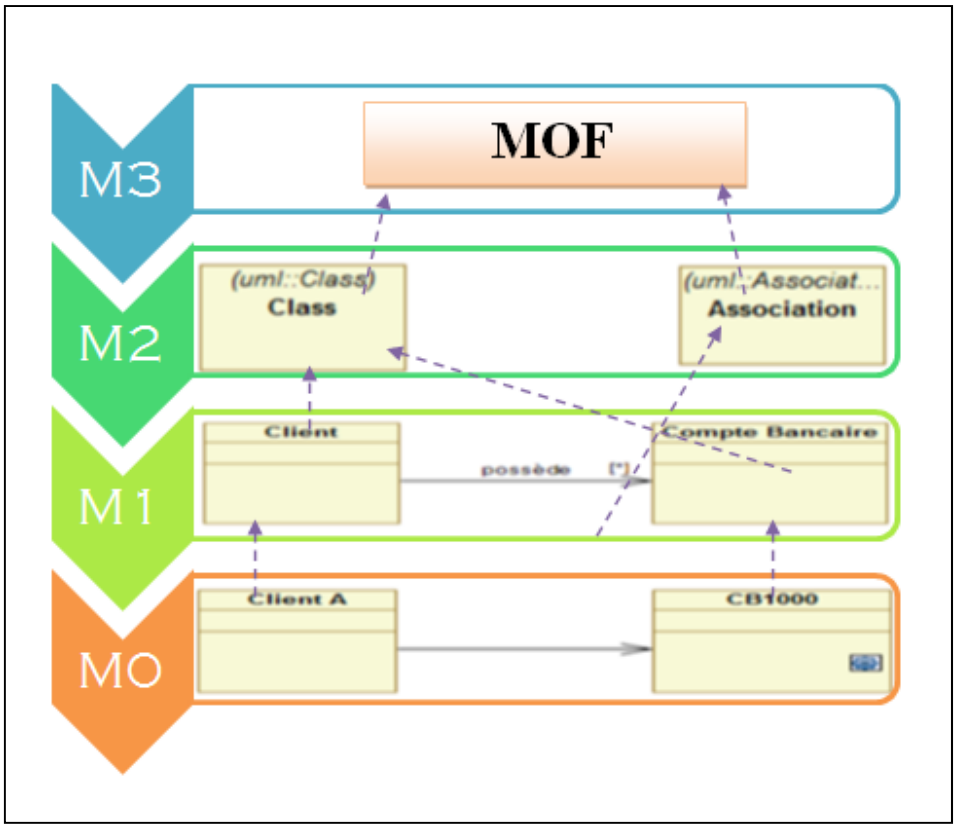


Figure 2.4 Exemple de l'architecture à 4 niveaux de MDA

2.5 Les profils UML

La notion de profil UML a été introduite dans les premières versions de la spécification UML. OMG définit la notion de profil comme suit (UML Infrastructure; UML Superstructure, 2012): *«The Profiles package contains mechanisms that allow metaclasses from existing metamodels to be extended to adapt them for different purposes. This includes the ability to tailor the UML metamodel for different platforms (such as J2EE or .NET) or domains (such as real-time or business process modeling). The profiles mechanism is consistent with the OMG Meta Object Facility (MOF) ».*

Souvent, les développeurs ont du mal à exprimer des besoins spécifiques à leur domaine en se basant seulement sur les éléments UML. Une solution à ce problème consiste à utiliser un des mécanismes d'extension d'UML qui est le profil. Ce dernier est donc un ensemble

d'extensions qui permet à un utilisateur d'un domaine particulier (temps réel, BPM, etc.) d'entreprendre les activités liées à son domaine tout en utilisant UML. De plus, les profils sont utilisés pour permettre d'adapter le méta-modèle UML à différentes plateformes techniques (J2EE, .Net, etc.). Le profil permet donc d'utiliser le méta-modèle UML pour modéliser des besoins particuliers (des concepts, des signes, des propriétés particulières, etc.).

La notion de profil a été définie comme étant un mécanisme d'extension légère du standard UML. Dans les spécifications d'UML 2 (UML Infrastructure; UML Superstructure, 2012), cette notion a été considérée comme une technique spécifique de méta-modélisation. La figure ci-dessous décrit les classes qui composent le package «profile».

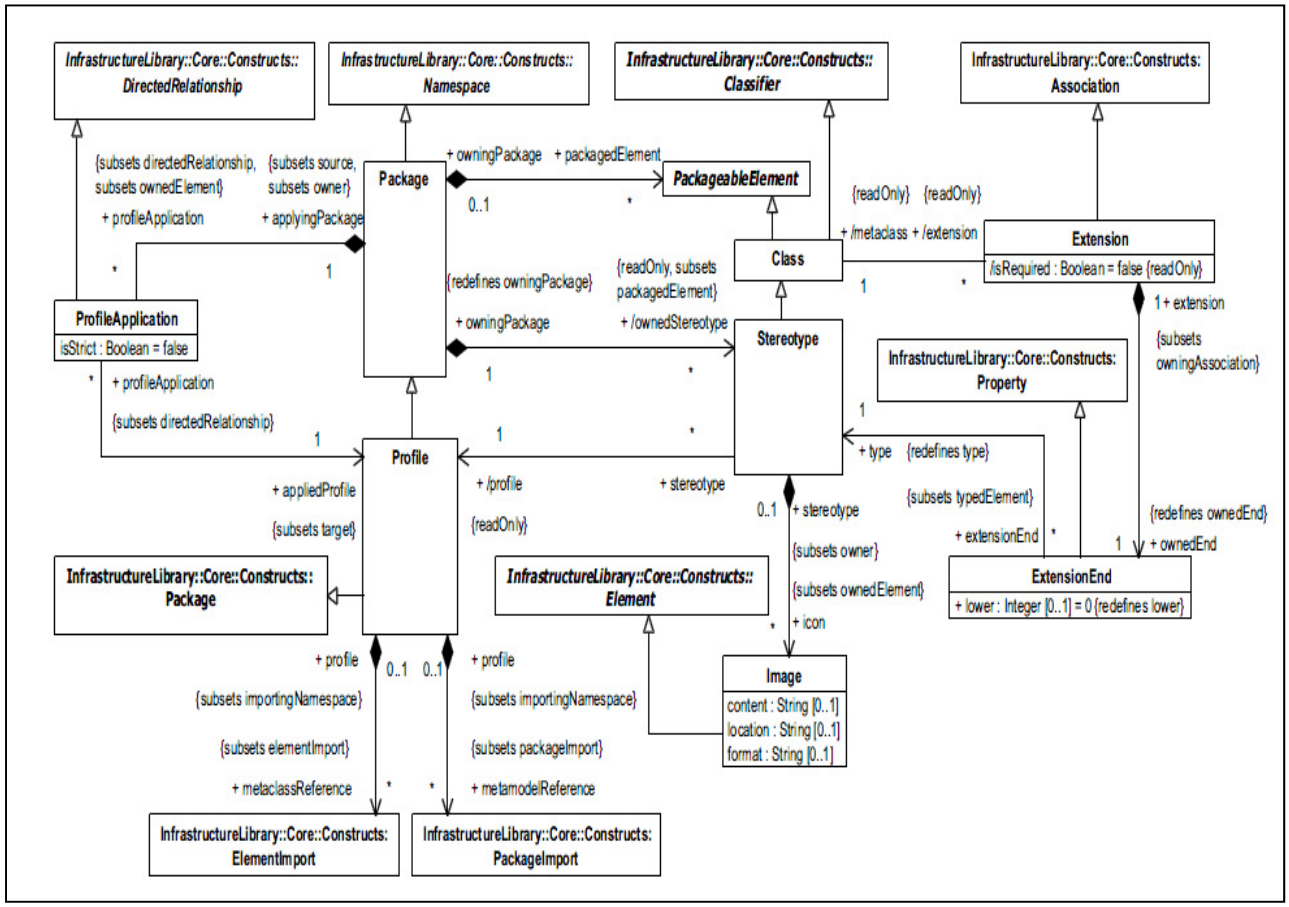


Figure 2.5 Les classes définies dans le package «profile» tiré de UML Infrastructure (2012)

Entre autres, le mécanisme d'extension par les profils permet les transformations des PIMs au PSMs puisqu'il permet de rendre un modèle conforme avec une plate-forme bien déterminée. C'est pour cette raison que l'approche MDA préconise l'utilisation des profils UML pour faire ce genre de transformations (Blanc et Salvatori, 2005).

Un profil UML est représenté sous la forme d'un package stéréotypé avec le mot «profile» (voir figure 2.6). Il est essentiellement basé sur les trois éléments suivants : les stéréotypes, les contraintes et les valeurs marquées (Tagged Values en anglais). Nous allons définir ces concepts dans ce qui suit.

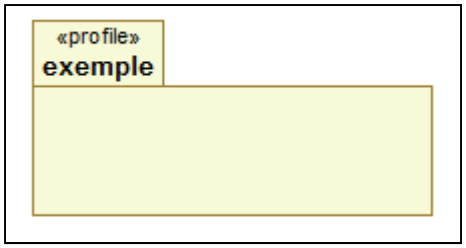


Figure 2.6 Modélisation du package profile

2.5.1 Stéréotypes

En effet, un stéréotype est une classe stéréotypée avec le mot «Stéréotype» (voir figure 2.7). Selon l'OMG (UML Infrastructure, 2012), un stéréotype définit comment une méta-classe existante peut être étendue et permet de modéliser les éléments spécifiques au domaine à profiler. Un stéréotype peut avoir des propriétés comme les classes. Ces propriétés sont appelées valeurs marquées. De plus un stéréotype ne peut que généraliser ou spécialiser un autre stéréotype (UML Infrastructure, 2012).

- ***La définition d'un stéréotype :***

  Le stéréotype est une entité de base pour le mécanisme d'extension. La définition d'un stéréotype consiste à étendre une méta-classe particulière appartenant un méta-modèle (méta-modèle UML par exemple) pour introduire une nouvelle notation spécifique à un

domaine ou une plate-forme. L'extension d'une méta-classe est décrite par une flèche. La figure 2.7 explique la définition d'un stéréotype.

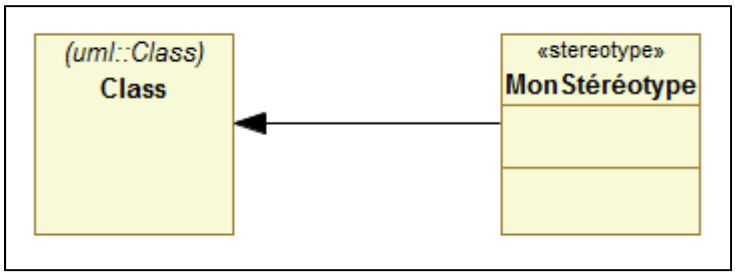


Figure 2.7 Définition d'un stéréotype

- ***Représentation d'un stéréotype :***

  Selon le méta-modèle du package « profile » (figure 2.6), un stéréotype peut être représenté par une image. Suite à cela, un stéréotype peut être représenté avec l'un des trois formats suivants :

  - Icône
  - Icône + texte
  - Texte

- ***Application d'un stéréotype :***

Un stéréotype est appliqué à un élément d'un modèle, c'est-à-dire une instance de la méta-classe sur laquelle le stéréotype est défini (voir exemple dans la figure 2.8). Il est à noter si un profil est appliqué à un modèle alors tout stéréotype appartenant à ce profil peut être appliqué à un élément de ce modèle. Bien évidement si un stéréotype étend la classe « actor » alors seulement les éléments de types « actor » peuvent appliquer ce stéréotype.

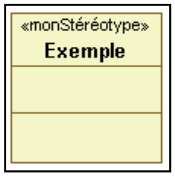


Figure 2.8 Application du stéréotype monStéréotype

2.5.2 Valeur marquée (Tagged Value)

Comme tout attribut, une valeur marquée doit avoir un nom et un type (Fuentes-Fernandez *et al.*, 2004). Parallèlement au concept de stéréotype, un tagged value définit un méta-attribut. Chaque valeur marquée est liée à un stéréotype dans le profil. La figure ci-dessous montre un exemple d'un stéréotype qui contient deux valeurs marquées.

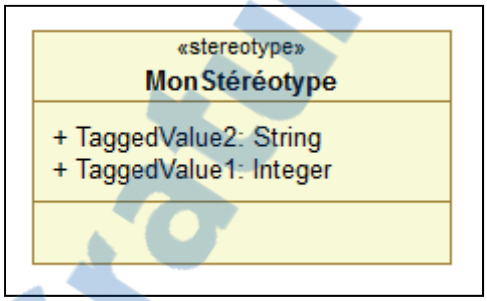


Figure 2.9 Exemple d'un stéréotype avec deux valeurs marquées

2.5.3 Contraintes

Les contraintes sont utilisées pour mettre des restrictions sur des éléments existants dans le méta-modèle ou sur les nouveaux stéréotypes (évidemment sur les tagged values) ajoutés ou encore sur les relations entre les différents éléments du profil. Souvent les contraintes sont exprimées en utilisant le langage OCL (Object Constraint Language). Ce langage s'applique sur les modèles UML et adopté par l'OMG.

2.6 Exemples de profils UML

Plusieurs profils UML existent actuellement tels que :

- SysML : extension d'UML pour l'ingénierie des systèmes
- Profil UML pour CORBA
- Profil EAI pour l'intégration des applications des entreprises
- CCM (CORBA Component Model)
- Profil MARTE pour les applications à temps réel et les systèmes embarqués

CHAPITRE 3

REVUE DE LITTÉRATURE

3.1 Introduction

Notre projet de recherche comporte deux volets principaux. Nous visons d'abord à étudier les principaux travaux de recherche antérieurs qui ont porté sur la façon de rapprocher la mesure de la taille fonctionnelle avec le formalisme UML. Ensuite, nous nous concentrons sur notre proposition d'une nouvelle approche permettant d'automatiser la mesure de la taille fonctionnelle basée sur la méthode COSMIC en utilisant les modèles UML. Dans ce chapitre, nous nous intéressons à faire une étude sur les principales propositions axées sur le rapprochement UML/COSMIC.

Notons que cette étude a été partiellement publiée dans (Barkallah *et al*., 2011). Un survol similaire a été introduit dans le travail de Marin *et al.* (2008) pour reporter les procédures de mesures basées sur COSMIC pour fournir un guide aux praticiens. Entre autres les auteurs ont exposé quelques travaux basés sur UML. Nous concluons ce chapitre par une analyse des travaux étudiés. Une comparaison entre ces travaux sera aussi présentée en résumé.

3.2 Les approches proposées pour mesurer les FUR documentés avec les diagrammes UML

3.2.1 Mapping entre les concepts UML et COSMIC (Bévo *et al.*)

Dans leur travail, Bévo et al. (1999) ont établi principalement un mapping entre les concepts UML et COSMIC. Notons que ce travail, introduit en 1999, été une première initiative de proposer l'utilisation des modèles UML pour la mesure de la taille fonctionnelle avec COSMIC. Le mapping établit consiste principalement à rapprocher le concept de processus fonctionnel à celui de cas d'utilisation en UML. Les mouvements de données de leur côté ont été rapproché au concept de scenario. Quant au concept de groupe de données, il est mappé

au concept de classe UML et par conséquent, un attribut de données correspond automatiquement à un attribut de classe UML. Les auteurs ont choisi aussi de rapprocher la frontière du logiciel au diagramme de cas d'utilisation, car ce diagramme montre bien les utilisateurs du logiciel. Ce qui fait qu'un utilisateur fonctionnel est proche du concept d'acteur en UML.

Cependant, le travail n'a pas été complété puisque deux concepts de COSMIC n'ont pas été rapprochés à des concepts UML, ce sont les concepts de couche et événement déclencheur. Les auteurs affirment que ces deux éléments ne peuvent pas avoir de correspondants en UML.

Le travail a été résumé par un exemple d'application d'affaire, qui est un système de contrôle d'accès. La mesure de la taille fonctionnelle a été établie en se basant sur les modèles UML de l'application. Une première mesure a été basée sur les cas d'utilisation et a donné une taille de 32 CFP et une deuxième sur les scénarios a donné 68 CFP. Cette différence entre les deux mesures nous mène à conclure qu'il devrait avoir un problème dans le mapping du concept de mouvement de données, ce qui signifie que probablement le scénario n'est pas le bon élément UML qui corresponde à ce concept.

Le tableau 3.1 résume le mapping établit par Bévo *et al.*

Tableau 3.1 Rapprochements COSMIC / UML selon Bévo *et al.* (1999)

CONCEPT COSMIC	CONCEPT UML
Processus fonctionnel	Cas d'utilisation
Mouvement de donnés	Scénario
Déclencheur	Pas d'équivalent
Groupe de données	Classe
Attributs de données	Attribut d'objet
Frontière	Diagramme de cas d'utilisation
Utilisateur	Acteur
Couche	Pas d'équivalent

3.2.2 Usage du diagramme de séquence pour adresser le problème de granularité des cas d'utilisation (Jenner)

Le travail de Jenner (Jenner, 2001) vient comme une extension du travail de celui de Bévo *et al.* (1999). L'auteur a raffiné le mapping entre les concepts de processus fonctionnel et scénario UML. Selon lui, Bévo *et al.* (1999) n'ont pas utilisé la notion de scénario comme spécifié en UML. Il affirme qu'un scénario, comme défini en UML, ce n'est qu'une instance d'un cas d'utilisation. Par conséquent, il a conclu que les deux estimations établies dans (Bévo *et al.*, 1999) utilisent les deux les cas d'utilisation mais avec des niveaux de granularité différents (cas d'utilisation plus détaillé).

L'auteur a suggéré de trouver un autre équivalent UML correspondant au concept de processus fonctionnel: alors il a proposé de représenter chaque cas d'utilisation par son diagramme de séquence. Ce denier représentera un processus fonctionnel où chaque mouvement de données consiste en un message du diagramme. Dans ce cas-là, il n'est plus important de connaitre le niveau de granularité du cas d'utilisation.

Le travail a été appuyé par un modèle développé en utilisant un outil CASE (Computer Aided Software Engineering) pour automatiser la mesure de la taille fonctionnelle en utilisant COSMIC (Jenner, 2002). Cependant, l'exemple n'est pas assez clair et compréhensible. De plus, Jenner n'a pas complété le travail de Bévo *et al.* en ce qui concerne le mapping des concepts de déclencheur et couche. Le tableau 3.2 résume l'extension de Jenner par rapport au travail de Bévo *et al.*

Tableau 3.2 Rapprochements COSMIC / UML selon Jenner (Jenner, 2001)

Concept COSMIC	Concept UML
Processus fonctionnel	Diagramme de séquence
Mouvement de donnés	Message
Couche	Pas d'équivalent
Déclencheur	Pas d'équivalent

3.2.3 RUP pour l'automatisation de la mesure de la taille fonctionnelle avec COSMIC (Azzouz *et al.*)

Azzouz *et al.* (Azzouz *et al.*, 2004) ont choisi d'utiliser RUP (Rational Unified Process) pour automatiser la mesure de la taille fonctionnelle avec COSMIC. L'outil a été implémenté dans l'environnement Rational Rose. Pour concevoir leur approche, ils ont commencé par faire un mapping des concepts COSMIC.

Ils se sont alors basés sur les travaux de Bévo *et al.* (1999) et de Jenner (Jenner, 2001) et ont ajouté une nouvelle extension pour donner un équivalent au concept d'événement déclencheur. Ils ont alors introduit un nouveau stéréotype UML pour saisir ce concept. Toutefois, le concept de couche n'a encore pas d'équivalent et doit être identifié manuellement (Azzouz *et al.*, 2004). De plus, le travail a été complété par une étude de cas mais il n'a pas été testé sur une grande échelle. Le tableau 3.3 présente l'extension proposée par Azzouz *et al.*

Tableau 3.3 Rapprochements COSMIC / UML selon Azzouz *et al.* (2004)

CONCEPT COSMIC	CONCEPT UML
Événement Déclencheur	Nouveau stéréotype 
Couche	Pas d'équivalent : doit être identifié manuellement

3.2.4 Extraction des concepts UML pour des fins d'automatisation de la taille fonctionnelle (Pourquoi mesurer en génie logiciel (Vilus *et al.*)

Le travail de Vilus *et al.* (Vilus *et al.*, 2004) est le complément du travail de Bévo et al. (Bévo *et al.*, 1999). En effet, ils proposent une méthode d'extraction des concepts UML pour des fins de mesure selon le mapping établi par Bévo *et al.*

Un prototype appelé MetricXpert a été développé à l'aide de l'outil Rational Rose pour automatiser la mesure de la taille fonctionnelle avec COSMIC. L'outil permet d'extraire les instances de concepts UML pertinents pour la procédure de mesure à partir d'un fichier XMI. Ce fichier XMI est le résultat de la sérialisation du méta-modèle MOF (Vilus *et al.*, 2004) (transformation du méta-modèle en format MOF à l'aide d'un plug-in par exemple). Enfin les informations extraites par l'outil seront stockées dans une base de données.

L'outil proposé a été testé à l'aide de trois études de cas, dont chacun a été modélisé à l'aide d'un outil CASE (Vilus *et al.*, 2004). Ensuite les méta-modèles correspondants aux modèles d'analyses ont été transformés en fichiers XMI qui seront utilisés pour l'extraction des données pertinentes. Cependant, les études de cas présentent une marge d'erreur de 20% (LATECE, 2011).

La figure 3.1 explique le fonctionnement de l'outil d'extraction basé sur le méta-modèle MOF proposé par vilus *et al.* (2004).

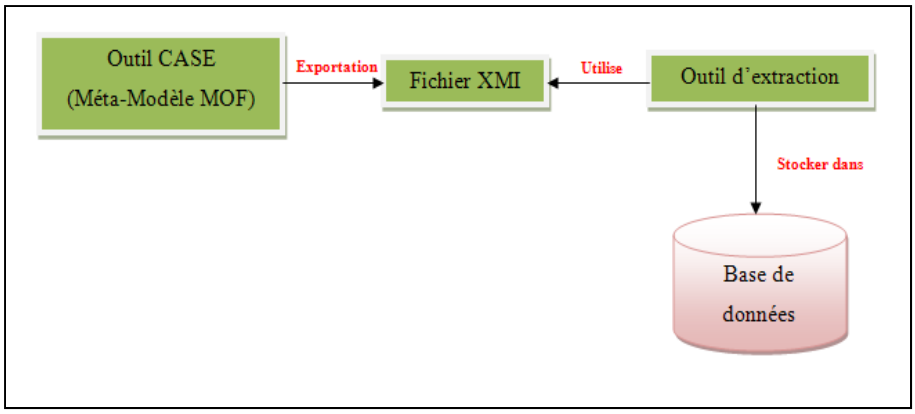


Figure 3.1 Outil d'extraction, proposé par vilus *et al.* (2004)

3.2.5 Mesure basée sur le modèle des cas d'utilisation (Habela *et al.*)

Dans leur travail, Habela *et al.* (Habela *et al.*, 2004) ont proposé d'utiliser le modèle des cas d'utilisation pour la mesure de la taille fonctionnelle avec COSMIC. Ils ont alors proposé un modèle qui prend en considération les règles de gestion, les préconditions ainsi que les post conditions. Ils ont suggéré aussi de prendre en considération les relations entre les cas d'utilisation, à savoir l'extension, l'inclusion et la généralisation afin d'éviter la redondance dans la mesure.

Quant au mapping qu'ils ont proposé, il consiste principalement à rapprocher un cas d'utilisation à un ou plusieurs processus fonctionnel. Cependant, nous pouvons conclure que l'approche proposée n'est pas assez complète pour appliquer la mesure avec COSMIC car le focus a été plutôt mis sur la phase de mesure et inclut les processus de mesure et les mouvements de données. Alors que beaucoup d'éléments essentiels de COSMIC ont été négligés. De plus, le modèle proposé n'a pas été vérifié ni appliqué. L'intéressant dans ce travail c'est de prendre en compte la redondance dans le calcul. Le tableau 3.4 présente le modèle proposé par les auteurs.

Tableau 3.4 Rapprochements COSMIC / UML selon Habela *et al.* (2004)

CONCEPT COSMIC	CONCEPT UML
Processus fonctionnel	Cas d'utilisation

3.2.6 Utiliser les modèles UML pour la mesure en prenant en considération les manipulations de données (Levesque *et al.*)

Levesque *et al.* dans (Levesque *et al.*, 2008) suggèrent l'utilisation des diagrammes de cas d'utilisation et de séquence pour les rapprochements COSMIC/UML. Particulièrement, ils ont souligné le fait de considérer les manipulations de données lors de la mesure en plus des mouvements de données. Cela dit, les mouvements de données sont alors mappés aux messages du diagramme de séquence et les manipulations de données sont rapprochées aux conditions liées aux messages d'erreur. Par conséquent, il devient simple de calculer la taille fonctionnelle en additionnant le nombre de messages dans le diagramme de séquence.

Toutefois, la procédure proposée par Levesque *et al.* (Levesque *et al.,* 2008) n'a pas été supportée par un outil pour la supporter, mais il reste intéressant de penser à intégrer les manipulations de données dans le calcul de la taille fonctionnelle avec COSMIC. Le tableau ci-dessous résume les rapprochements proposés par les auteurs.

Tableau 3.5 Rapprochements COSMIC / UML selon Levesque *et al.* (2008)

CONCEPT COSMIC	CONCEPT UML
Processus fonctionnel	Cas d'utilisation /Diagramme de séquence correspondant à chaque cas d'utilisation
Mouvement de données	Message
Manipulation données	Conditions liées aux messages d'erreur
Utilisateur Fonctionnel	Acteur du cas d'utilisation

3.2.7 Mesure de la taille fonctionnelle des diagrammes de cas d'utilisation (Sellami *et al.*)

Sellami et al. (Sellami *et al.*, 2009) ont choisi de mesurer la taille fonctionnelle des diagrammes de cas d'utilisation. Selon eux, les diagrammes de cas d'utilisation reflètent bien les processus fonctionnels puisqu'ils décrivent le logiciel tel qu'il est perçu par ses utilisateurs.

La technique utilisée dans ce travail consiste à utiliser un format spécifique pour décrire les diagrammes de cas d'utilisation. Chaque cas d'utilisation est en effet composé de un ou plusieurs scénarios. La somme des tailles fonctionnelles des scénarios reflète la taille fonctionnelle du cas d'utilisation qu'il les compose. La procédure de mesure prend en compte un nombre de règles qui considèrent les relations d'extension, d'inclusion et d'héritage entre les cas d'utilisation. La figure ci-dessous représente le format utilisé pour la documentation des cas d'utilisation.

```
Use case < name >
Actors: < Actor1,…>
Pre-condition: < condition1,…>
Extension point: < condition, use case uj>
NS /* Nominal scenario */
Begin
< Num > [<Pre-condition>] <Actor|System ><Action >
< Num > [<Pre-condition>] <Actor|System ><Action >
End
AS /* Alternative scenario */
Begin < Event, begin at Num >
< Num > [<Pre-condition>] <Actor|System ><Action >
< Num > [<Pre-condition>] <Actor|System ><Action >
The nominal scenario back to NUM
End
ES /* Error scenario */
Begin < Event, begin at Num k >
< Num > [<Pre-condition>] <Actor|System ><Action >
< Num > [<Pre-condition>] <Actor|System ><Action >
End
End use case
```

Figure 3.2 Format de documentation des cas d'utilisation tirée de Sellami *et al. (*2009)

Par la suite, les auteurs ont proposé d'appliquer le modèle de mesure de la taille fonctionnelle des cas d'utilisation pour mesurer la taille fonctionnelle des diagrammes de classe et de séquence. Enfin l'approche a été vérifiée en utilisant une étude de cas d'une application d'affaire.

La proposition de Sellami *et al.* (2009) semble être intéressante puisqu'elle vise à unifier la façon de calculer la taille fonctionnelle des différents diagrammes UML, contrairement à d'autres travaux qui ont essayé de mesurer la taille fonctionnelle d'un logiciel à partir de ses spécifications basées sur les diagrammes UML mais qui traitent ses diagrammes séparément.

Toutefois, le processus de mesure proposé demande encore du travail pour qu'il soit plus générique, par exemple l'appliquer sur une application de type temps réel.

3.2.8 Un espace d'exigences pour montrer comment UML peut supporter la mesure de la taille fonctionnelle (Van den Berg *et al.*)

Selon Van den Berg *et al.* (Van Den Burg *et al.*, 2005), UML peut être bien adapté pour spécifier les fonctionnalités utilisateurs requises. Alors, ils ont eu l'idée de proposer un modèle basé sur les exigences qui montre bien qu'UML peut supporter les FUR à différents niveaux de raffinement. Pour chaque niveau, un nombre de diagramme UML est défini dans le modèle d'exigences.

À partir de ce modèle, les auteurs ont établi un ensemble de rapprochements entre les concepts UML et les concepts de COSMIC. Ils affirment alors qu'à partir du diagramme de cas d'utilisation, on peut déduire les utilisateurs fonctionnels et la frontière du logiciel. Quant au diagramme de classe, il peut être utilisé pour mesurer les groupes de données et les attributs de données. Enfin, les auteurs ont proposé d'utiliser le diagramme d'activités pour représenter les processus fonctionnels et rapprochent celui-là avec le concept de flux d'événements.

Une étude de cas (Hotel case) a été proposée pour supporter l'approche décrite dans le travail. Le diagramme d'activités a été utilisé alors pour établir la mesure. En effet, les utilisateurs et la frontière sont déduits du diagramme de cas d'utilisation et, pour chaque cas, on identifie les différents mouvements de données pour déduire la taille fonctionnelle qui consiste à additionner les différents mouvements. Toutefois, l'inconvénient du travail de Van den Burg *et al.* (2005) est que le raisonnement derrière le choix de l'utilisation du diagramme d'activité n'est pas assez clair. Le tableau 3.6 résume l'approche proposée.

Tableau 3.6 Rapprochements COSMIC / UML selon Van den Berg *et al.* (2005)

Concept COSMIC	Concept UML
Processus fonctionnel	Diagramme d'activité correspondant à chaque cas d'utilisation
Mouvement de données	Cas d'utilisation
Utilisateur Fonctionnel	Acteur du cas d'utilisation
Frontière du logiciel	Diagramme de cas d'utilisation
Groupe de données	Diagramme de classe
Attributs de données	Diagramme de classe

3.2.9 Utilisation d'UML pour supporter la mesure de la taille fonctionnelle avec COSMIC (Lavazza *et al.*)

Le travail proposé par Lavazza *et al.* dans (Lavazza *et al.*, 2009) est l'un des travaux les plus récents qui ont étudié la problématique de supporter la mesure de la taille fonctionnelle avec COSMIC à l'aide des diagrammes UML. L'approche a été résumée à l'aide de l'étude de cas de COSMIC « The rice cooker controller » qui est une application de type temps réel. Ils se sont basés alors sur un mapping entre les concepts de COSMIC et UML.

Les diagrammes UML utilisés par ces chercheurs sont les diagrammes de cas d'utilisation, de composant, de classe et de séquence. Les utilisateurs fonctionnels et la frontière du logiciel peuvent être déduits à partir du diagramme de cas d'utilisation ou le diagramme de composant. Quant au concept d'événement déclencheur, on peut le déduire à partir du

diagramme de composant. Selon ces auteurs, le diagramme de séquence est bien placé pour représenter les concepts de processus fonctionnels et de mouvements de données.

Finalement, les groupes de données peuvent être retranchés du diagramme de composant ou de classe. Une autre problématique proposée par les auteurs dans ce travail est d'utiliser les diagrammes de cas d'utilisation et de composant pour identifier les utilisateurs fonctionnels de l'application à mesurer.

En totalité, l'approche proposée dans ce travail est très intéressante car elle couvre bien tous les concepts COSMIC; de plus, elle peut aider les concepteurs d'une part à construire des modèles UML orientés mesure et les mesureurs, d'autre part à appliquer les règles de mesure d'une façon plus simple (Lavazza *et al.*, 2009). Le tableau 3.7 résume les mapping UML/COSMIC établis dans ce travail.

Tableau 3.7 Mapping COSMIC / UML selon Lavazza *et al.* (2009)

Concept COSMIC	Concept UML
Processus fonctionnel	Cas d'utilisation / Interaction dans le diagramme de séquence
Mouvement de données	Message
Utilisateur Fonctionnel	Acteur du cas d'utilisation / composant externe dans le diagramme de composant
Frontière du logiciel	Diagramme de cas d'utilisation
Groupe de données	Diagramme de classe/ Diagramme de composant
Événement déclencheur	Élément dans le diagramme de composant

3.3 Comparaison des travaux de recherche

La revue de littérature présentée dans ce chapitre comporte neuf travaux qui discutent l'idée d'utiliser UML pour la mesure de la taille fonctionnelle avec COSMIC.

Nous avons constaté que la plupart des propositions se sont basées sur les diagrammes de cas d'utilisation et de séquence puisque ce sont les deux diagrammes les plus utilisés pour spécifier les exigences du système.

Cela n'empêche pas que d'autres auteurs ont choisi d'utiliser d'autres diagrammes UML tel que les diagrammes de classe, de composant et d'activité pour faires des rapprochements UML / COSMIC. Dans cette section nous présentons un résumé de ces propositions - voir tableau 3.9. Nous avons utilisé les abréviations pour les concepts UML pour une contrainte d'espace dans le tableau 3.9. Les significations des différentes abréviations sont présentés au tableau 3.8:

Tableau 3.8 Tableau des abréviations des concepts COSMIC

ABRÉVIATION	CONCEPT
ED	Evénement Déclencheur
PF	Processus Fonctionnel
MD	Mouvement de Données
GD	Groupe de Données
AD	Attribut de Données
FR	Frontière
UF	Utilisateur Fonctionnel
MPD	Manipulation de Données

Tableau 3.9 Tableau récapitulatifs des propositions étudiées

Auteur / Année	Concepts COSMIC couverts	Diagrammes UML	Domaine d'application	Automatisation	Outil supportant l'automatisation
Bévo et al. / 2005	PF/MD/GD/ AD/ FR/UF	-Cas d'utilisation -Scénarios -Classe	Affaire	Oui	Metric Xpert
Jenner / 2001	PF/MD	-Cas d'utilisation -Séquence	Affaire	Oui	Un prototype a été proposé pour faciliter les estimations
Azzouz et al. / 2004	ED/PF/MD/G D/ AD/ FR/UF	-Cas d'utilisation -Scénarios -Classe Ils ont utilisé RUP et Rational Rose en particulier	Temps réel	Oui	Prototype de RUP-COSMIC FFP
Luckson et al. / 2004	PF/MD/GD/ AD/ FR/UF	-Méta-modèle MOF	Affaire et temps réel	Oui	Mertic Xpert
Habela et al. / 2005	PF/MD	-Cas d'utilisation	Affaire	Non	Aucun
Van den Berg et al. / 2005	PF/MD/MPD/ UF	-Cas d'utilisation -Activité -Classe	Affaire	Non	Aucun
Sellami et al. / 2009		-Cas d'utilisation -Séquence -Classe	Affaire	Non	Aucun
Levesque et al. / 2008	PF/MD/UF/ FR/GD/AD	-Cas d'utilisation -Séquence	Affaire	Non	Aucun
Lavazza et al. / 2009	PF/MD/UF/ FR/GD/ED	-Cas d'utilisation -Séquence -Composant -Class	Temps réel	Non	Aucun

3.4 Autres travaux reliés

3.4.1 Un méta-modèle pour la méthode COSMIC proposé par Condori-Fernandèz *et al.* (2007)

Dans leur travail, Fernandez *et al.* (Condori-Fernandez *et al.*, 2007) introduisent une procédure de mesure appelée **RmFFP** pour modéliser et estimer la taille fonctionnelle des systèmes orientés objet à partir des spécifications de haut niveau en utilisant le modèle d'exigences pour les méthodes orientées objets. Leur but est de quantifier la taille fonctionnelle à un stade précoce du cycle de vie du logiciel.

Les auteurs ont utilisé le processus de production d'un logiciel en utilisant les méthodes orientées objet. Le processus est défini à la figure 3.3:

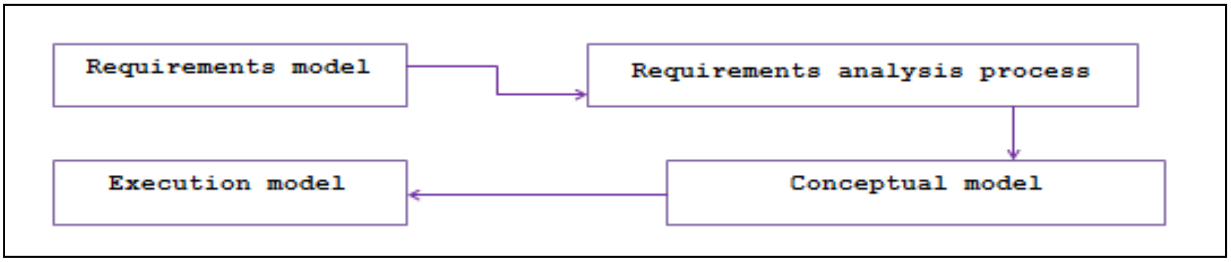


Figure 3.3 Processus de production d'un logiciel en utilisant les méthodes orientées objet

Les auteurs ont procédé au design de leur processus **RmFFP** comme suit :

- Étape 1 : Définition des objectifs
- Étape 2 : Caractérisation des concepts à mesurer
- Étape 3 : Sélection du méta-modèle :

- Identifier les concepts du modèle de domaine et leurs relations afin de modéliser le logiciel selon COSMIC.
- Utiliser UML pour représenter le méta-modèle (plus particulièrement sous forme de diagramme de classe).

- Le design du méta-modèle COSMIC a été basé sur le manuel de mesure de COSMIC (version 2.2).

Le but derrière le design du méta-modèle est d'effectuer un mapping entre les concepts du méta-modèle COSMIC et les concepts du modèle d'exigences d'une méthode orientée objet. Dans la figure 3.4, nous reportons le méta-modèle tel que défini dans (Condori-Fernandéz *et al.*, 2007).

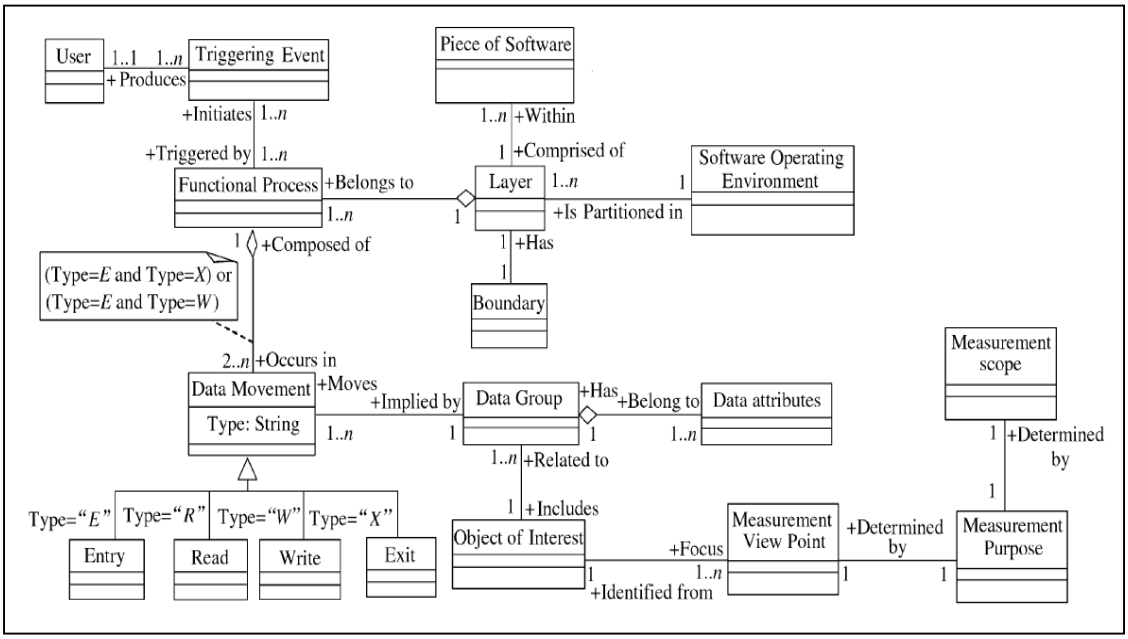


Figure 3.4 Méta-modèle COSMIC définit par Condori-Fernandèz *et al.* (2007)

Voici au tableau 3.10 quelques règles de mapping définies dans ce travail.

Tableau 3.10 Tableau récapitulatifs des propositions étudiées

CONCEPT COSMIC	CONCEPT UML
Groupe de données	Classe / Diagramme de classe
Frontière du logiciel	Frontière entre les cas d'utilisation et les acteurs
Utilisateur Fonctionnel	Acteur / Diagramme de cas d'utilisation

Pour conclure, nous pouvons déduire que le méta-modèle proposé par Condori-Fernandèz *et al.* (2007) est conçu pour deux raisons :

1- Formaliser les concepts de COSMIC en se basant sur le manuel de mesure. (Formaliser ici veut dire classer les éléments de mesure d'une manière formelle et claire).
2- Produire les règles sur lesquelles s'est basée la procédure de mesure **`RmFFP`**.

Remarque : Nous pouvons nous baser sur le méta-modèle de Fernandèz *et al.* pour bâtir notre méta-modèle COSMIC mais en effectuant des changements (une justification des changements sera présentée un peu plus loin).

3.4.2 Estimation automatisée de la taille des composants logiciels embarqués proposée par Lind.K et al (Lind.K et Rogardt.H, 2011)

Dans le contexte des systèmes embarqués, Kenneth *et al.* (Lind.K et Rogardt.H, 2011) s'intéressent à l'estimation de la taille du code à l'aide de la méthode COSMIC. Leur but est de minimiser l'effort manuel pour faire les estimations de la taille du code. Pour cela, ils ont choisi de concevoir un profil UML qui prend en compte l'information nécessaire pour faire la mesure (Lind.K et Rogardt.H, 2011).

Leur travail a été principalement basé sur les composants UML. Selon eux, on peut facilement faire ressortir les concepts de frontière et de mouvement de données de type «Entrée » et « Sortie » à partir des diagrammes de composant. Pour les deux types de

mouvement de données « Lire » et « Écrire », ils ont proposé un nouveau stéréotype nommé CompSizeProperty qui étend la classe Property qui capture cette information. D'autres concepts liés à la méthode COSMIC comme le niveau de granularité et le niveau de décomposition ont été capturés aussi à l'aide d'un nouveau stéréotype nommé CompSizeComponent qui étend la classe component. Ces concepts constituent des valeurs marquées pour ce stéréotype. Le but principal des auteurs à travers ce travail est de modéliser le modèle générique de COSMIC à l'aide d'UML en utilisant un profil UML basé sur les composants.

Dans un autre travail (Lind.K et Rogardt.H, 2011), les auteurs ont développé un outil qui permet d'importer l'information modélisée avec le profil UML conçu. L'outil permet aussi de stocker cette information pour faire l'estimation de la taille du code à l'aide de la régression linéaire et présenter enfin les résultats de ces estimations.

3.5 Conclusion

Nous avons reporté dans ce chapitre les principales propositions qui ont comme problématique de formaliser les procédures liées à la méthode COSMIC à l'aide d'UML puisque c'est la notation la plus utilisée pour concevoir et modéliser les systèmes logiciels.

CHAPITRE 4

PROFIL UML POUR LA METHODE COSMIC

4.1 Introduction

Dans ce chapitre nous introduisons notre approche. Après avoir étudié les différents travaux reliés à la mesure de la taille fonctionnelle à partir des spécifications écrites à l'aide d'UML, nous proposons un profil UML permettant la modélisation des procédures liées à COSMIC. Nous nous concentrons d'abord sur la conception du méta-modèle qui décrit le domaine COSMIC en se basant sur le manuel de mesure. Par la suite, nous présentons l'architecture globale du profil à concevoir et les règles du modèle du domaine. Nous justifions ensuite le choix technologique pour l'implémentation du profil. Nous terminons le chapitre en exposant les résultats de l'implémentation du profil P-COSMIC.

4.2 Modèle du domaine : Métamodèle COSMIC

La première étape de la définition d'un profil UML consiste à concevoir le modèle du domaine ou plus spécifiquement le méta-modèle (Selic, 2007). Ceci se fait en considérant les étapes du processus de mesure. Les trois phases de mesure avec la méthode COSMIC sont :

a. La phase de stratégie de mesure
b. La phase d'arrimage, et
c. La phase de mesure

Cependant avant de présenter notre méta-modèle, nous allons présenter le méta-modèle déjà proposé par Condori-Fernandèz *et al.* dans (Condori-Fernandez *et al.*, 2007). La figure 4.1 décrit ce méta-modèle.

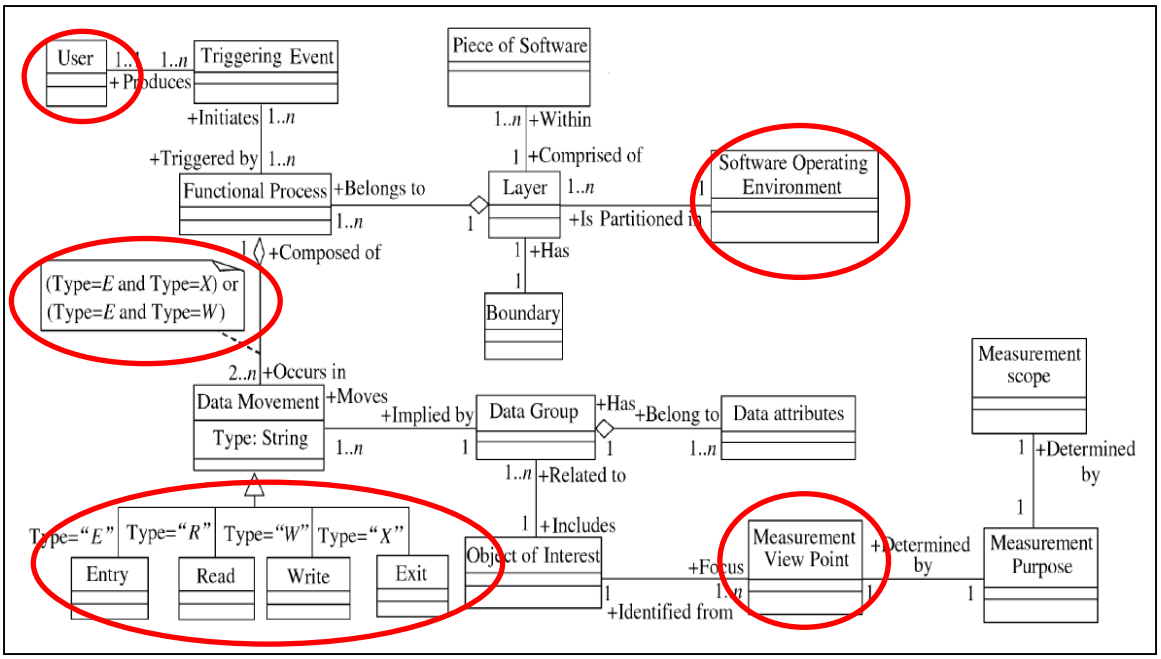


Figure 4.1 Méta-Modèle COSMIC tiré de Condori-Fernandèz *et al.* (2007)

Il est à noter que ce méta-modèle a été conçu pour regrouper les concepts liés au domaine COSMIC afin d'établir des règles sur lesquelles se base le modèle de mesure proposé par Condori-Fernandèz *et al.* (2007).

Comme le montre la figure 4.1, quelques changements seront effectués sur ce méta-modèle, les concepts encerclés en rouge seront soit modifiés ou éliminés dans notre méta-modèle.

- Le concept de Mesurement View Point sera supprimé : le concept de Mesurement Purpose inclut déjà le point de vue de la mesure, donc il devient inutile de créer une classe Mesurement View Point.

- Le concept Software Operating Environment sera supprimé : ce terme a été identifié pour expliquer que la frontière d'un logiciel est une interface conceptuelle qui permet au mesureur de distinguer ce qui fait partie de son logiciel et ce qui fait partie de son environnement. Selon mon point de vue, ce concept n'ajoute aucune valeur à la mesure.

- Le concept de « user » sera changé pour devenir Functional User : dans le manuel de mesure de COSMIC version 3.0, le concept de utilisateur a été changé par utilisateur fonctionnel. Rappelons qu'un utilisateur fonctionnel peut être un utilisateur humain, un logiciel qui interagit avec le logiciel à mesurer ou un tout autre dispositif.

- Les quatre classes Entry, Read, Write et Exit ne seront pas obligatoires : la classe Data Movement a déjà un attribut type qui aura une de ces quatre valeurs : E,R,W ou X

Suite à ces changements, notre modèle de domaine pour la méthode COSMIC est présenté dans la figure 4.2.

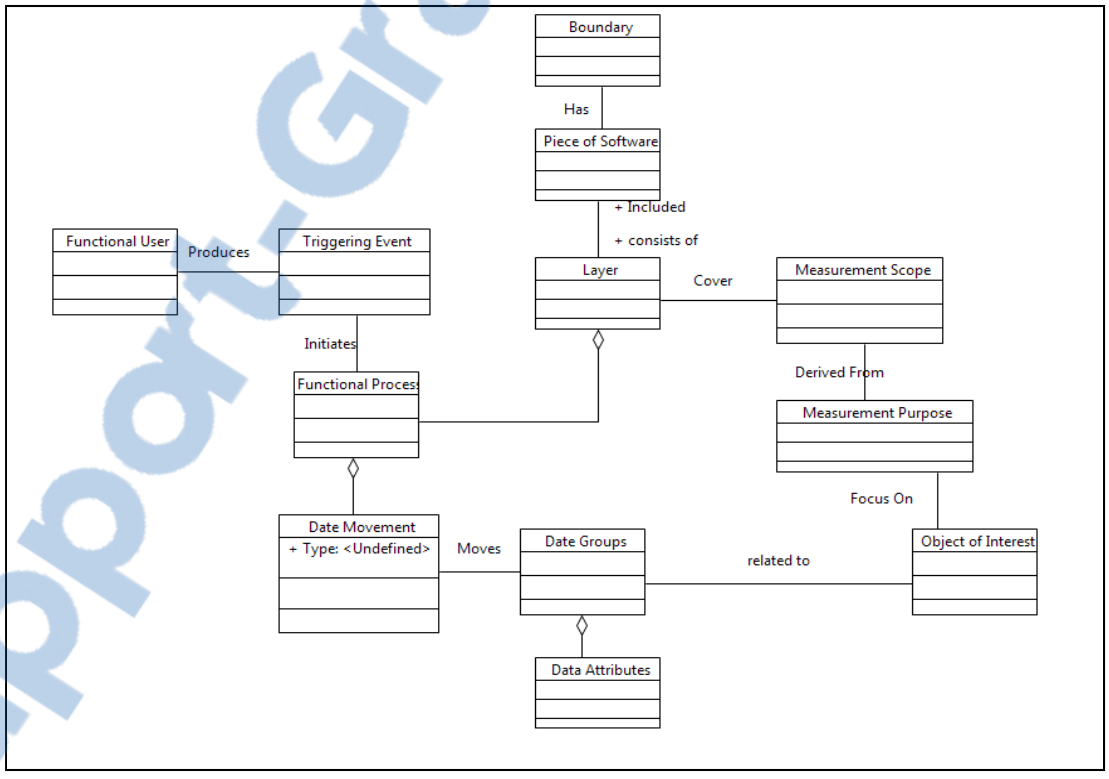


Figure 4.2 Premier prototype du méta-modèle COSMIC

La phase de stratégie de la mesure consiste à identifier la raison pour laquelle on mesure, le périmètre de la mesure et les utilisateurs fonctionnels. En plus de ça, il est important de définir le niveau de granularité auquel la mesure doit être effectuée. D'autres paramètres sont aussi à considérer pendant cette phase. Ces paramètres figureront dans notre méta-modèle et sont les suivants :

- ⇨ Raison d'être de la mesure : Pourquoi on mesure?
- ⇨ Périmètre de la mesure : Ensemble des FUR.
- ⇨ Morceau logiciel : Élément sur lequel s'applique la mesure.
- ⇨ Composant logiciel : Résultat de la division d'un morceau logiciel.
- ⇨ Niveau de décomposition : Niveau résultant de la division du morceau logiciel en composants.
- ⇨ Couche : Couche logicielle dans laquelle se situe le logiciel à mesurer.
- ⇨ Utilisateurs fonctionnels : Font partie des utilisateurs potentiels du logiciel à mesurer pour lesquels la fonctionnalité est offerte.
- ⇨ Niveau de granularité : Résultat du raffinement des FUR de haut niveau à un niveau plus détaillé. A chaque fois la fonctionnalité est décrite de manière plus profonde, on a un nouveau niveau de granularité.

Nous avons choisi de rassembler les deux paramètres « raison d'être » et « périmètre » dans un seul paramètre intitulé « contexte de mesure » qui aura comme attributs ces deux concepts.

Lors de la phase d'arrimage, il est essentiel d'identifier les différents processus fonctionnels initialisés par des événements déclencheurs qu'on doit identifier aussi. De plus, il faut identifier la liste des objets d'intérêt et les groupes de données qui les décrivent. Tous ces paramètres s'ajouteront aussi à notre modèle de domaine. Voici le résumé de ces paramètres :

- Processus fonctionnels : Une partie des FUR du logiciel à mesurer, autrement dit, les FUR sont composées de un ou plusieurs processus fonctionnels.

- Événements déclencheurs : Un événement perçu par un utilisateur fonctionnel pour déclencher un processus fonctionnel.
- Objets d'intérêt : Éléments liés aux utilisateurs fonctionnels pour lesquels le logiciel traite les fonctionnalités.
- Groupe de données : L'ensemble des données qui décrivent un objet d'intérêt.
- Attributs de données : Les éléments qui composent un groupe de données.

Finalement, afin de procéder à la mesure, il est incontournable d'identifier les différents mouvements de données et leurs types pour pouvoir appliquer les fonctions de mesure.

- Mouvement de données : Paramètre très important pour la phase de mesure, constitue un élément fonctionnel qui délace un ensemble de groupes de données.

Le Méta-Modèle pour COSMIC doit alors refléter les paramètres principaux de la méthode qui contribuent au calcul de la taille fonctionnelle du logiciel. Ce méta-modèle étend alors le Méta-Modèle UML en ajoutant les nouveaux éléments de modélisation reliés à COSMIC. Ces concepts sont en résumé : *Utilisateur Fonctionnel, Événement Déclencheur, Morceau Logiciel, Composant Logiciel, Couche, Objet d'intérêt, Contexte de mesure, Processus Fonctionnel, Mouvement de Donnée, Groupe de donnée et Attribut de Donnée.*

Une vue globale du méta-modèle proposé est illustrée dans la figure 4.3.

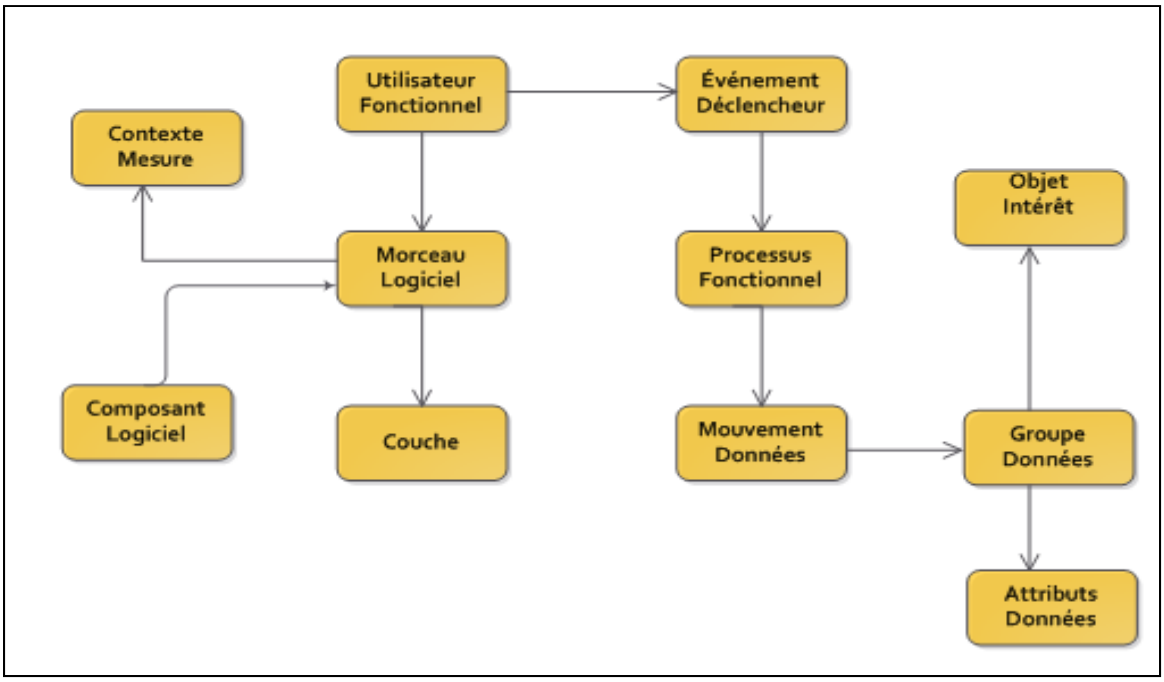


Figure 4.3 Vue globale du méta-modèle proposé pour COSMIC

Comme décrit dans la figure 4.3, les relations entre les paramètres de la méthode COMIC sont les suivantes :

- Un utilisateur fonctionnel utilise le morceau logiciel à mesurer et déclenche un événement déclencheur.
- Le morceau logiciel peut être conçu à base de composants logiciel et il appartient à une couche. Un contexte de mesure est alors défini pour le morceau logiciel ou pour chaque composant.
- Un événement déclencheur initialise un processus fonctionnel.
- Un processus fonctionnel est composé de mouvements de données.
- Les mouvements de données déplacent les groupes de donnés qui sont composés d'attributs de données.
- Les groupes de données sont liés à des objets d'intérêt.

La figure 4.4 illustre le méta-modèle proposé pour la méthode COSMIC.

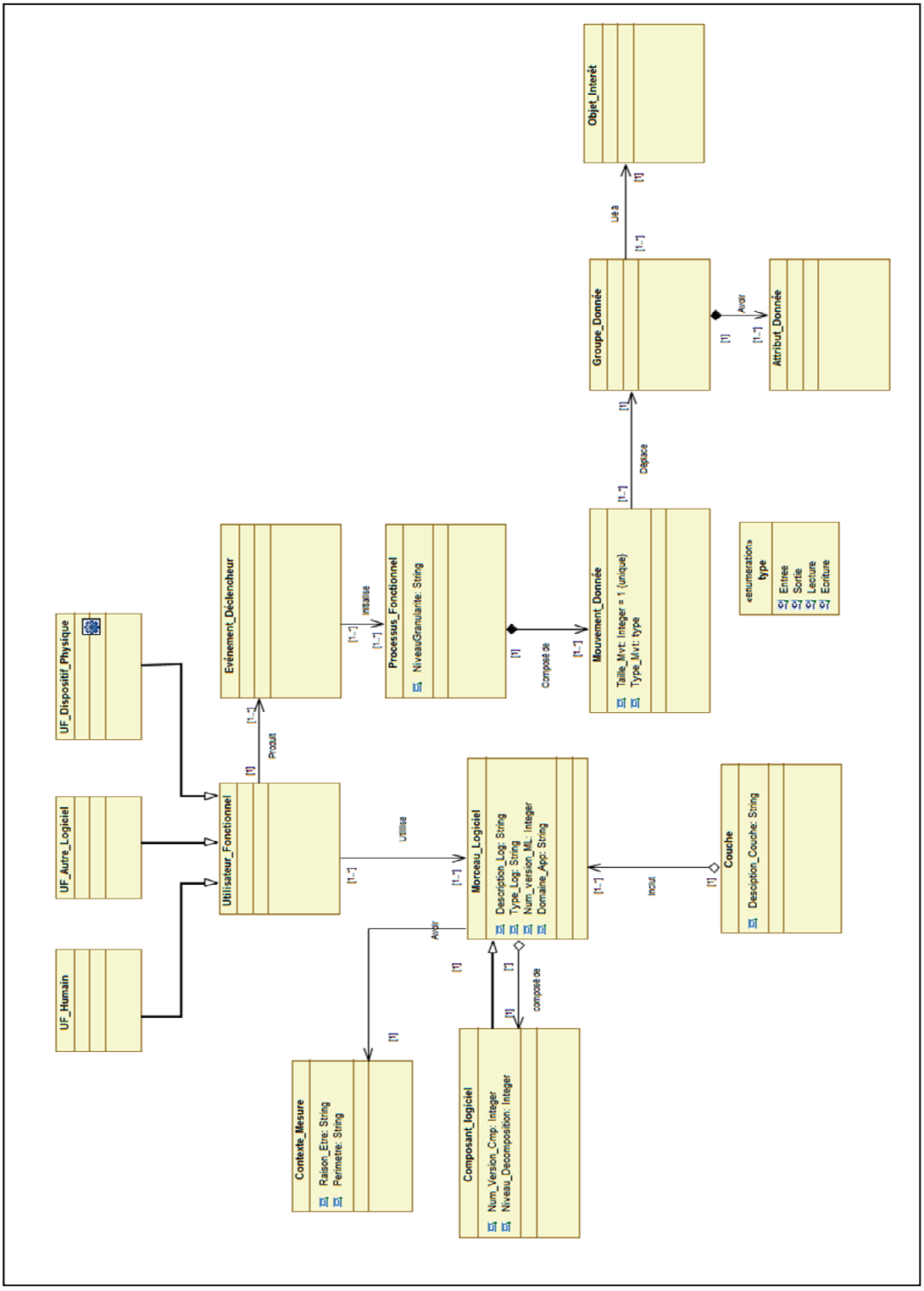


Figure 4.4 Méta-modèle COSMIC proposé

4.3 Règles liées au domaine

Il est très important de spécifier les règles du domaine pour que le méta-modèle soit valide. Afin de respecter les règles de la méthode COSMIC, nous avons étudié toutes les règles liées aux paramètres de la méthode. Nous avons alors sélectionné les règles qui s'appliquent à notre méta-modèle. La liste est la suivante :

- ✓ R1: Un morceau logiciel appartient à une seule couche.
- ✓ R2: Un morceau logiciel peut être conçu à la base de composants logiciels.
- ✓ R3: Un utilisateur fonctionnel peut être un être humain, un dispositif physique ou un autre logiciel qui interagit avec le logiciel à mesurer.
- ✓ R4: Un mouvement de données ne déplace qu'un seul groupe de données.
- ✓ R5: Un groupe de données décrit un et un seul objet d'intérêt.
- ✓ R6: Un processus fonctionnel est complété par au moins 2 mouvements de données tel que : un mouvement de données en entrée + un mouvement de données en sortie et/ou un mouvement de données en écriture.
- ✓ R7: Chaque groupe de donnée identifié doit être unique pour l'ensemble des attributs des données qu'il contient.
- ✓ R8: Un événement déclencheur peut déclencher 1 ou plusieurs processus fonctionnels.
- ✓ R9: Si un événement déclencheur est introduit alors un processus fonctionnel est produit.
- ✓ R10: Le type de mouvement de donnée ne peut être autre que : Entrée, Sortie, Lecture ou Écriture.
- ✓ R11: Le périmètre de mesure ne doit pas couvrir plus d'une couche logicielle.
- ✓ R12: La taille d'un mouvement de données est de 1 CFP, c'est l'étalon de mesure COSMIC.
- ✓ R13: Si un morceau logiciel est conçu à la base de composants alors :
    - Il faut définir un périmètre unique pour chaque composant.
    - Calculer séparément la taille de chaque composant.

4.4 Architecture du profil P-COSMIC

Une fois le méta-modèle défini et les règles du domaine définies, on peut passer à l'étape suivante qui est la définition du profil COSMIC. Cela revient à définir les stéréotypes nécessaires avec leurs attributs (Tagged values) et les contraintes sous le format OCL éventuellement. Le profil UML pour COSMIC est basé essentiellement sur les règles de mesure telles que définies par le groupe COSMIC (Abran *et al.*, 2009).

Cependant, avant de commencer la définition ou la conception du profil UML, nous devons répondre à la question suivante : l'application du profil pour COSMIC est-elle pratique?

Selon mon point de vue, l'application d'un tel profil UML pour supporter la mesure de la taille fonctionnelle des logiciels en utilisant la méthode COSMIC reste utile et pratique. Il y'aura une standardisation de l'application de cette méthode de mesure. Cela veut dire qu'au lieu d'avoir de multiples choix de formalisation des éléments COSMIC (mapping entre éléments COSMIC/ éléments UML), avec notre framework (Profil COSMIC), on aura une seule et unique forme de mesure. Les résultats attendus du profil pour COSMIC sont:

- Mesure plus compatible avec les règles de mesure COSMIC.
- Application de la mesure standardisée avec COSMIC.
- Taille fonctionnelle facile à calculer.
-Faciliter la tâche au concepteur et au mesureur.

La figure 4.5 résume l'architecture globale de notre profil.

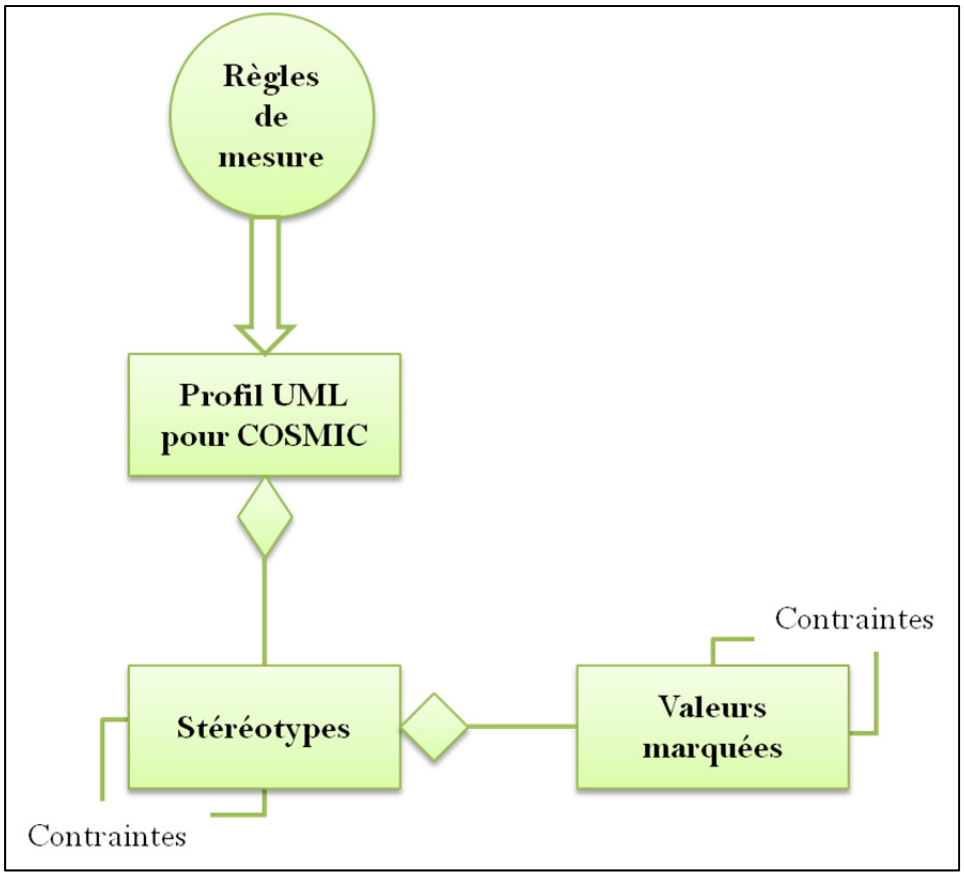


Figure 4.5 Architecture globale du profil UML pour COSMIC

4.5 Description du profil P-COSMIC

4.5.1 Mapping du méta-modèle vers UML : Liste des stéréotypes

La figure 4.6 définit le principe de mapping entre les concepts UML et les concepts du méta-modèle.

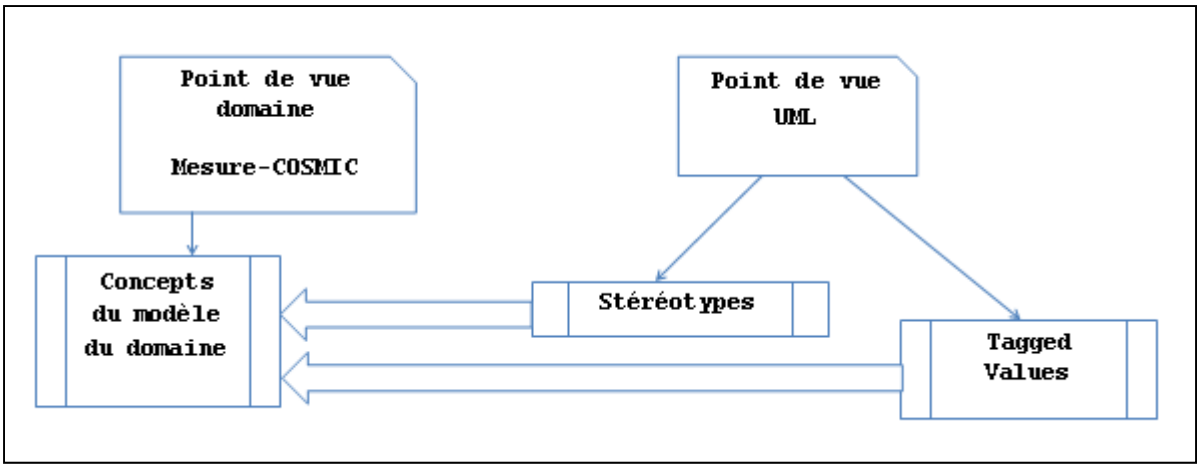


Figure 4.6 Principe de mappage des concepts du modèle de domaine aux concepts UML

Les tableaux 4.1 à 4.4.14 résument la liste des stéréotypes définis dans le profil UML pour COSMIC.

Tableau 4.1 Spécification du stéréotype Utilisateur_Fonctionnel

Méta-Classe/ Concept	**Utilisateur_Fonctionnel**	
Stéréotype	***Nom***	***Icône***
	«FunctionalUser»	Pas d'icône
Valeurs Marquées	Aucune	
Classe(s) étendue(s)	Actor : en modélisant le diagramme de cas d'utilisation Lifeline: en modélisant le diagramme de séquence	
Description	-Décrit un utilisateur fonctionnel. -Il peut être soit un humain, un dispositif physique ou un autre logiciel qui interagit avec le logiciel à mesurer.	

Tableau 4.2 Spécification du stéréotype UF-Humain

Méta-Classe/ Concept	**UF_Humain**	
Stéréotype	***Nom***	***Icône***
	«FU-Humain»	
Valeurs Marquées	Aucune	
Classe(s) étendue(s)	Actor : en modélisant le diagramme de cas d'utilisation Lifeline: en modélisant le diagramme de séquence	
Description	-Décrit un utilisateur fonctionnel de type humain.	

Tableau 4.3 Spécification du stéréotype UF-Autre-Logiciel

Méta-Classe/ Concept	**UF_Autre_Logiciel**	
Stéréotype	***Nom***	***Icône***
	«FU-OtherSoft»	Pas d'icône
Valeurs Marquées	Aucune	
Classe(s) étendue(s)	Actor : en modélisant le diagramme de cas d'utilisation Lifeline: en modélisant le diagramme de séquence	
Description	-Décrit un utilisateur fonctionnel de type autre morceau logiciel ou autre composant logiciel.	
Contrainte(s)	Aucune	

Tableau 4.4 Spécification du stéréotype UF-Dispositif Physique

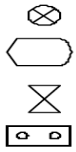
Méta-Classe/ Concept	**UF_Dispositif_Physique**	
Stéréotype	***Nom***	***Icône***
	«FU-EngDevice»	
Valeurs Marquées	Aucune	
Classe(s) étendue(s)	Actor : en modélisant le diagramme de cas d'utilisation Lifeline: en modélisant le diagramme de séquence	
Description	-Décrit un utilisateur fonctionnel de type dispositif physique.	

Tableau 4.5 Spécification du stéréotype Événement_Déclencheur

Méta-Classe/ Concept	**Événement_Déclencheur**	
Stéréotype	***Nom***	***Icône***
	«TriggEvent»	Pas d'icône
Valeurs Marquées	Aucune	
Classe(s) étendue(s)	Message	
Description	-Décrit un événement Déclencheur produit par un utilisateur fonctionnel.	

Tableau 4.6 Spécification du stéréotype Morceau_Logiciel

Méta-Classe/ Concept	**Morceau_Logiciel**	
Stéréotype	***Nom***	***Icône***
	«PieceOfSoft»	Pas d'icône
Valeurs Marquées	- SoftDescription -SoftType -NVersion -AppDomain	
Classe(s) étendue(s)	Package : en modélisant le diagramme de cas d'utilisation Lifeline: en modélisant le diagramme de séquence	
Description	-Décrit le morceau logiciel à mesurer.	

Tableau 4.7 Spécification du stéréotype Couche

Méta-Classe/ Concept	**Couche**	
Stéréotype	***Nom***	***Icône***
	«layer»	Pas d'icône
Valeurs Marquées	-LayerDescription	
Classe(s) étendue(s)	Package	
Description	-Décrit une couche spécifique à laquelle appartient le logiciel à mesurer.	

Tableau 4.8 Spécification du stéréotype Processus_Fonctionnel

Méta-Classe/ Concept	**Processus_Fonctionnel**	
Stéréotype	***Nom***	***Icône***
	«FunctProcess»	Pas d'icône
Valeurs Marquées	-GranularityLevel	
Classe(s) étendue(s)	Interaction	
Description	-Décrit un processus fonctionnel qu'est un composant élémentaire de l'ensemble des fonctionnalités utilisateur requises.	

Tableau 4.9 Spécification du stéréotype Mouvement_Donnée

Méta-Classe/ Concept	**Mouvement_Donnée**	
Stéréotype	***Nom***	***Icône***
	«DataMvt»	Pas d'icône
Valeurs Marquées	- Size - Type	
Classe(s) étendue(s)	Message	
Description	-Décrit un mouvement de données appartenant à un processus fonctionnel.	

Tableau 4.10 Spécification du stéréotype Groupe_Donné

Méta-Classe/ Concept	**Groupe_Donnée**	
Stéréotype	***Nom***	***Icône***
	«DataGr»	
Valeurs Marquées	Aucune	
Classe(s) étendue(s)	Class	
Description	-Décrit un groupe de données qui est un ensemble non vide, non ordonné et non redondant d'«Attribut_Donnée».	

Tableau 4.11 Spécification du stéréotype Attribut_Donné

Méta-Classe/ Concept	**Attribut_Donnée**	
Stéréotype	***Nom***	***Icône***
	«DataAtt»	
Valeurs Marquées	Aucune	
Classe(s) étendue(s)	Property	
Description	-Décrit chaque attribut de donnée composant un groupe de données lié aux FUR.	

Tableau 4.12 Spécification du stéréotype Objet_Intérêt

Méta-Classe/ Concept	**Objet_Intérêt**	
Stéréotype	***Nom***	***Icône***
	«ObjInterest»	
Valeurs Marquées	Aucune	
Classe(s) étendue(s)	Class	
Description	-Décrit l'objet d'intérêt identifié du point de vue des FUR.	

Tableau 4.13 Spécification du stéréotype Contexte_Mesure

Méta-Classe/ Concept	**Contexte_Mesure**	
Stéréotype	***Nom***	***Icône***
	«MeasContext »	Pas d'icône
Valeurs Marquées	-MeasPurpose -MeasScope	
Classe(s) étendue(s)	Comment	
Description	-Description textuelle de la raison d'être de la mesure et du périmètre de la mesure.	

Tableau 4.14 Spécification du stéréotype Composant_Logiciel

Méta-Classe/ Concept	**Composant_Logiciel**	
Stéréotype	***Nom***	***Icône***
	«SoftComp»	Pas d'icône
Valeurs Marquées	- DecompLevel	
Classe(s) étendue(s)	Package	
Description	-Décrit le composant logiciel à mesurer (Ce concept est inclus si la raison d'être de la mesure est de mesurer séparément chaque composant logiciel constituant le morceau logiciel)	

Nous pouvons constater que notre profil s'appliquerait plus adéquatement aux diagrammes de séquence UML. En effet, un processus fonctionnel est une liste de séquences composée de mouvements de données, ce qui ressemble à une interaction en UML.

4.6 Implémentation du profil P-COSMIC

4.6.1 Choix technologique

Pour pouvoir créer et générer un profil UML, il faudrait disposer d'un outil de modélisation basé sur le standard UML et qui permet également la création des profils. Suite à un sondage effectué dans (Bobkowska.A et Marek.W, 2008), il a été évalué que l'outil Visual Paradigm est l'un des outils de modélisation les plus utilisés par les programmeurs et les analystes. Dans le même travail, une étude de cas a montré que Visual Paradigm est bien adapté pour les analystes d'affaires, les développeurs et même pour les concepteurs de tests.
Voici un exemple de profil UML crée à l'aide de Visual Paradigm :

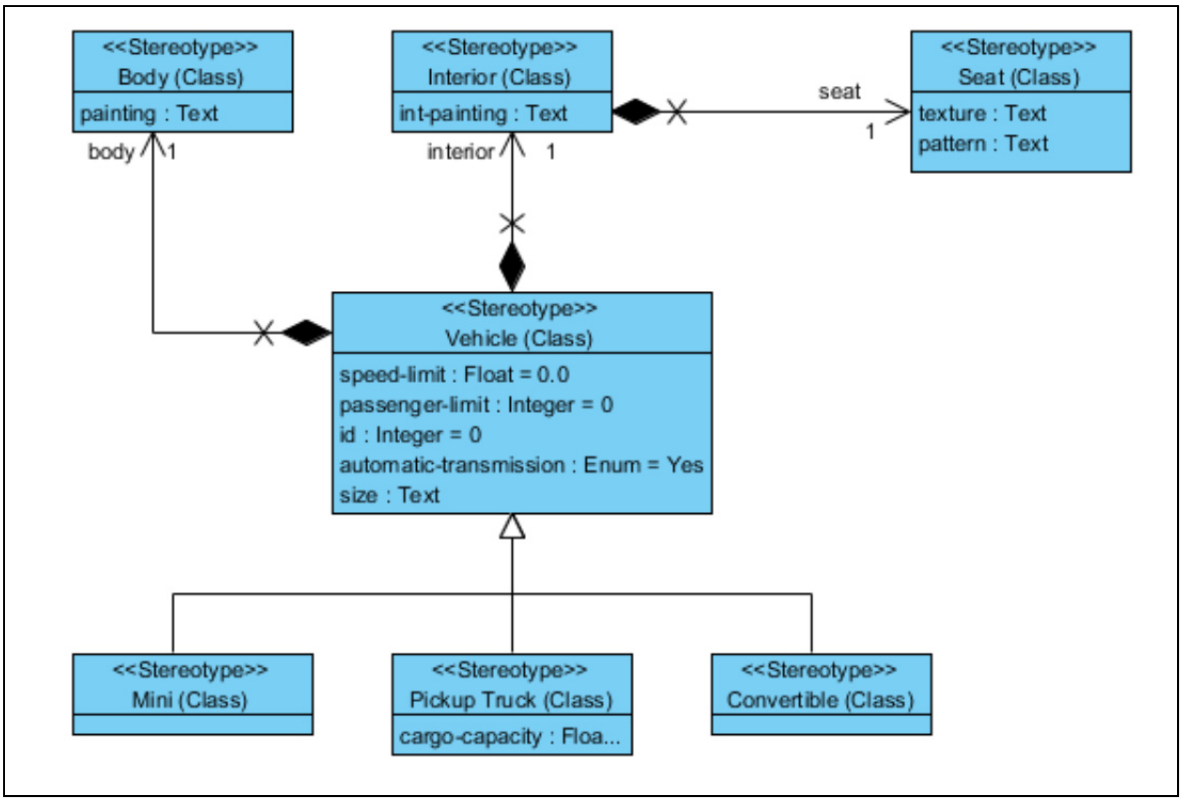


Figure 4.7 Exemple de profil UML tiré de Visual Paradigm (2011)

Certes, Visual Paradigm est un outil très populaire pour la modélisation avec UML, et répond au besoin de notre projet puisqu'il permet la gestion des profils UML. L'inconvénient que j'ai rencontré avec cet outil est qu'il n'est pas assez souple à manipuler ainsi qu'il souffre de manque de documentation au niveau de la gestion de profils. Ce qui m'a encouragé à voir d'autres outils qui pourront me servir mieux.

L'outil Papyrus (Papyrus, 2011) qui est un outil open source et basé sur la technologie Eclipse est l'un des outils les plus utilisés pour la modélisation des profils UML. Il est dédié à la modélisation UML2 et soutient également les langages SysML et MARTE. Le point qui m'a intéressé le plus avec cet outil, c'est qu'il contient un module spécifique à la gestion des profils UML (voir figure 4.8) et il est accompagné d'une bonne documentation spécifique à l'usage des profils UML. Il fournit en plus un assistant qui facilite la génération du code OCL pour la génération des contraintes et de les appliquer aux profils.

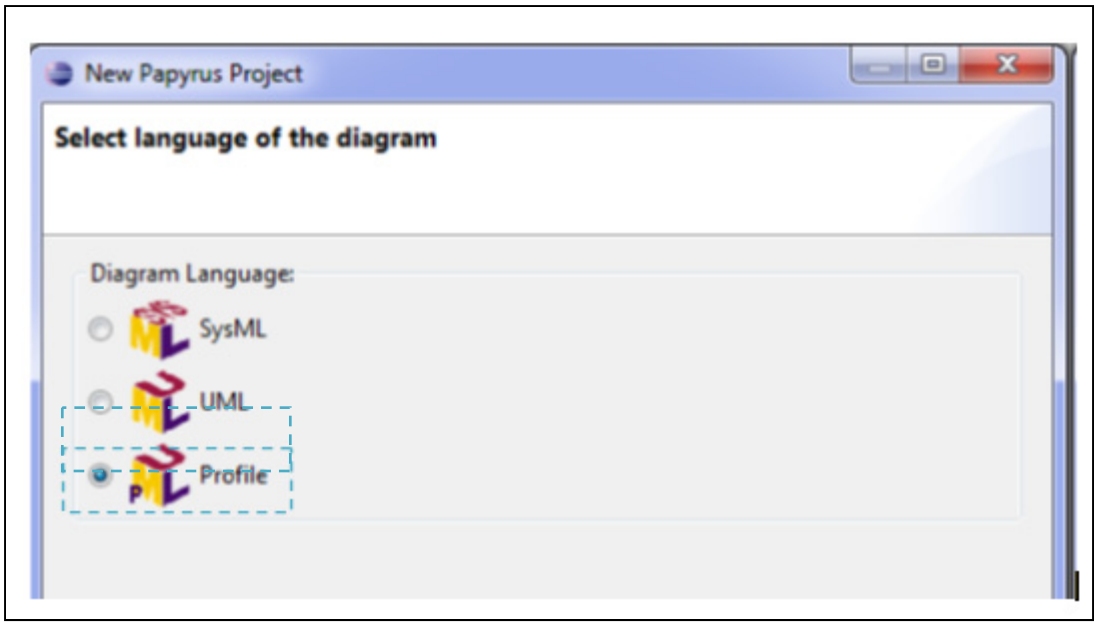


Figure 4.8 Modules de gestion des profils avec Papyrus

Un autre avantage de cet outil est qu'il peut être utilisé directement sous Eclipse ou aussi il peut être installé séparément en tant qu'une application autonome.

4.6.2 Résultats de l'implémentation

Dans cette section, nous allons décrire les étapes suivies pour la création du profil UML dédié pour COSMIC que nous avons nommé P-COSMIC.

La première étape consiste à mettre en place les stéréotypes définis précédemment dans la section 4.5.1. Nous pouvons voir dans la figure 4.9 les différents stéréotypes composant le profil avec l'étiquette « stereotype », les propriétés des classes du méta-modèle sont définies de même dans le profil mais en tant que tagged values.

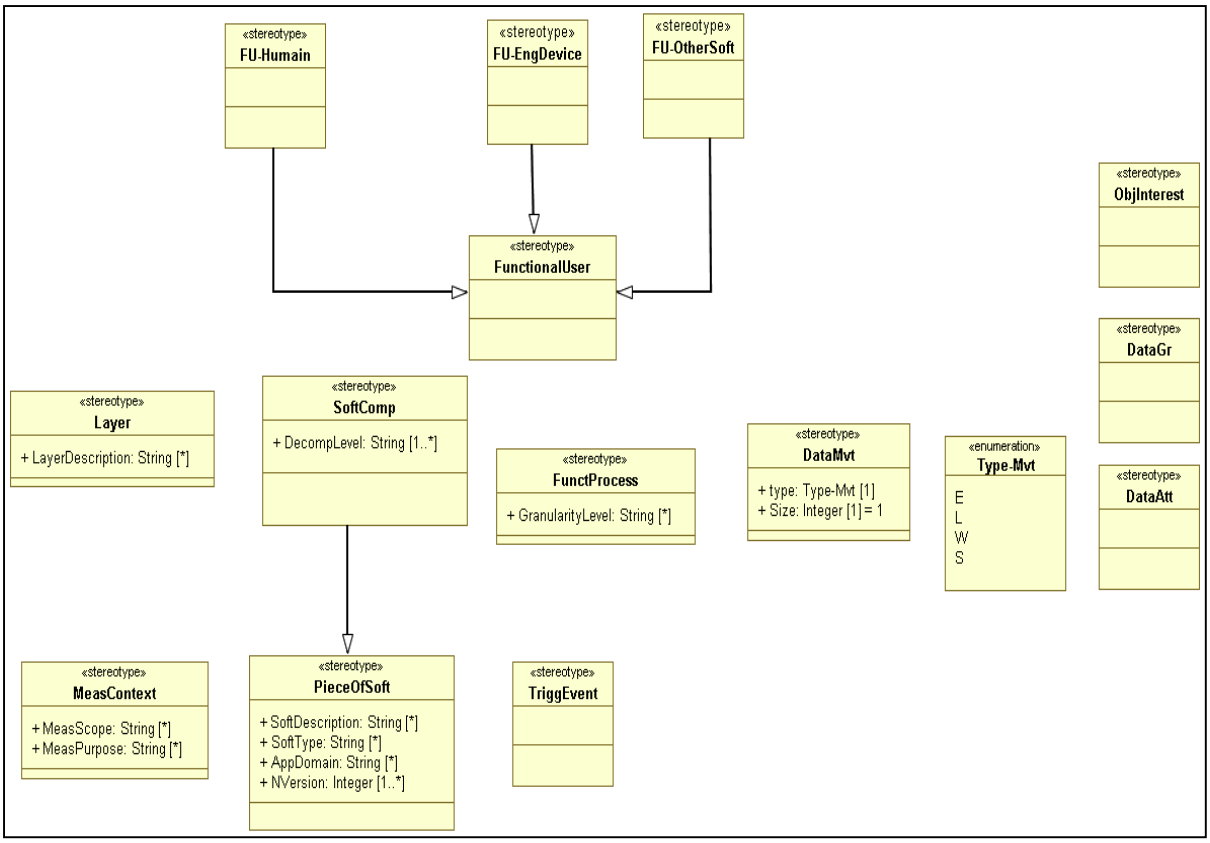


Figure 4.9 Définition des stéréotypes

Les propriétés des tagged values doivent être respectées dans le profil, voir figure 4.10.

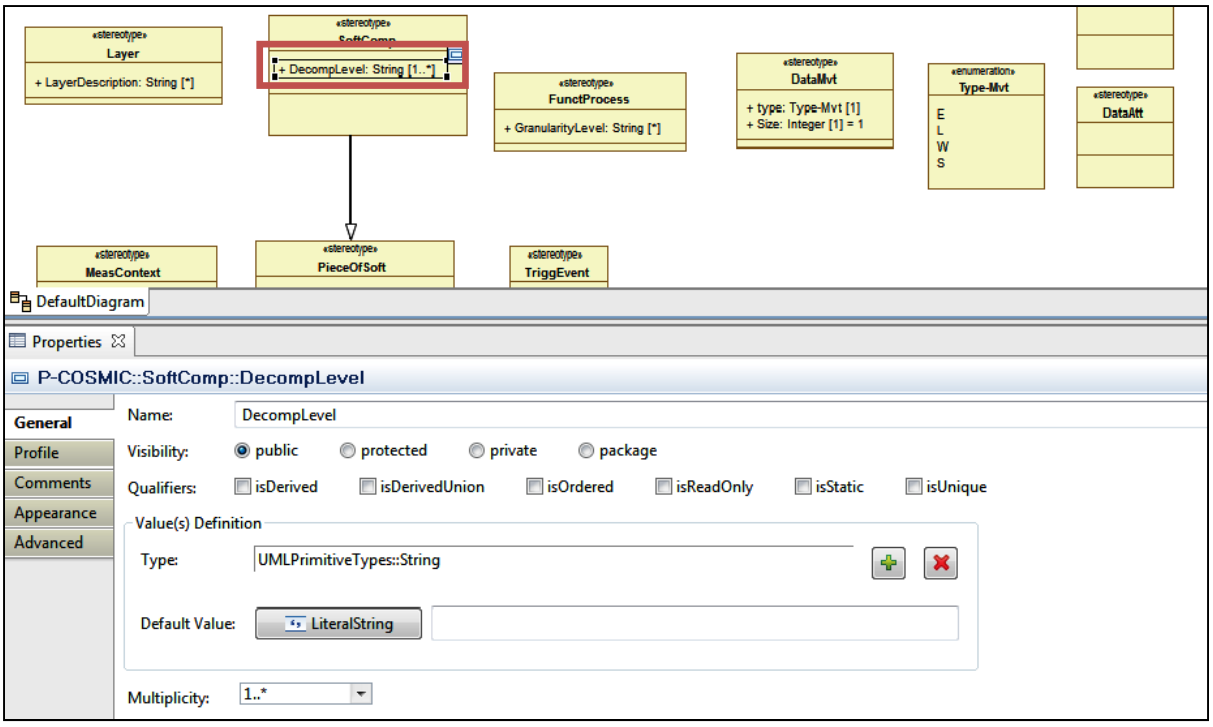


Figure 4.10 Définition des propriétés d'une valeur marquée

Après avoir défini les différents stéréotypes, il faut mettre en place les classes UML étendues pour avoir la fonctionnalité du stéréotype. La figure 4.11 est un exemple de la définition complète du stéréotype layer.

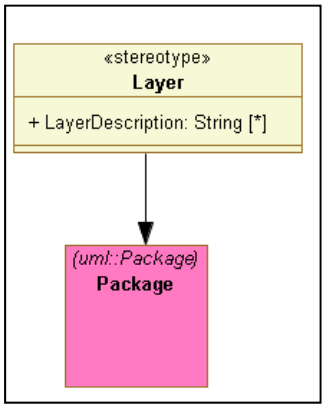


Figure 4.11 Extension du stéréotype Layer

Notons bien que le même principe s'applique sur le reste des stéréotypes du profil P-COSMIC. Le type de mouvement de donnée est illustré à l'aide d'une énumération qui prend une des quatre valeurs (E : entré, S : Sortie, W : écriture ou L : Lecture).

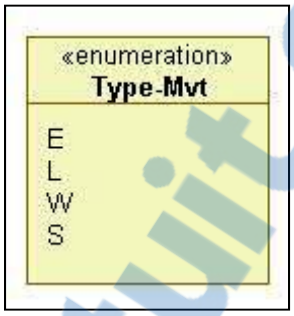


Figure 4.12 Énumération pour décrire le type d'un mouvement de données

Quant à la taille d'un mouvement de données, elle est initialisée à 1 puisque c'est une valeur unique.

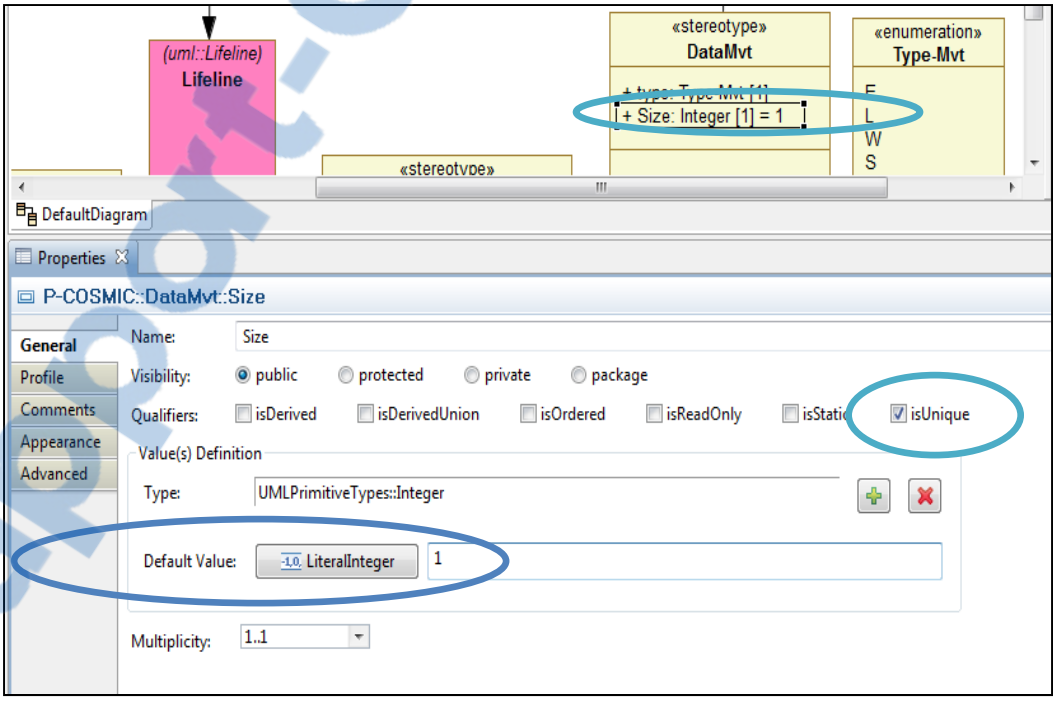


Figure 4.13 Unicité de la taille d'un mouvement de données

Le résultat final du profil conçu est représenté dans la figure 4.14.

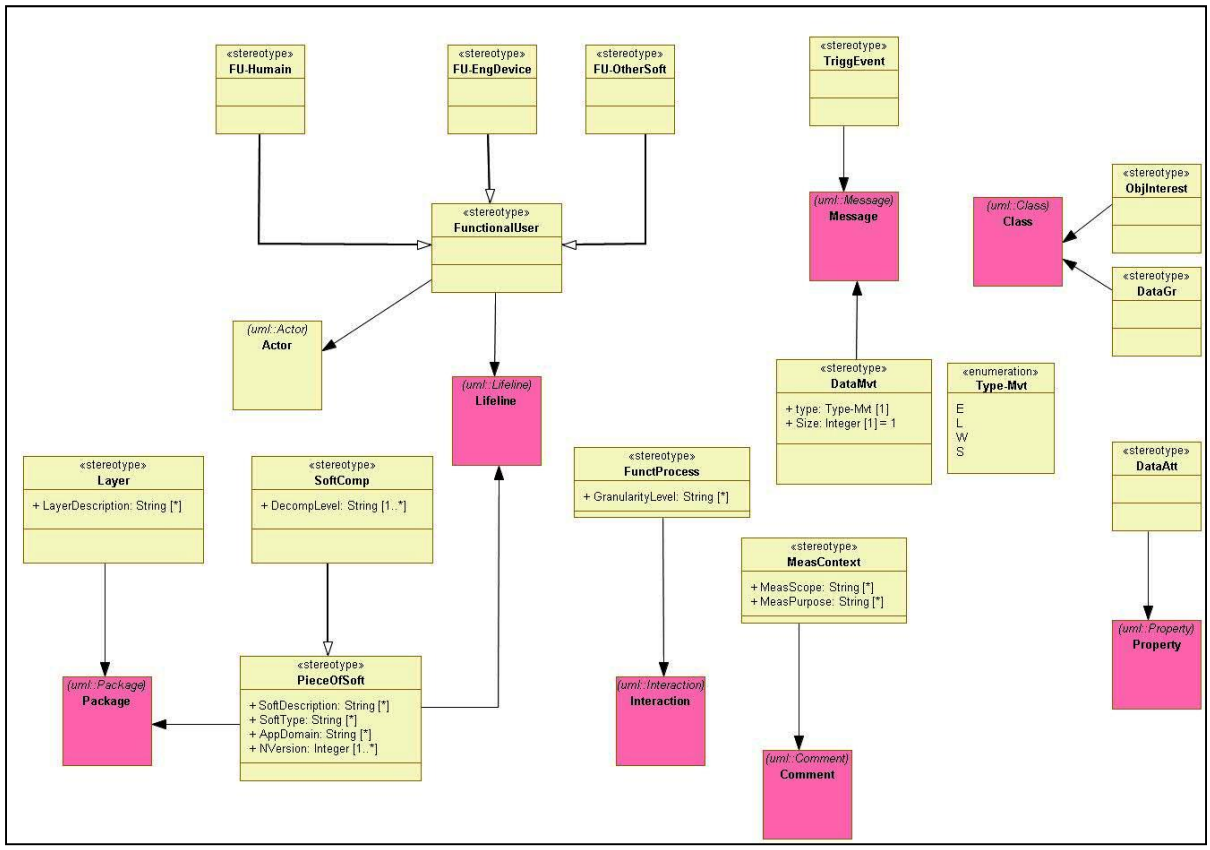


Figure 4.14 Conception du profil P-COSMIC

4.7 Conclusion

En conclusion, nous avons exposé dans ce chapitre les étapes suivies pour concevoir le profil P-COSMIC pour supporter la méthode de mesure COMSIC. Nous avons tout d'abord défini le modèle de domaine et les règles liées au méta-modèle. Nous avons par la suite présenté le mapping des concepts du domaine vers UML en définissant les stéréotypes. Le prochain chapitre fera l'objet d'une étude de cas qui illustre l'utilisation du profil proposé.

CHAPITRE 5

EXPÉRIMENTATION ET RÉSULTATS

5.1 Introduction

Ce chapitre a pour objectif d'illustrer l'utilisation du profil proposé P-COSMIC à l'aide d'une étude de cas de type temps réel. Le but de cet exemple est de mettre en évidence les stéréotypes proposés dans notre profil et leur utilisation dans la pratique. Le chapitre est composé de deux parties: la première introduit l'étude de cas et la deuxième une discussion des résultats obtenues et des propositions futures.

5.2 Application de P-COSMIC

Dans le but de tester l'applicabilité du profil P-COSMIC, nous avons choisi une étude de cas publiée par le groupe COSMIC, c'est l'étude de cas du cuiseur de riz nommée « Rice Cooker » (GÉLOG, Rice Cooker, 2010). La même étude de cas été révisée par les auteurs Lavazza *et al.* dans (Lavazza *et al.*, 2009).

L'étude de cas correspond à un logiciel de type temps réel. Ce choix revient au but de mettre en évidence la capacité de la méthode COSMIC à mesurer les applications de types embarqués et temps réel. De plus, à l'aide de cet exemple nous pouvons mettre en évidence la plupart des concepts définis dans le profil P-COSMIC.

Le but dans cette étude est de mesurer la taille fonctionnelle du morceau logiciel qui est le contrôleur du cuiseur de riz. Les spécifications du logiciel sont (Lavazza *et al.*, 2009) :

1. Le cuiseur à riz (Rice cooker) doit être capable de cuire le riz avec trois modes: rapide, normal et gruau. L'utilisateur sélectionne le mode via des boutons de sélection.

2. Lorsque l'utilisateur appuie sur le bouton START, le cuiseur à riz commence la cuisson et la lampe (Cooking lamp) est allumée. Si le bouton START est activé sans que l'utilisateur ne sélectionne un mode, le cuiseur à riz démarre automatiquement en mode normal.

3. Le chauffage (Heater) doit être contrôlé en fonction de la spécification de cuisson, une fonction qui détermine la température de cuisson sur la base du mode et du temps écoulé.

4. La période de référence pour l'ensemble du système sera assurée par une fonction particulière qui émet trois signaux, à savoir le temps écoulé à un chaque intervalle de une seconde depuis que le bouton START a été enfoncée, et les signaux à chacun des intervalles de 5 secondes et 30 secondes.

5. Toutes les 30 secondes, une température de consigne sera à nouveau établie.

6. Toutes les 5 secondes, la température actuelle du cuiseur sera obtenue à l'aide d'un capteur de température externe (Sensor) et comparée à la température cible. Le chauffage sera mis en service ON si la température actuelle est inférieure à la température cible et sera mis en OFF dans le cas contraire.

Pour tester notre profil, nous avons commencé par créer un modèle UML annoté avec le profil P-COSMIC nommé RiceCooker. Voir figure 5.1.

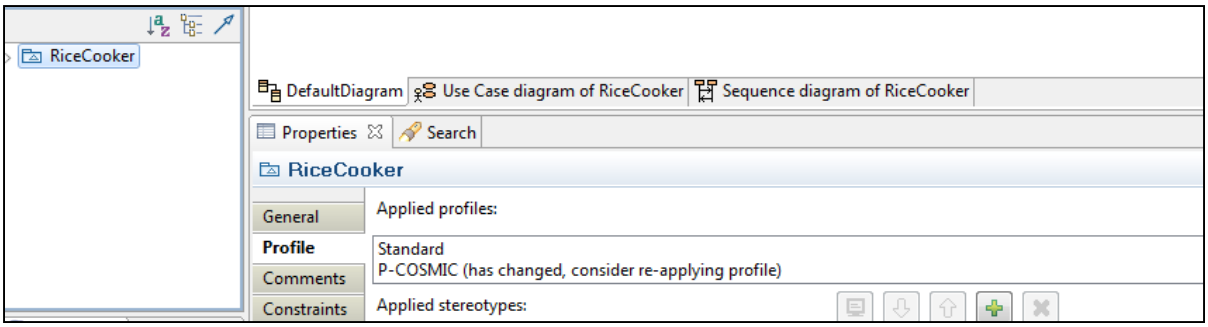


Figure 5.1 Application du profil P-COMIC au modèle RiceCooker

Pour illuster les concepts de morceau logiciel, couche, utilisateur fonctionnel et le contexte de mesure, nous avons conçu le diagramme de cas d'utilisation du rice cooker (voir Figure 5.2).

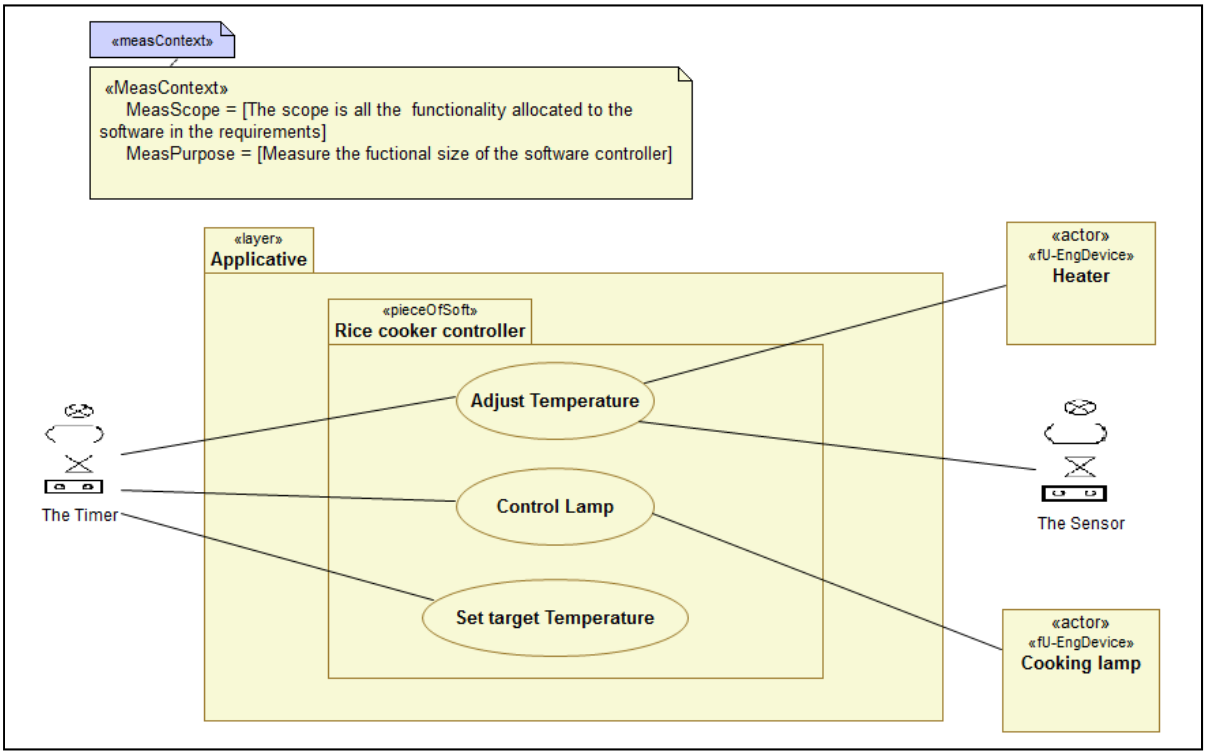


Figure 5.2 Diagramme de cas d'utilisation étendu par le profil P-COSMIC

Nous pouvons voir que le diagramme de cas d'utilisation du rice cooker n'a pas changé mais plutôt a été enrichi avec les concepts liés à COSMIC. Nous pouvons directement voir sur le diagramme le contexte de mesure et la couche à laquelle appartient le morceau logiciel à mesurer. De plus, un acteur est passé d'un acteur simple à un acteur spécifique au domaine du morceau logiciel. De plus, il est possible de l'afficher de deux façons, soit par une icône qui le décrit soit par un acteur sous forme d'un classifier stéréotypé avec le nom du stéréotype correspondant. La figure 5.3 montre plus précisément les deux formats.

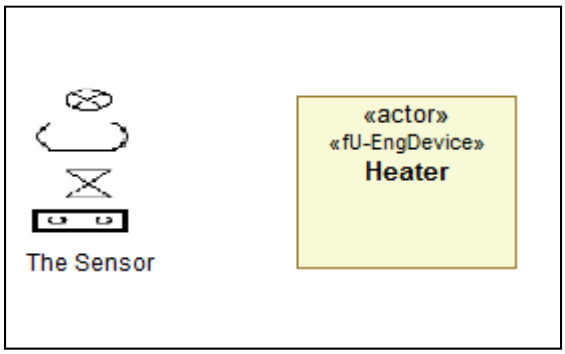


Figure 5.3 Représentations d'un utilisateur fonctionnel

De plus, dans l'onglet propriétés, nous pouvons vérifier les propriétés liées au morceau logiciel à mesurer. Ces propriétés représentent les valeurs marquées (Voir Figure 5.4).

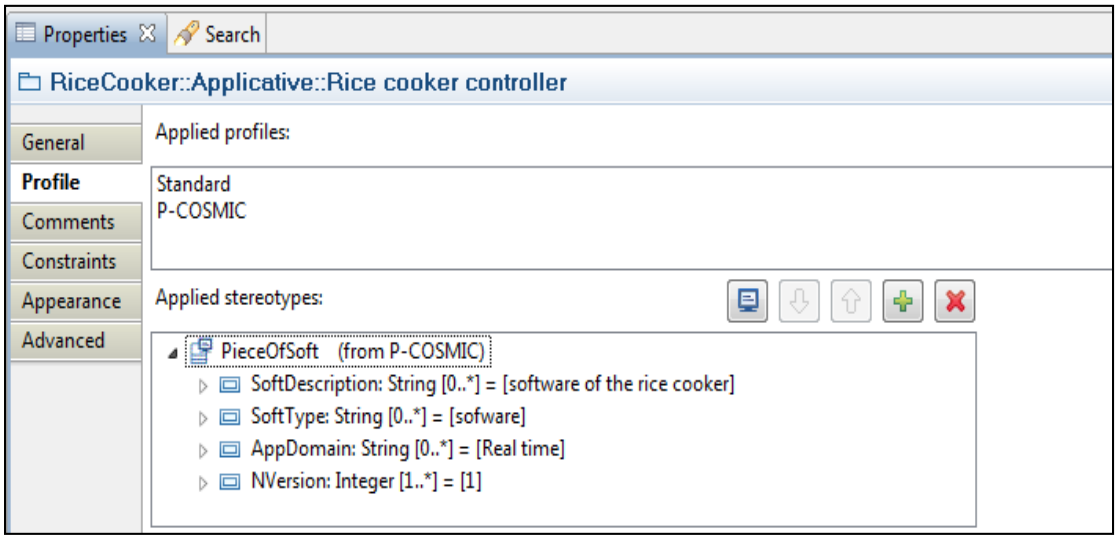


Figure 5.4 Propriétés du morceau logiciel Rice Cooker

Suivant le même principe introduit dans (Lavazza *et al.*, 2009), chaque cas d'utilisation sera représenté en détail sous la forme d'un diagramme de séquence pour représenter un processus fonctionnel. La Figure 5.5 illustre le processus fonctionnel nommé « Control cooking lamp » qui représente le cas d'utilisation « Contol Lamp ».

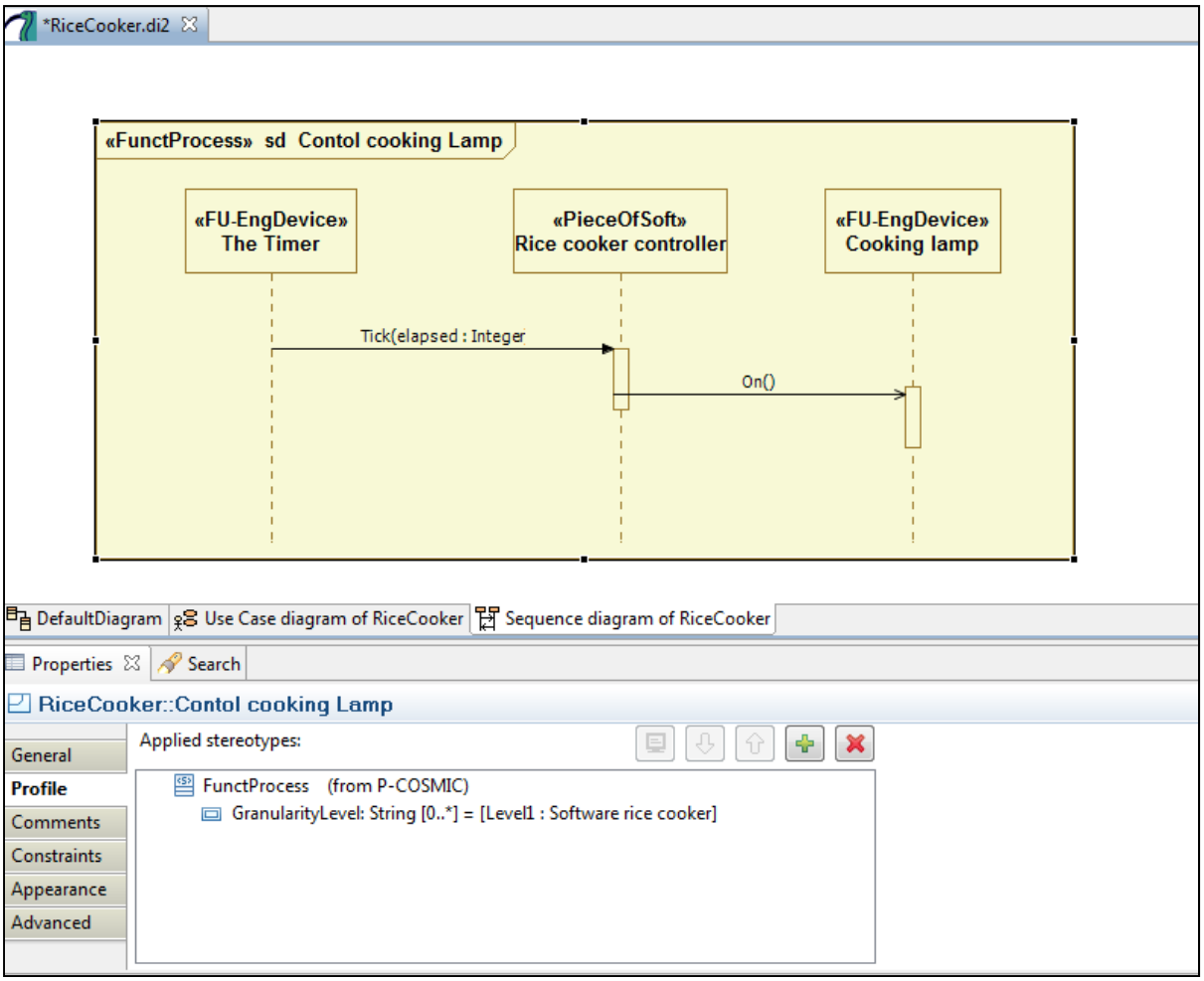


Figure 5.5 Représentation du processus fonctionnel « Control cooking lamp »

Le processus fonctionnel est représenté par une interaction qui contient des informations pertinentes pour la mesure. Nous remarquons que le diagramme est plus lisible pour un connaisseur du domaine de mesure. Il est assez clair qu'un premier utilisateur fonctionnel (Timer) envoi de l'information au logiciel à mesurer et un deuxième utilisateur fonctionnel

(Cooking Lamp) reçoit les données de ce dernier. De plus la propriété du processus fonctionnel indique le niveau de granularité dans lequel la mesure doit être faite.

Dans la figure ci-dessous, nous montrons la nature du message envoyé du « Timer » au « Software Controller », c'est bien un événement déclencheur.

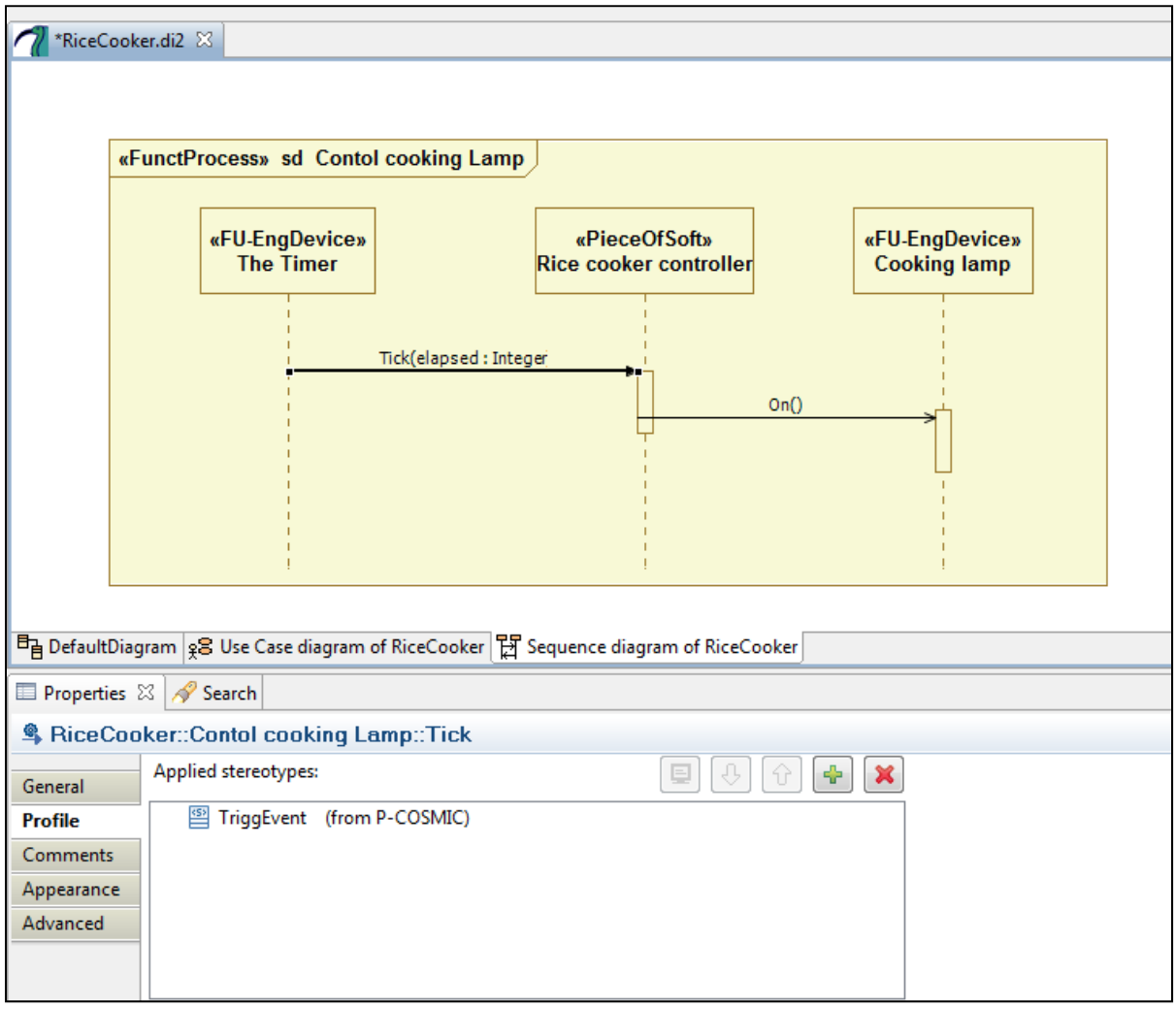


Figure 5.6 Identification de l'événement déclencheur « Tick »

Dans la même figure, nous basculons vers le deuxième message envoyé du « Software Controller » à l'utilisateur fonctionnel « Cooking Lamp », c'est un mouvement de données de type sortie (voir Figure 5.7).

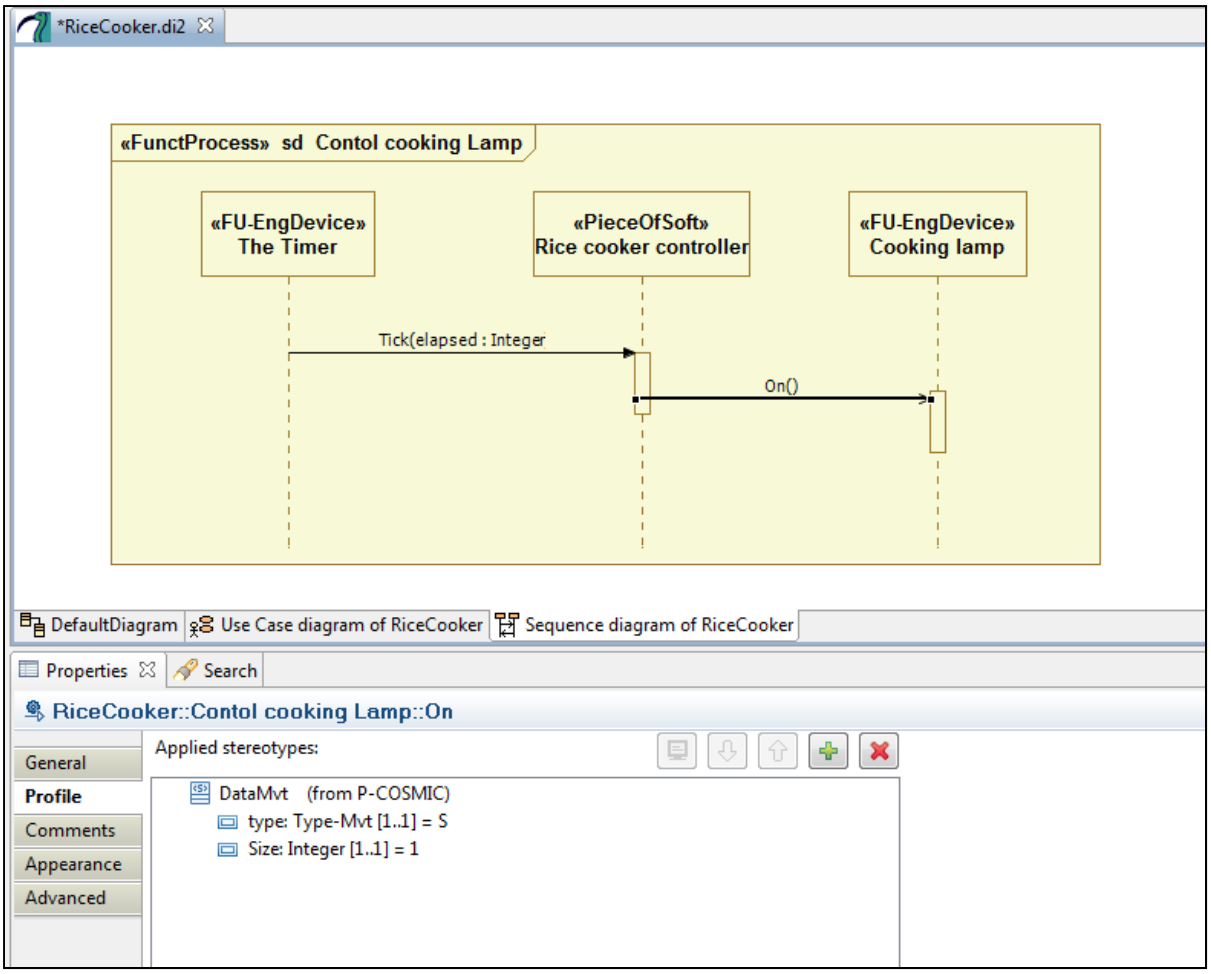


Figure 5.7 Identification d'un mouvement de donnée de type sortie

À l'aide de l'énumération définie dans le profil P-COSMIC, on peut facilement choisir un des quatre mouvements de donnée COSMIC (Voir figure 5.8).

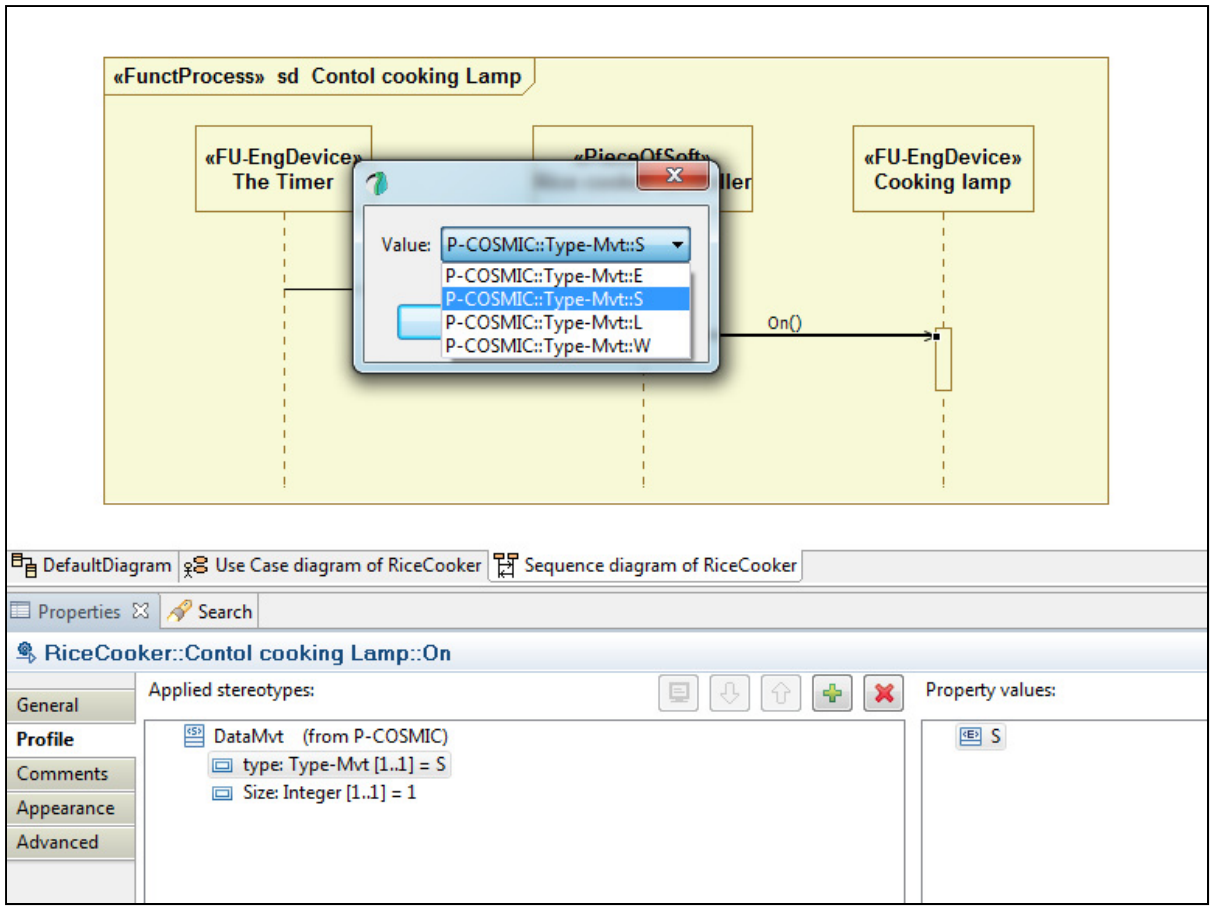


Figure 5.8 Restriction du choix du type d'un mouvement de donnée

Dans le but d'éviter les répétitions, nous allons nous limiter à une seule représentation d'un processus fonctionnel. L'objectif de cet exemple est de montrer à quel point le profil P-COSMIC peut faciliter la modélisation pour des fins de mesure.

À l'aide de cette étude de cas, nous avons pu capturer la plupart des concepts COSMIC à l'aide de notre profil P-COSMIC appliqué sur le modèle UML du Rice cooker. Dans la figure 5.9, nous montrons une partie du diagramme de classe du Rice cooker dans laquelle nous appliquons les stéréotypes ObjInterest pour modéliser un objet d'interêt, DataGr, pour modéliser un groupe de données et DataAtt pour représenter un attribut de donnée.

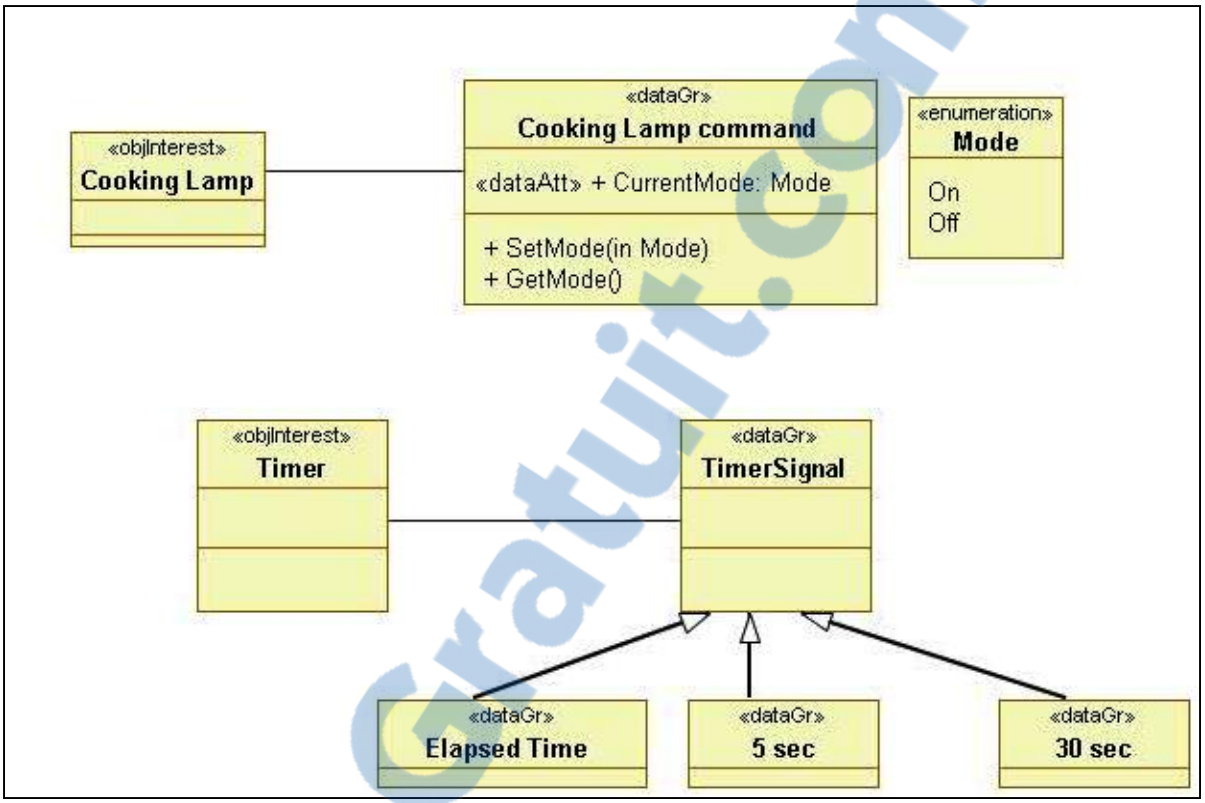


Figure 5.9 Extrait su diagramme de classe du Rice Cooker

5.3 Discussion des résultats et travaux futurs

Nous avons pu à l'aide de l'étude de cas du cuiseur de riz de montrer l'utilité du profil P-COSMIC et son application sur un modèle UML. En effet, un modélisateur peut désormais concevoir des modèles UML orientés mesure (COSMIC) en appliquant les nouveaux stéréotypes ajoutés à ses modèles. Cette tâche permettra au mesureur de gagner le temps qu'il peut perdre en faisant sortir manuellement les concepts dont il a besoin.

Avec le profil P-COSMIC, il devient aussi plus simple de lire un modèle UML orienté mesure et c'est là que réside la différence entre notre proposition et les autres propositions liées à ce sujet. De plus, à l'aide des nouveaux stéréotypes définis, un modèle UML peut contenir toute l'information pertinente pour établir la mesure avec COSMIC ou pour faire des documents de mesure automatiquement.

L'utilité du profil conçu ne s'arrête pas dans l'étape de modélisation. En effet, dans des travaux futurs, les informations capturées à l'aide du profil peuvent être utiles pour automatiser la mesure avec COSMIC.

L'idée est d'exporter les données COSMIC déjà définies dans un modèle UML qui étend le profil P-COSMIC dans un fichier XMI (XML [2] Méta Interchange). Ce dernier peut être une entrée pour un logiciel qui permet d'établir automatiquement la taille fonctionnelle et de générer des documents de mesure.

Nous constatons que le profil UML pour COSMIC pourrait être utile pour plusieurs raisons :

1) D'une part il permet à un modélisateur de concevoir des modèles UML orientés mesure sans pour autant perdre du temps (l'application d'un profil à un modèle UML n'est pas une tâche cruciale).
2) D'autre part, le mesureur gagnera le temps qu'il perd pour établir les documents de mesure et l'effort manuel pour le faire.

5.4 Conclusion

Dans ce dernier chapitre, nous avons présenté une étude de cas qui permet de valider partiellement l'utilité du profil UML proposé. Nous avons essayé d'appliquer tous les nouveaux stéréotypes sur les modèle UML de l'étude de cas afin d'expliquer l'applicabilité de notre travail sur un exemple réel. Ce dernier est alors annoté avec l'information pertinente pour COSMIC afin de faciliter le processus de mesure.

---

[2] *Extensible Markup Language*

CONCLUSION GÉNÉRALE

Le processus de mesure COSMIC inclut plusieurs concepts à partir desquels nous pouvons appliquer des règles de mesure qui permettent d'établir des modèles d'estimation et de productivité. Plusieurs chercheurs ont trouvé qu'UML est approprié pour faciliter, ou plus spécifiquement automatiser (partiellement ou totalement) la mesure avec COSMIC.

Quelques concepts de la méthode COSMIC n'ont pas été mappés à des concepts UML, tel que le concept d'événement déclencheur. Ce dernier a été représenté par un nouveau stéréotype proposé par Azzouz *et al.* (2004). Il est à noter que cette idée été une bonne initiative des auteurs de penser à étendre UML pour un modéliser un concept lié à la méthode COSMIC.

Dans ce mémoire, nous avons présenté une revue de littérature des travaux de recherche antérieurs qui ont porté sur la façon d'appliquer la mesure de la taille fonctionnelle avec COSMIC utilisant les spécifications écrites avec UML. Une comparaison entre les différents travaux nous a permis d'identifier les points communs entre les différentes approches.

Notre idée a été alors d'étendre UML pour COSMIC à l'aide d'un profil basé sur de nouveaux stéréotypes qui permettent de modéliser les concepts de la méthode COSMIC. Cette idée nous a parue intéressante pour différentes raisons. En effet, le profil permet de capturer toute l'information pertinente pour établir la mesure, ce qui facilite la tâche de lecture des spécifications des exigences pour le mesureur. De plus, le modélisateur peut concevoir des modèles UML orientés mesure à l'aide des nouveaux stéréotypes.

Le profil P-COSMIC a été conçu à l'aide de l'outil papyrus pour la modélisation avec UML 2. C'est un environnement qui offre un support très avancé pour la conception des profils. Nous avons pu aussi tester l'applicabilité du profil à l'aide d'une étude de cas d'une application de type temps réel. C'est une étude de cas qui a été publiée par le groupe COSMIC pour servir comme un référentiel pour la mesure avec COSMIC.

Bien que nous n'ayons pas pu valider l'utilité du profil en l'appliquant sur un cas plus complexe incluant des experts en COSMIC qui peuvent utiliser notre profil, nous sommes optimistes que notre approche pourrait être intéressante dans ce domaine.

Il serait possible de mener à une étude plus approfondie qui consiste à implémenter un outil qui se base sur le profil P-COSMIC. L'outil permettra d'extraire l'information résultante du profil et sert pour établir automatiquement la mesure et produire des documents de mesures.

RÉFÉRENCES BIBLIOGRAPHIQUES

Abran, A., Desharnais, J.M., Londeix, B., Stambollian, A. 2009. *Manuel de mesure de la méthode de mesure COSMIC de la taille fonctionnelle Version 3.0.1.* ISO 19761: Common Software Measurement International Consortium. 86 p.

Abran, Alain. 2010. *Software Metrics and Software Metrology*. Hoboken, N.J.: Wiley, Los Alamitos, Calif.: IEEE Computer Society, 328 p.

Abran, A., et Dumke, R. 2011. *COSMIC Function Points: Theory and Advanced Practices*, CRC Press, Boca Raton, 334 p.

Azzouz, S. et Abran, A. 2004. « A proposed measurement role in the Rational Unified Process (RUP) and its implementation with ISO 19761: COSMIC-FFP ». In *Software Measurement European Forum - SMEF*, Rome, Italy.

Azzouz, Saadi. 2003. « Calcul avec Iso 19761 de la taille de logiciels développés selon Rational Unified Process ». Thèse de master en ligne, Montréal, Université du Québec à Montréal – UQAM. <http://s3.amazonaws.com/publicationslist.org/data/gelog/ref-463/893.pdf>. Consulté le 20 Mars 2011.

Barkallah, S., Abdeouahed, G.et Abran, A. 2011. « COSMIC Functional Size Measurement Using UML Models ». In *Communication in Computer and information Science: software Engineering, Business Continuity and Education*, Jeju Island, Korea : Springer-Verlag. p.137-146

Bévo, V., Lévesque, G. et Abran, A. 1999. « Application de la méthode FFP à partir d'une spécification selon la notation UML : compte rendu des premiers essais d'application et questions ». In *International Workshop on Software Measurement (IWSM'99)*, Lac Supérieur, Canada.

Bévo, Valéry. 2005. « Analyse et formalisation ontologique des procedures de measures associées aux méthodes de mesure de la taille fonctionnelles des logiciels: de nouvelles perspectives pour la mesure », Thèse de doctorat, en ligne, Montréal, Université du Québec à Montréal, 314p. <http://www.dic.uqam.ca/upload/files/theses/bevo_these.pdf>. Consulté le 14 Mars 2011.

Blanc, X. et Salvatori, L. 2005. *MDA en action - Ingénierie logicielle guidée par les modèles,* Paris : Eyrolles 2005, 270 p.

Bobkowska, A. et Weihs, M. 2008. «Verification of the Fit to User Profiles for UML Tools». In *1st International Conference on Information Technology*, Gdansk, Pologne. En ligne, IEEE Xplore. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4621673>. Consulté le 10 septembre 2011.

Boehm, Barry W. 2000. *Software cost estimation with COCOMO II*. Upper Saddle River, N.J. : Prentice Hall PTR.

Common Software Measurement International Consortium group. 2012. *Page web officielle du groupe.* <http://www.cosmicon.com/>.

Condori-Fernández, N., Abrahão, S. et Pastor, O. 2007. « On the Estimation of Software Functional Size from Requirements Specifications ». *Journal of Computer science and Technology*, P.358-370.

Desharnais, J.M., Abran, A., Maya, M. et St-Pierre, D. 1998. *Mesure de la taille fonctionnelle des logiciels temps réel*. « Rapport Technique No. 12 ». Montréal: Université du Québec à Montréal, 17p. < http://www.gelog.etsmtl.ca/publications/pdf/406.pdf >. Consulté le 20 Mars 2011.

Fuentes-Fernández, L. et Vallecillo-Moreno, A. 2004. « An introduction to UML profiles ». In *UML and Model Engineering*, Vol. V, No. 2.

Gabay, J. et Gabay, D. 2008. *UML2 Analyse et conception*. Dunod, Paris.

GÉLOG. 2010. Rice Cooker Case Study. In *Site web de COSMIC - Case Studies*. En ligne. < http://www.cosmicon.com/portal/public/Rice_Cooker.pdf>. Consulté le 16 janvier 2011.

Habela, P., Glowacki, E., Serafinski, T. et Subieta, K. 2005. « Adapting Use Case Model for COSMIC-FFP Based Measurement ». In *15th International Workshop on Software Measurement – IWSM 2005*, Montréal, p. 195–207.

IFPUG group. *Site official du groupe IFPUG*. http://www.ifpug.org/

Jenner, Malcolm.S. 2001. « COSMIC-FFP and UML: Estimation of the Size of a System Specified in UML – Problems of Granularity». In *The 4th European Conference on Software Measurement and CT Control*, Heidelberg. p. 173–184.

Jenner, Malcolm.S. 2002. « Automation of Counting of Functional Size Using COSMIC-FFP in UML».In The 12th International Workshop Software Measurement. p. 43–51.

Kelvin, Lord. 1891. «Electrical Units of Measurement». In Popular lectures and addresses, vol 1. P80. London: Mac Millan.

Khelifi, Adel. 2005. «Un référenciel pour la mesure des logiciels avec la norme ISO 19761(COSMIC-FFP) : une étude exploratoire ». Thèse de doctorat en génie en ligne, Montréal, École de Technologies Supérieur, 230 p. In *La vitrine de diffusion des mémoires et thèses de l'ÉTS*. Consulté le 17 Février 2011.

Laboratory for research on Technology for Ecommerce LATECE. En ligne. http://www.latece.uqam.ca/fr/c_projets.html

Lavazza, L. et Del Bianco, V. 2009. « A Case Study in COSMIC Functional Size Measurement: the Rice Cooker Revisited ». In *IWSM/Mensura, Amsterdam.*

Lavazza, L et Robiolo, G. 2010. « Introducing the Evaluation of Complexity in Functional Size Measurement: a UML-based Approach ». In *ESEM'10 Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement.* Bolzano-Bozen, Italy.

Levesque, G., Bévo, V. et Cao, D.T. 2008. « Estimating software size with UML models». In *Proceedings of the 2008 C3S2E Conference*, Montreal. p. 81–87.

Lind, K. et Heldal, R. 2011. «A Model-Based and Automated Approach to Size Estimation of Embedded Software Components». In *Model Driven Engineering Languages and Systems*. Berlin Heidelberg : Springer-Verlag. P.334-348.

Lind, K. et Heldal, R.2011. «CompSize: Automated Size Estimation of Embedded Software Components».In *IWSM/MENSURA 2011.* Nara, Japan. En ligne, IEEE Xplore. < http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6113048>. Consulté le 19 janvier 2012.

Mannox, Alan. 2005. *Rapid J2EE Development : An adaptive Foundation for Entreprise Applications.* Pearson Education, Inc. United States.

Marin, B., Giachetti, G. et Pastor, O. 2008. « Measurement of Functional Size in Conceptual Models : A Survey of Measurement Procedures based on COSMIC ». In MENSURA 2008, Munich, Germany. November 18-19. p. 170–183

Object Management Group. *UML 2.4 Infrastructure specification.* < http://www.omg.org/spec/UML/2.4/>, consulté le 12 juin 2012.

Object Management Group. *UML 2.4 Superstrucure specification.* < http://www.omg.org/spec/UML/2.4/>, consulté le 12 juin 2012.

Object Management Group.2012. *Le catalogue les spécifications des profils UML*. http://www.omg.org/technology/documents/profile_catalog.htm

Object Management Group.2012. *À propos de UML.*En ligne. Consulté le 12 juin 2012.

Object Management Group.2012. *À propos de l'OMG.*
En ligne. < http://www.omg.org/gettingstarted/gettingstartedindex.htm>, consulté le 17 Mai 2012.

Object Management Group.2012. *À propos de MDA : Model Driven Architecture.* En ligne.
<http://www.omg.org/mda/>, consulté le 17 Mai 2012.

Oman, Paul et Shari Lawrence Pfleeger. 1997. *Applying Software Metrics*. The Institute of Electrical and Electronics Engineers, Inc. IEEE Computer Society, 319 p.

Papyrus. 2011. Open source tool for graphical UML 2 modelling. En ligne.
< http://www.papyrusuml.org>. Consulté le 15 septembre 2011.

Sellami, A. et Abran, A. 2006. « Analyse d'une mesure de la taille fonctionnelle du logiciel selon le point de vue de la métrologie et application à la méthode COSMIC-FFP ». *Revue Génie Logiciel*, vol.1, no 74, p. 2-12.

Sellami, A et Ben Abdallah, H. 2009. « Functional Size of Use Case Diagrams: A Fine-Grain Measurement ». In *4th* International Conference on Software Engineering Advances, ICSEA 2009. September 20-25. Porto, Portugal. p. 282–288.

Selic, Bran. 2007. « A Systematic Approach to Domain-Specific Language Design Using UML» *In The 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*. IEEE Xplore,
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4208820>. Consulté le 17 Mars 2012.

Symons, Charles. 1999. « *COSMIC FFP – Buts, approche conceptuelle et progrès* ». ASSEMI, Paris.

Van den Berg, K., Dekkers, T. et Oudshoorn, R. 2005. « Functional Size Measurement applied to UML-based user requirements ». In t*he 2nd Software Measurement European Forum*, Rome, Italy. p. 69–80.

Vilus, L. et Lévesque, G. 2004. « Une méthode efficace pour l'extraction des instances de concepts dans une spécification UML aux fins de mesure de la taille fonctionnelle de logiciels ». In t*he 17th International Conference Software & Systems Engineering & their Applications, ICSSEA 2004*, Paris, 30 Novembre au 4 Décembre.

Visual Paradigm. 2011. *UML profile management.* En ligne, < http://www.visual-paradigm.com/product/vpuml/tutorials/umlprofile.jsp>. Consulté le 08 septembre 2011.