UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ANALYSE ET FORMALISATION ONTOLOGIQUE DES PROCEDURES DE MESURE ASSOCIÉES AUX METHODES DE MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS: DE NOUVELLES PERSPECTIVES POUR LA MESURE

THÈSE
PRESENTÉE
COMME EXIGENCE PARTIELLE
DU DOCTORAT EN INFORMATIQUE COGNITIVE

PAR
EVARISTE VALERY BEVO WANDJI

AVRIL 2005

REMERCIEMENTS

En premier lieu, je tiens à remercier *Ghislain LÉVESQUE* et *Jean-Guy MEUNIER* d'avoir accepté de diriger cette thèse. Vos remarques pertinentes et éclairées m'ont permis de mieux structurer mes idées, de mieux les décrire.

Je remercie *Alain ABRAN* et chacun des membres du LRGL en particulier, pour l'accueil chaleureux que vous m'avez réservé dans l'équipe et pour l'ambiance familiale dans laquelle j'ai baignée pendant les premières années de ce doctorat (une pensée particulière pour *Michèle HÉBERT* et *Pierre BOURQUE*). Vous faites désormais partie de ma famille élargie.

Je remercie l'*IFI* et à travers elle l'AUF pour l'opportunité qu'ils m'ont offerte et qui a débouché sur cette thèse. Un merci particulier à *Marc BOUISSET* qui a crû en moi et en mon potentiel.

Je suis très honoré que *Marc FRAPPIER* de l'Université de Sherbrooke, *Pierre POIRIER* et *Bernard LEFEBVRE* de l'Université du Québec À Montréal aient accepté d'évaluer ce travail.

Je remercie aussi *Johanne GÉLINAS* du programme de Doctorat en Informatique Cognitive pour le soutien administratif et logistique qu'elle a toujours su apporter aux étudiants du programme.

Je tiens à dire ma reconnaissance à *Roger NKAMBOU* du GDAC, *Raphaël NBOGNI, Roland YATCHOU* et chacun des membres de Le Groupe Infotel Inc. pour leur soutien. Une pensée particulière à *Hervé DONFACK* et *Ghislain NGANTCHAHA* (je vous dis simplement MERCI).

Cette thèse, notamment dans sa phase terminale, doit beaucoup à mon épouse *Yolande ADDIE BÉVO*, pour sa patience et son soutien tant moral que spirituel pendant les moments difficiles. Merci d'avoir crû en moi, d'avoir toujours été là, au moment où j'en avais vraiment besoin.

Je ne saurais oublier de remercier mes parents, mes frères et sœurs pour leur soutien inconditionnel. Il m'a été simplement très précieux. Une pensée toute particulière à *Bertille MBEUTCHOU WANDJI* qui nous a malheureusement quitté.

Je tiens également à remercier tous ceux et celles qui de près ou de loin ont contribué de quelque manière que ce soit à l'aboutissement de cette thèse. Je pense particulièrement aux familles *Mésaac NOUNJIO, Robert DEMBI, Abel NDJIKI, Jude Jacob NSIEMPBA*. Ma profonde reconnaissance va aussi à l'endroit de tous ceux et celles que j'ai sûrement eu l'indélicatesse d'oublier.

TABLES DES MATIÈRES

LISTES DES FIGURES

LISTE DES TABLEAUX

LISTE DES ACRONYMES ET ABRÉVIATIONS

CBR : Case Base Reasoning

CMM : Capability Maturity Model

CMMI : Capability Maturity Model - Integration

COSMIC-FFP : Common Software Measurement International Consortium – Full Function Point (méthode de mesure de la taille fonctionnelle des logiciels maintenue par le groupe COSMIC)

CPR : Case Problem Reasoning

EIF : External Interface File

EI : External Input

EO : External Output

EQ : External Inquiry

FPA : Function Point Analysis (méthode de mesure de la taille fonctionnelle des logiciels maintenue par le groupe IFPUG [International Function Point Users Group])

FUR : Functional User Requirement

IA : Intelligence Artificielle

ILF : Internal Logical File

LRGL : Laboratoire de Recherche en Gestion de Logiciels

LANCI : Laboratoire d'Analyse Cognitive de l'Information

Mk II FPA : Mark II Function Point Analysis (méthode de mesure de la taille fonctionnelle des logiciels maintenue par le groupe UKSMA [United Kingdom Software Measurement Association])

ROOM : Real-time Object Oriented Modeling

STI : Système Tutoriel Intelligent

SMA : Système Multi-agent

UML : Unified Modeling Language

RÉSUMÉ

Bien des auteurs estiment que la taille du logiciel est l'un des paramètres les plus importants, sinon le plus important, pour faire de l'estimation (coût de projet ou effort) ou pour effectuer des analyses de productivité et de qualité. Le logiciel étant un produit intellectuel, un «objet abstrait », l'évaluation/la mesure de sa taille, n'est pas chose évidente. Dans l'industrie en général, la taille des logiciels est exprimée en points de fonction (taille fonctionnelle) ou en lignes de code source. L'atout majeur de la mesure de la taille fonctionnelle des logiciels, réside dans le fait qu'elle peut être faite très tôt dans le cycle de vie du logiciel. L'implantation des systèmes existants de mesure de la taille fonctionnelle des logiciels au niveau de l'industrie doit après la bataille administrative (acceptation des décideurs, motivation, engagement), faire face à deux problèmes majeurs d'ordre technique : la difficulté à appliquer les méthodes de mesure (ce qui rend fastidieuse la tâche du mesureur et nécessite parfois l'intervention d'un ou plusieurs experts malheureusement pas toujours disponibles), le manque ou la rareté d'outils d'assistance pour l'application des méthodes de mesure.

Dans ce projet de recherche l'attention est focalisée sur les *procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels*. Il y est jeté les bases d'un corpus de «connaissances » associé au processus d'application de chaque méthode de mesure, à travers ***une formalisation ontologique des procédures de mesure***. Une telle formalisation pourrait être exploitée dans le cadre du développement d'outils d'aide à la mesure. L'accent est mis sur le cas particulier des outils d'« automatisation » de la mesure : une *approche orientée ontologie est proposée pour l'automatisation d'une bonne partie du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications présentées dans un formalisme bien défini (syntaxe et sémantique)*. L'approche introduite est basée sur un *modèle cognitif décrivant une procédure de mesure*. Ce modèle est en fait une généralisation des ontologies de tâches préconisées dans le cadre de la formalisation ontologique des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels. Il est inspiré du *modèle d'un processus cognitif d'interprétation*. Finalement, l'approche pour l'automatisation est illustrée avec la *construction d'un prototype qui automatise partiellement la mesure de la taille fonctionnelle des logiciels avec la méthode COSMIC-FFP, lorsque les spécifications sont disponibles dans le formalisme UML standard*. Une *architecture générale pour un système d'automatisation de la mesure* est également proposée.

ABSTRACT

Many authors consider that software size is one of the most important parameters, if not the most important, in estimation activities (cost or effort related to a project) or for quality and productivity analysis. Software being an intellectual product, an "abstract object", measuring it size is not obvious at all. In the industry, the most popular units for software size are function points and source lines of code. The main advantage of software functional size measurement is that it can be performed very early in the software lifecycle. But measuring software functional size in the industry still faces two major technical problems: (1) applying existing measurement methods is not an easy task (it sometimes requires the help of one or many experts, who are unfortunately not numerous for instance in the world); (2) the lack or rarity of measurement tools to help measurers applying measurement methods.

In this thesis, the focus is on *measurement procedures related to software functional size measurement methods*. Basis is set for a body of knowledge related to each measurement method application process, through **an ontological formalisation of measurement procedures**. Such a formalisation could be useful for a better understanding of measurement procedures or when developing measurement tools. Automation tools development is examined in this thesis: A *general architecture for such a tool is introduced*; an *ontology driven approach is also introduced for partial automation of a software functional size measurement method application process*. The approach is developed upon a *cognitive model describing a measurement procedure*. This model is in fact a generalisation of tasks ontologies related to measurement procedures. It is inspired from the *model of an interpretation cognitive process*. Finally the introduced approach is illustrated in developing a *prototype for measuring software functional size using the COSMIC-FFP method, from software specifications presented in standard UML formalism.*

INTRODUCTION

« *L'abeille confond par la structure de ses cellules de cire l'habileté de plus d'un architecte. Mais ce qui distingue dès l'abord le plus mauvais architecte de l'abeille la plus experte, c'est qu'il a construit la cellule dans sa tête avant de la construire dans la ruche.* » [Karl Marx, *Le capital*, 1, p. 428]

FEUILLE DE ROUTE

1. *Contexte du travail et motivations*
2. *Questions spécifiques de recherche*
3. *Objectifs fixés au départ pour la thèse*
4. *Structuration de la thèse et plan de lecture*

Georges Vignaux [Vignaux 92] définit les **sciences cognitives,** ou plutôt les **sciences de la cognition** (pour être plus précis tel que suggéré par Jean-Louis Lemoigne [Lemoigne 90]), comme étant un *ensemble de disciplines s'appliquant à analyser les comportements intelligents (celui de l'homme, des animaux ou des machines) et à analyser les supports matériels qui paraissent conditionner ces comportements (le cerveau ou l'ordinateur, par exemple).* Au cœur du débat en sciences cognitives, l'on retrouve parmi les théories majeures, la théorie computationnelle appliquée à des comportements dit intelligents. Au rang des outils utilisés par les chercheurs pour exprimer leurs thèses relativement à cette théorie et sur l'activité mentale en général, l'on retrouve l'ordinateur. Ainsi, l'ordinateur et par ricochet l'informatique (en tant que «science des ordinateurs »), représente l'un des centres d'intérêt de l'activité de recherche en sciences cognitives. L'ordinateur ici est un outil parmi plusieurs et est pris à titre de modèle. Inversement, les sciences cognitives ont un impact indéniable sur les théories informatiques, notamment les théories de modélisation. Les modèles dits cognitifs dans les travaux de recherche en informatique (informatique cognitive notamment) sont aujourd'hui monnaie courante (modèles connexionnistes, modèles symboliques et modèles dynamiques). À la suite de ces modèles, on a assisté à l'émergence d'une nouvelle catégorie de logiciels dits « intelligents » (systèmes experts, systèmes tutoriels intelligents, *systèmes multi-agents*, …) qui manipulent les modèles suscités et simulent l'activité mentale pour résoudre certains problèmes du monde réel. Ainsi, la question d'exploitation à bon escient des sciences cognitives pour s'attaquer à certains problèmes dans le domaine de l'informatique, constituerait une piste intéressante pour les chercheurs en informatique. Le projet de recherche que nous avons mené dans le cadre de cette thèse, s'insère dans ce cadre.

1 CONTEXTE DU PROJET DE THESE ET MOTIVATIONS

L'importance que revêt aujourd'hui l'estimation (coûts de projets ou effort), les analyses de productivité et de qualité, en matière de gestion de projets informatiques en général, n'est plus à démontrer [Jones 96a][1]. Relativement à l'estimation, qu'il s'agisse d'un projet de développement ou de maintenance informatique, il est nécessaire, déjà en tout

[1] Approximately 40% of all projects fail due to lack of management control. (Coopers & Lybrand - Sept. 1995)

début de projet, d'avoir une idée de l'effort requis (en termes de ressources tant humaines que matérielles et financières) pour sa réalisation. Quant à la productivité, il s'agit d'un paramètre essentiel pour bon nombre d'entreprises. C'est un paramètre qui pourrait attirer l'attention des gestionnaires sur le processus de production en place dans une corporation. Pour ce qui est de la qualité, dans le contexte de « course » à l'amélioration des processus de développement de logiciels qui concerne bon nombre (sinon la plupart) des compagnies de développement à travers le monde depuis quelques années maintenant, l'assurance qualité est incontournable. Relativement au CMM [Capability Maturity Model] (devenu CMMI [Capability Maturity Model - Intégration]), l'assurance qualité constitue l'un des critères requis pour atteindre les plus hauts niveaux du modèle [Van den Berg 98] (au niveau 4, un des « *key process areas* » est la gestion de la qualité logicielle). Qu'il s'agisse de l'estimation (coût de projet ou effort), ou des analyses de qualité et de productivité associées au développement et à la maintenance des produits logiciels, **la mesure apparaît comme une activité incontournable** [Jones 96a]. Le modèle CMMI lui accorde d'ailleurs une attention assez particulière (au niveau 2, un des «*key process areas* » est réservé à la mesure). Il est stipulé qu'un programme de mesure doit être appliqué aux produits logiciels en vue de développer une compréhension quantitative de la qualité desdits produits et atteindre les objectifs spécifiques de qualité. En outre, une étude effectuée par *Peter Kulik & Catherine Weber* [Kulik et al. 02b], dont les résultats ont été publiés en mars 2002, indique l'importance reconnue de la mesure au sein des organisations, dans leurs activités de développement de logiciels. Parmi les principales mesures reconnues dans le domaine du logiciel, il y a celle de la taille des logiciels. La mesure de la taille d'une application logicielle désigne le processus permettant d'assigner à l'attribut « taille » de l'application, une valeur numérique ou symbolique, suivant un ensemble de règles définies. Cette valeur indique comment grosse est l'application. Bien des auteurs estiment que la taille du logiciel est **l'un des paramètres les plus importants, sinon le plus important, pour faire de l'estimation (coût de projet ou effort) ou effectuer des analyses de productivité et de qualité [Bradford 97, Jones 96b, Isakowitz 02].**

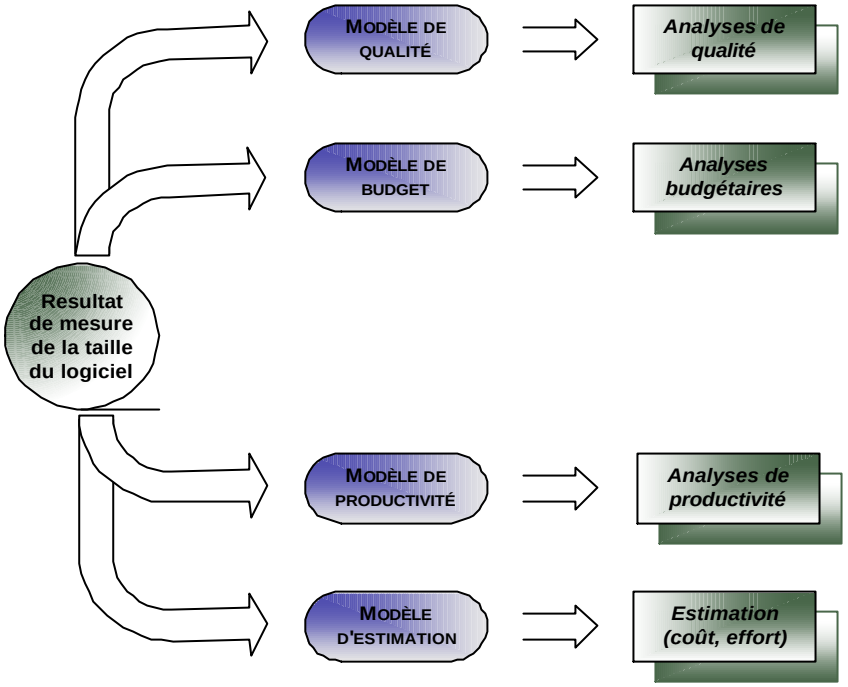


Figure 1 : Exploitation des résultats de mesure la taille du logiciel [Jacquet et al. 97]

Comme l'indique la Figure 1, la taille du logiciel est utilisée dans les modèles de qualité, de budget, de productivité ou encore d'estimation, lesquels modèles sont respectivement exploités dans les analyses de qualité, budgétaires, de productivité et d'estimation. Cependant, le logiciel étant un produit intellectuel, un «objet abstrait » [Naur et al. 69][2], l'évaluation/la mesure de sa taille, n'est pas chose évidente [Putnam et al. 92, Zuse 03] et ne fait pas toujours l'unanimité. Il n'est donc pas surprenant qu'il existe à ce jour plusieurs approches pour mesurer la taille d'un logiciel. Les principales approches sont : l'approche basée sur les fonctionnalités (« function-point sizing » : la taille est exprimée en points de fonction), l'approche basée sur les lignes de code source (« Code/Change sizing » : la taille est exprimée en nombre de lignes de code source [SLOCs[3]]) et l'approche basée sur les composants (« Standard-component sizing » : la taille est exprimée en nombre de composants) [Putnam et al. 92]. La question de la prééminence de l'une ou l'autre approche

2 Le produit logiciel peut être considéré comme un objet abstrait qui a évolué à partir d'un énoncé d'un besoin pour se finaliser par un logiciel, lequel comprend tout aussi bien le code, objet ou source, que les diverses formes de documentation produites tout au long du processus de développement (Naur et al., 1969)

Logiciel : l'ensemble complet des programmes, des procédures et de la documentation connexe associée à un système, et particulièrement à un système informatique. (Bohem, B. W., 1981 « Software Engineering Economics », Prentice Hall.)

3 Source Lines Of Code

reste d'actualité à ce jour. Dans ce projet de thèse nous nous intéressons à la mesure de la taille fonctionnelle des logiciels (« function-point sizing »).

Nous focalisons notre attention sur les **procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels**. Notre préoccupation première est la **formalisation** *de ces procédures*. Une telle formalisation constitue un pas décisif vers la ***facilité d'utilisation***, l'***objectivité*** et la ***répétabilité*** (donc l'« automatisabilité ») qui figurent au premier plan parmi les caractéristiques désirables pour la «nouvelle génération » des méthodes de mesure de taille fonctionnelle des logiciels [Symons et al. 99]. En effet, le type de formalisation que nous préconisons (**formalisation ontologique**), ouvre de nouvelles perspectives pour les principales pistes que nous avons recensées dans la littérature, dans le cadre de l'assistance relativement à l'application des méthodes de mesure. Ces pistes sont : (1) l'apprentissage traditionnel des méthodes de mesure (interaction entre un expert et un utilisateur), (2) le développement d'outils d'aide à la mesure (système tutoriel intelligent [STI], questionnaire [liste de questions spécifiques permettant d'orienter un utilisateur en fonction de ses réponses], système de raisonnement à base de cas [CBR], de raisonnement à base de problèmes [CPR] ou système de diagnostic…), (3) l'automatisation (partielle ou complète) des procédures de mesure. Présentement, ces pistes se heurtent à un certain nombre de difficultés, dont une bonne partie pourraient être surmontées grâce à la formalisation. La piste (1) par exemple, fait face au problème de la disponibilité des experts (ils sont malheureusement *peu nombreux*); Son succès dépend fortement de la transmission adéquate des connaissances nécessaires (il faudrait donc les identifier clairement et les rendre accessibles au plus grand nombre). De plus, la mesure reste manuelle (tâches répétitives et donc fastidieuses à la longue). Quant aux pistes (2) et (3), elles sont fortement dépendantes d'une formalisation adéquate des méthodes de mesure (notamment des procédures de mesure associées), ce qui fait défaut actuellement. De plus, ces pistes sont encore en pleine exploration et font l'objet de plusieurs travaux de recherche depuis quelques années maintenant [Gramantieri et al. 97, Ho et al., 99, Paton 99, Edge 00, Diab et al. 01, Uemura et al. 01, Desharnais 03, Azzouz et al. 04]. Dans cette thèse, seule la piste (3) retient notre attention (bien que notre travail puisse être profitable aux autres pistes). Nous montrons comment la formalisation que nous préconisons pourrait être exploitée dans le cadre du développement d'outils d'aide à la mesure (nous nous limitons au cas particulier des outils d'« automatisation » de la mesure). Pour ce faire, nous développons une **approche**

orientée ontologie *pour l'automatisation d'une bonne partie du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications présentées dans un formalisme bien défini (syntaxe et sémantique)*. L'approche développée est basée sur un **modèle cognitif décrivant une procédure de mesure**, lequel modèle est inspiré du *modèle d'un processus cognitif d'interprétation*. Elle est illustrée à travers la construction d'un prototype qui automatise partiellement la mesure de la taille fonctionnelle des logiciels avec la méthode COSMIC-FFP, lorsque les spécifications sont disponibles dans le formalisme UML standard.

Sur le plan théorique, les principales questions de recherche auxquelles nous nous attaquons dans la thèse sont relatives à la représentation et l'exploitation des connaissances relevant du domaine de la mesure de la taille fonctionnelle des logiciels à partir des spécifications.

2 QUESTIONS SPÉCIFIQUES DE RECHERCHE

Trois questions servent de fil conducteur dans ce travail de recherche :

1. Quelles sont les connaissances en jeu dans le processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels (on parle aussi de «procédure de mesure»[4] [Abran et al. 03, ISO/IEC 03a]) ? *[Comprendre les procédures de mesure]*
2. Comment représenter ces connaissances de manière non seulement à les rendre accessibles aussi bien au novice qu'à l'expert en mesure, mais également à les rendre exploitables par les concepteurs d'outils logiciels pour la mesure (systèmes tutoriels intelligents, systèmes d'analyse et d'exploitation des résultats de mesure, systèmes experts pour l'aide à la mesure, systèmes d'automatisation complète ou partielle de la mesure...) ? *[Décrire les procédures de mesure]*
3. Comment exploiter ces connaissances dans le cas particulier d'un système qui automatiserait une bonne partie du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications présentées dans un formalisme bien défini ? *[Systématiser les procédures de mesure]*

[4] Dans tout le document les expressions suivantes sont équivalentes : « processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels » et « procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels »

Ces questions constituent les piliers de la stratégie que nous avons adoptée pour mener à bien cette thèse. Les objectifs fixés au départ sont précisés dans la section qui suit.

3 OBJECTIFS DE LA THESE

Nous nous proposons dans cette thèse de jeter les bases de la représentation des connaissances dans domaine de la mesure de la taille fonctionnelle des logiciels, à la lumière des travaux de recherche menés en Intelligence Artificielle (IA) en matière de représentation des connaissances. Plus spécifiquement, nous nous sommes fixés deux objectifs principaux pour cette thèse :

- Il s'agit d'abord de jeter les bases d'un corpus de connaissances associées au processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels. À la lumière des possibilités offertes par les sciences de la cognition, sera proposée et illustrée, une approche pour la formalisation des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels. Pour l'illustration, les méthodes COSMIC-FFP [Abran et al. 03, ISO/IEC 03a], Mk II FPA [UKSMA 00, ISO/IEC 02] et FPA [IFPUG 01, ISO/IEC 03b], seront utilisées.
- Il s'agit ensuite de montrer comment une formalisation telle que celle susmentionnée pourrait être exploitée dans le cadre du développement d'outils d'aide à la mesure (nous nous limitons aux outils d'automatisation de la mesure). Toujours à la lumière des possibilités offertes par les sciences de la cognition, sera proposée et illustrée, une approche pour l'automatisation d'une bonne partie des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels lorsque les spécifications sont fournies dans un formalisme bien défini (comme le formalisme UML standard par exemple). Pour l'illustration, il sera construit, suivant l'approche proposée, un prototype permettant d'automatiser une bonne partie du processus d'application de la méthode de mesure COSMIC-FFP à partir de spécifications présentées dans la notation UML standard. Ce prototype pourrait être arrimé ultérieurement, à un outil case tel que Rational Rose^{TM}, qui permet de produire des spécifications de logiciels dans le formalisme UML. Les spécifications produites avec l'outil case serviraient alors d'entrées pour le prototype.

Il est à noter que la finalité de ce travail n'était pas la vérification par l'analyse statistique des réponses aux questions de recherche posées au départ. Il s'agissait plutôt, d'élaborer une théorie sur un phénomène observable (la représentation des « connaissances » relatives aux procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels, l'automatisation complète ou partielle de telles procédures). Par conséquent, nous avons procédé par des observations qualitatives sur le domaine visé (le domaine de la mesure de la taille fonctionnelle des logiciels), suivi d'analyses et de la formulation d'hypothèses. La consultation directe de certains experts disponibles (en sciences de la cognition et en mesure de la taille fonctionnelle des logiciels) a été requise, question d'assurer un degré de crédibilité *raisonnable* aux hypothèses formulées.

L'organisation de la thèse est détaillée dans la section qui suit.

4 STRUCTURATION DE LA THÈSE ET PLAN DE LECTURE

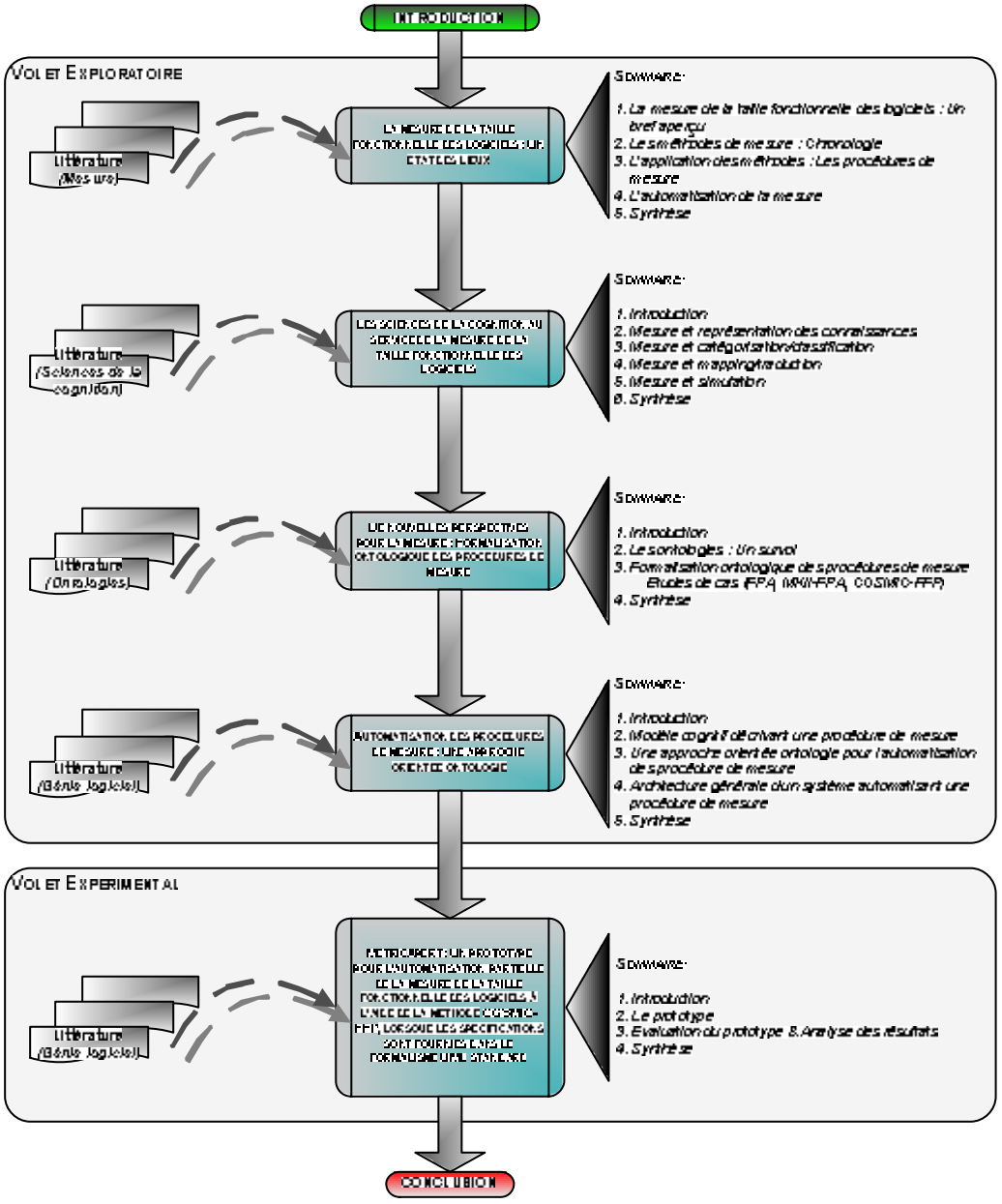


Figure 2 : Structuration de la thèse et plan de lecture

Cette thèse comporte deux volets principaux : Un **volet exploratoire** et un **volet expérimental**. Pour une meilleure compréhension, nous suggérons au lecteur de commencer par l'Annexe A de ce document (éléments de terminologie spécifiques à cette thèse), d'enchaîner avec le volet exploratoire, suivi du volet expérimental.

Le volet exploratoire se penche sur les questions de *formalisation et d'automatisation des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels*, à la lumière des possibilités offertes par les sciences de la cognition, notamment en matière de ***représentation et exploitation des connaissances***. Il s'organise en quatre (4) chapitres. Le **premier chapitre** est consacré à une revue de littératures et offre un panorama sur la mesure de la taille fonctionnelle des logiciels (les motivations, le principe, les principales méthodes de mesure et leur exploitation). Ce chapitre lève un pan de voile sur les difficultés auxquelles les utilisateurs des méthodes de mesure sont confrontés, lesquelles difficultés justifient ce travail de thèse. Il en ressort que la formalisation et l'automatisation des procédures de mesure associées aux méthodes de mesure, sont des pistes prometteuses pour surmonter une bonne partie de ces difficultés. Ces pistes sont examinées dans le **chapitre deuxième**, sous le prisme des sciences de la cognition. Le chapitre fait ressortir les principales questions théoriques relevant des sciences de la cognition, et jugées pertinentes relativement aux questions de formalisation et d'automatisation des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels. Tour à tour, y sont abordées les questions de représentation des connaissances, de catégorisation/classification, de « mapping » et de simulation. La pertinence de ces questions dans le domaine de la mesure de la taille fonctionnelle des logiciels est mise en exergue. Ainsi la mesure de la taille fonctionnelle des logiciels pourrait tirer profit de certains travaux de recherche en sciences cognitives. Dans cette thèse, les travaux sur les questions mentionnées plus haut sont exploités, à des degrés divers. Par exemple, les travaux sur la représentation des connaissances sont exploités pour la formalisation des procédures de mesure, au cœur du **chapitre troisième**. Le type de formalisation que nous proposons (formalisation ontologique) y est documenté. Des études de cas de formalisation ontologique de procédures de mesure associées à des méthodes de mesure y sont également présentées. La question de la validation de telles formalisations y fait l'objet d'une section. Ce type de formalisation est au centre de l'approche présentée dans le **chapitre quatrième** (approche orientée ontologie) pour l'automatisation des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels. L'approche proposée trouve son fondement dans un modèle cognitif décrivant une procédure de mesure. Ce modèle est détaillé dans le chapitre. Il s'agit en fait d'une **généralisation** des ontologies de tâches que nous préconisons dans le cadre de la formalisation ontologique des procédures de mesure. Pour appuyer l'approche

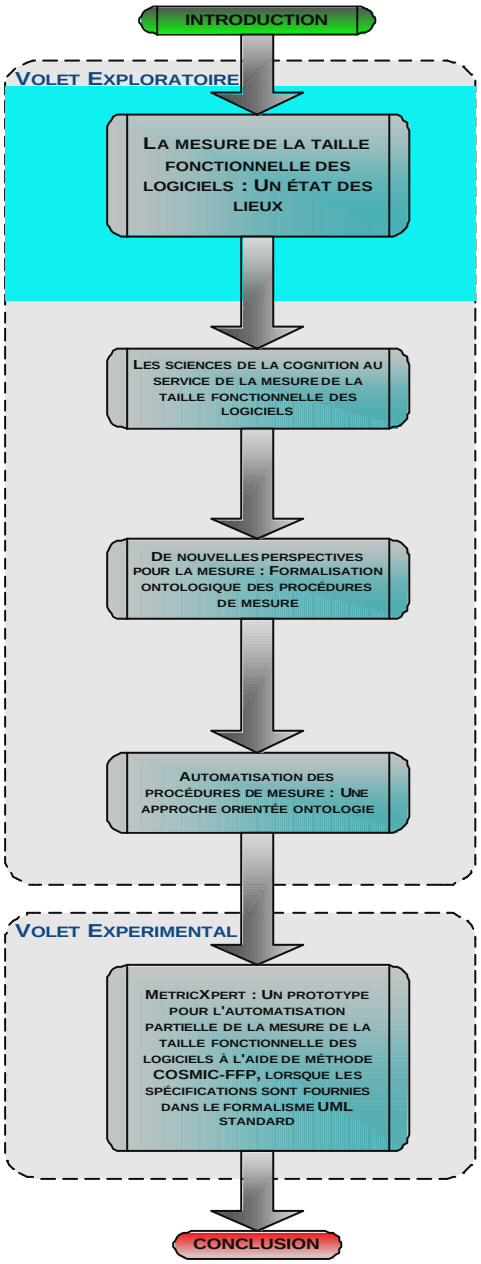
présentée dans le chapitre et la rendre plus concrète, une architecture générale pour un système automatisant une procédure de mesure est spécifiée et décrite. Cette architecture est partiellement mise en œuvre dans le volet expérimental de la thèse.

Ce dernier volet de la thèse qui compte un seul chapitre (**chapitre cinquième**), montre entre autres à travers le prototype qui y est présenté, comment les formalisations proposées dans le volet exploratoire pourraient être exploitées dans le cadre du développement d'outils d'aide à la mesure (notamment les outils d'automatisation de la mesure*)*. Le prototype est construit suivant l'approche orientée ontologie proposée dans le volet exploratoire pour l'automatisation. L'évaluation du prototype ainsi que l'analyse des résultats de l'évaluation occupent une place non négligeable dans le chapitre.

Remarque : *Veuillez vous reporter à l'Annexe G pour le planning détaillé de la thèse.*

CHAPITRE 1.

LA MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS : UN ÉTAT DES LIEUX



FEUILLE DE ROUTE

1. *La mesure de la taille fonctionnelle des logiciels : Un bref aperçu*
2. *Les méthodes de mesure : Chronologie*
3. *L'application des méthodes : Les procédures de mesure*
4. *L'automatisation de la mesure*
5. *Synthèse*

EN BREF …

Ce chapitre introduit la mesure de la taille fonctionnelle des logiciels (les motivations, le principe, sa place dans les activités associées au développement logiciel, …). Un diagramme chronologique des principales méthodes de mesure y est fourni. La question de l'application des méthodes de mesure est également abordée, ainsi que celle de l'automatisation de la mesure. Pour chacune de ces questions, un état des lieux est fait et les principales difficultés à surmonter sont identifiées. Le chapitre s'achève sur une note d'espoir avec la possible contribution des sciences de la cognition pour s'attaquer aux difficultés recensées.

1 LA MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS : UN BREF APERÇU

Dans l'industrie en général, la taille des logiciels est exprimée en points de fonction (taille fonctionnelle) ou en lignes de code source [Kulik et al. 02b]. L'approche de mesure de la taille basée sur les lignes de code source est la plus populaire. Elle consiste en une estimation du volume de code source associé à une application logicielle. La mesure se fait par analogie ou à l'aide d'un compteur automatique de lignes de code sources. Cette approche souffre de la non existence d'un standard définissant la notion de « ligne de code source » (implantation physique ou énoncé logique). De plus, cette approche s'avère peu pertinente (pour une bonne estimation de coût ou d'effort) avec les environnements de programmation modernes (langages visuels et générateurs de codes), et également du fait que les « lignes de code source » ne sont disponibles que trop tard dans le processus de développement. L'autre approche (approche par points de fonction) a été introduite dans les années 70, comme alternative aux lignes de code source. Ce n'est que depuis les années 90 qu'elle a commencé à gagner du terrain notamment en Amérique du nord. Au cours de la dernière décennie, elle a connu une expansion fulgurante (Etats-Unis, Canada, Australie, Nouvelle Zélande, Afrique du sud, Inde, certains pays de la communauté européenne et d'Amérique du sud). Elle est même devenue, selon certains auteurs, une approche incontournable pour la mesure de la taille des logiciels, dans certains des pays sus-cités [Jones 96a]. Elle consiste en une **détermination de la quantité de fonctionnalités requises et/ou fournies aux utilisateurs pour une application logicielle donnée**. On parle de la mesure de la taille fonctionnelle de l'application logicielle considérée. A la base donc de l'approche, il y a ce qui apparaît dans la norme ISO/IEC 14143-1:1998 [ISO/IEC 98] sous le vocable FUR (Functional User Requirement), désignant les « fonctionnalités utilisateurs requises » (***les fonctionnalités fournies aux utilisateurs pour une application logicielle sont décrites à travers ces FURs***). D'une certaine manière, les FURs représentent les « possibilités d'utilisation »/services offerts aux utilisateurs par une application logicielle donnée (cf. Annexe A). Les FURs sont identifiables implicitement ou explicitement à partir des spécifications ou plus généralement à partir des artefacts des logiciels : Certains auteurs distinguent les FURs pré-implantation (issus des artefacts associés à la définition des besoins, aux données d'analyse/modélisation, ou résultant de la décomposition fonctionnelle

des besoins), des FURs post-implantation (issus des programmes physiques, des manuels et procédures opérationnelles, ou encore des artefacts associés au stockage physique des données).

L'atout majeur de la mesure de la taille fonctionnelle des logiciels réside dans le fait qu'elle peut être faite très tôt dans le cycle de vie du logiciel (phase d'analyse ou de conception). En effet, le résultat de la mesure peut être exploité à des fins d'estimation (coût de projet, effort), un travail revêtant un caractère essentiel pour les chefs de projet en tout début de projet. De plus, la taille obtenue est indépendante de la technologie utilisée pour implanter les logiciels (plateforme de développement, langage de programmation, etc.). Comme autre point d'orgue, contrairement à l'approche basée sur les lignes de code source, l'approche de mesure de la taille des logiciels basée sur les fonctionnalités bénéficie d'un début de standardisation [ISO/IEC 98, ISO/IEC 97a, ISO/IEC 97b]. Le document ISO/IEC 14143-1 [ISO/IEC 98], publié en 1998, définit les concepts fondamentaux de la mesure de la taille fonctionnelle des logiciels et précise les principes généraux pour l'application d'une méthode de mesure de la taille fonctionnelle des logiciels. Il identifie un ensemble de concepts obligatoires requis et de critères que doit rencontrer une méthode de mesure de la taille fonctionnelle des logiciels pour être reconnue conforme au standard. Parmi les principaux concepts obligatoires définis par l'ISO notons à titre d'exemple :

- Le *Base Functional Components* (BFC) qui désigne une unité élémentaire de FUR (Functional User Requirement) définies et utilisées par une méthode de mesure de la taille fonctionnelle des logiciels pour des besoins de mesure. *Exemple : Soit le FUR « Gestion des clients ». Ce FUR pourrait consister en les BFCs suivantes: «Ajout d'un nouveau client», «Listing des achats d'un client» et «Mise à jour des informations relatives à un client»;*

- Le *Base Functional Component Types* (BFC Types) qui fait référence à une catégorie de BFCs. *Exemple : Pour la méthode de mesure FPA, on a les BFC Types suivants : « External Input », « External Output », « Logical Transaction », « Internal Logical File », etc. Pour la méthode de mesure COSMIC-FFP, on a les BFCs Types suivants : Entry type, Exit type, Read type, Write type;*

- La *frontière (Boundary)* qui fait référence à une interface conceptuelle entre le logiciel considéré et ses utilisateurs.

Parmi les critères que doit rencontrer une méthode de mesure de la taille fonctionnelle des logiciels pour être reconnue conforme au standard, notons qu'elle doit :

- pouvoir être appliquée dès que les FURs sont définis et disponibles;
- être autant que possible indépendante d'une méthode/technologie de développement logiciel particulière;
- inclure les activités suivantes pour déterminer la taille fonctionnelle d'un logiciel : (a) Déterminer l'envergure de la mesure; (b) Identifier les FURs contenus dans l'envergure (« scope ») de la mesure; (c) Identifier les BFCs constituant chaque FUR; (d) Classifier les BFCs suivant les BFCs Types lorsque applicable; (e) Assigner une valeur numérique appropriée à chaque BFC; (f) Calculer la taille fonctionnelle du logiciel;
- spécifier le type d'information nécessaire pour son application;
- définir une unité de mesure dans laquelle doit être exprimée la taille fonctionnelle.

Les principales méthodes de mesure (COSMIC-FFP, MkII-FPA et FPA) sont devenues des standards ISO depuis mars 2003. Un travail de généralisation de la mesure a été entrepris par le groupe COSMIC [Symons et al. 99] avec les diverses versions de la méthode de mesure des points de fonction étendue, fruit de l'analyse d'un certain nombre de méthodes de mesure parmi les plus connues. Notre travail s'insère à la suite de ce travail de généralisation. Dans la section suivante, nous nous intéressons à l'évolution des méthodes de mesure de la taille fonctionnelle des logiciels.

2 LES MÉTHODES DE MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS : CHRONOLOGIE

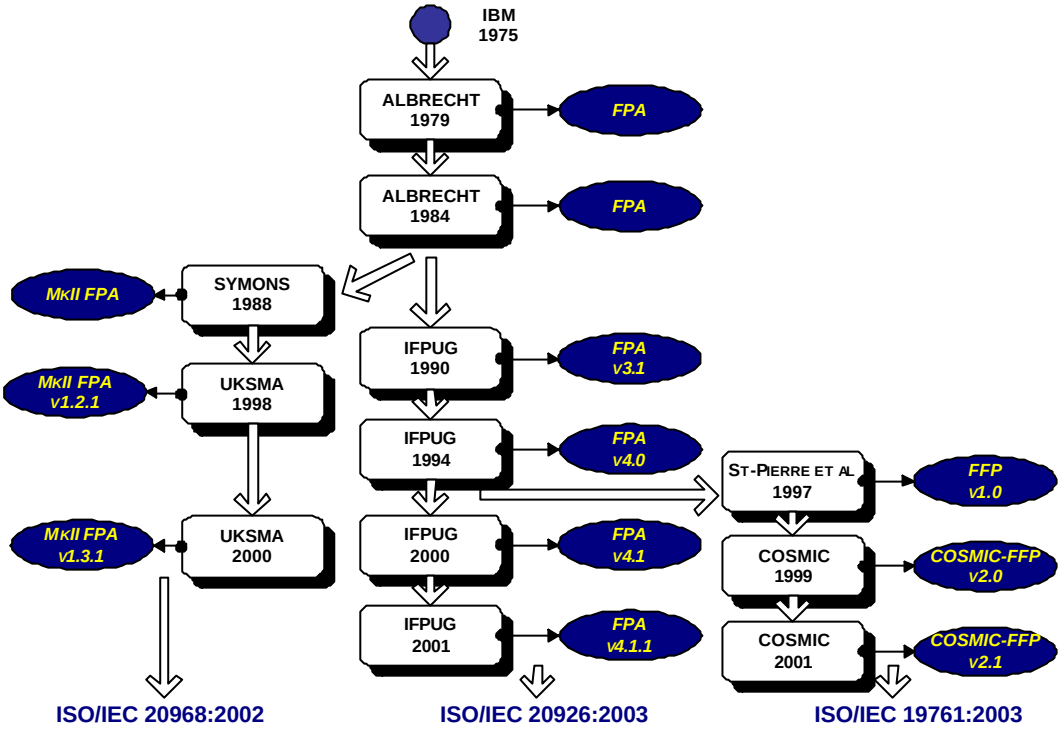


Figure 3 : Diagramme chronologique des principales méthodes de mesure de la taille fonctionnelle des logiciels

L'histoire de la mesure de la taille fonctionnelle des logiciels débute dans les années 75 au sein d'une équipe de la société IBM (le « Data Processing Services group »). Albrecht qui y travaille alors, propose cette nouvelle approche de mesure pour la taille des logiciels. Il s'agit d'une alternative aux lignes de codes, qui permet de mesurer la productivité économique (« Bien et services produits par unité de travail ou de dépense »). Le premier article abordant le sujet n'est publié qu'en octobre 1979. La première méthode de mesure de la taille fonctionnelle des logiciels est alors rendue publique. Elle porte le nom FPA (Function Point Analysis). Quelques années après sa mise au point, un groupe sera crée pour assurer sa maintenance : Il s'agit du groupe IFPUG (International Function Point Users Group) qui regroupe les experts et utilisateurs de la méthode. La méthode en est aujourd'hui à sa version 4.1.1 [IFPUG 01, ISO/IEC 03b]. Elle a été adoptée comme standard par l'ISO en mars 2003 (ISO/IEC 20926:2003). Une pléiade de variantes de cette méthode existe à ce

jour [Jones 96b] : Mk II FPA (Mark II Function Point Analysis) [UKSMA 00, ISO/IEC 02], F.F.P. (Full Function Points) devenue COSMIC-FFP (Common Software Measurement International Consortium - Full Function Points) [Abran et al. 01, Abran et al. 03, ISO/IEC 03a], Assert-R. [Reifer 90], 3D Function Points [Whitmire 95], variante FPA-NESMA (Netherlands Software Metrics Association), Feature Points [Jones 96a] et bien d'autres. De toutes ces variantes seules deux (2) à notre avis sont émergentes (COSMIC-FFP et Mk II FPA), du moins si nous en jugeons par la quantité de références dans la littérature associée au domaine, ou encore par la place occupée dans l'industrie selon leurs auteurs à la suite de quelques études menées. Ce sont d'ailleurs les seules avec FPA, qui ont été adoptées comme standard ISO à cette date (mai 2004).

Mk II FPA apparaît en 1988, à la suite de critiques formulées par Charles Symons relativement à la version 1984 de FPA. Symons propose une méthode plus simple, dépouillée des facteurs de complexité introduits par la méthode FPA et compatible avec les idées d'analyse et conception structurées. En moyenne, les deux (2) méthodes tendent à donner la même taille pour des logiciels de taille allant jusqu'à 400 points de fonction. Pour des logiciels de plus grande taille, Mk II FPA tend à donner des tailles supérieures à celles obtenues avec FPA. Un comité a été mis sur pied pour la maintenance de la méthode : il s'agit de l'UKSMA (United Kingdom Software Metrics Association). La méthode en est aujourd'hui à sa version 1.3.1. Elle a été adoptée comme standard par l'ISO en 2002 (ISO/IEC 20968:2002). A l'instar de FPA, Mk II FPA n'est pas universellement applicable à tous les types de logiciels. Par exemple, les caractéristiques fonctionnelles des logiciels de type temps-réel, systèmes ou embarqués ne sont pas adéquatement pris en compte par les deux méthodes. Jusqu'en septembre 1997, il n'existe aucune variante de FPA adéquate pour la mesure de la taille fonctionnelle des logiciels de type temps-réel, systèmes ou embarqués. A cette date, une équipe conjointe L.R.G.L. (Laboratoire de Recherche en Gestion de Logiciels) et S.E.L.A.M. (Software Engineering Laboratory in Applied Metrics) rend publique la version 1.0 de la méthode F.F.P. (Full Function Point).

FFP est conçue au départ pour permettre la mesure de la taille fonctionnelle des logiciels de type temps-réel. A la suite de tests effectués non seulement par le L.R.G.L. et le S.E.L.A.M., mais également dans certaines organisations, il appert que la méthode n'est pas uniquement efficace pour la mesure de la taille fonctionnelle des logiciels de type temps-réels, mais aussi des logiciels de type systèmes (en l'occurrence les systèmes

d'exploitation), ou encore de type MIS (Management Information Systems). En 1998, le groupe COSMIC (Common Software Measurement International Consortium) voit le jour et la tâche de maintenance de la méthode lui est confiée. La méthode est passée au peigne fin par le groupe. La nouvelle version de la méthode apparaît en 1999 sous un nouveau nom de baptême : COSMIC-FFP (Common Software Measurement International Consortium - Full Function Points). La méthode en est aujourd'hui à sa version 2.2. Elle a été adoptée comme standard par l'ISO en mars 2003 (ISO/IEC 19761:2003).

En dépit des efforts significatifs observés ces dernières années, avec notamment la standardisation des méthodes, l'application des méthodes de mesure de la taille fonctionnelle des logiciels demeure une entreprise difficile et ardue. Elle fait appel à une expertise propre pour chacune des méthodes [Low 90]. Chaque méthode met à la disposition des utilisateurs (dans le manuel mesure associé) une procédure de mesure, qui devrait être suffisamment claire et explicite pour permettre aux utilisateurs de se débrouiller tout seuls. Dans la suite, nous donnons un aperçu des procédures de mesure associées aux principales méthodes de mesure identifiées plus haut, telles qu'elles apparaissent dans les manuels de mesure.

3 L'APPLICATION DES MÉTHODES DE MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS : LES PROCÉDURES DE MESURE

Nous nous limitons dans cette section aux procédures de mesure associées aux principales méthodes de mesure identifiées plus haut. Nous commençons par la première née des méthodes, F.P.A.

3.1 FUNCTION POINT ANALYSIS (F.P.A.)

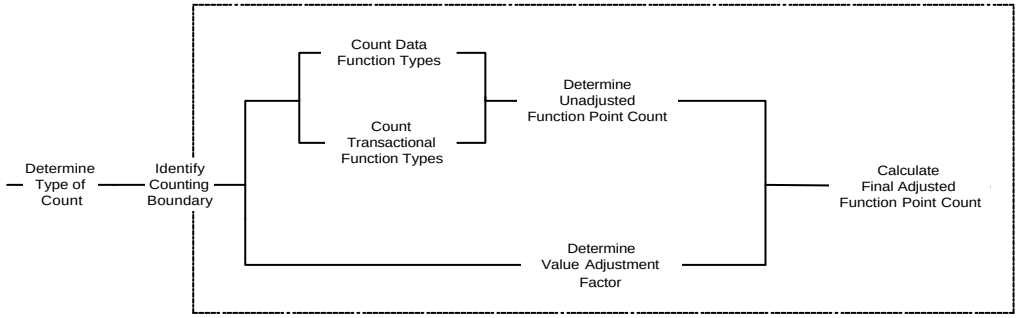


Figure 4 : Procédure de mesure FPA [IFPUG 01, ISO/IEC 03b]

La mesure de la taille fonctionnelle d'un logiciel par le biais de la méthode F.P.A. commence par la détermination du type de «comptage »/mesure (« comptage » pour projet de développement, pour projet d'amélioration/maintenance ou pour application). Le calcul final de la taille fonctionnelle en dépend (il existe une formule de calcul pour chaque type de « comptage »). L'identification de la frontière de « comptage » constitue la deuxième étape de la procédure de mesure. La frontière permet de déterminer les fonctions qui doivent être incluses dans le «comptage ». L'étape suivante consiste en la détermination du total des points de fonction non ajustés. Pour ce faire, il faut identifier et comptabiliser toutes les types de données (Data Function Types) et les types de transaction (Transactional Function Types) pour toutes les fonctions incluses dans le «comptage ». F.P.A. identifie deux (2) catégories de types de données : les fichiers logiques internes (Internal Logical Files) et les fichiers d'interface externes (External Interface Files). Quant aux types de transaction, F.P.A. les regroupe en trois (3) catégories : Entrées externes (External Inputs), sorties externes (External Outputs) et requêtes externes (External Inquiries). L'avant dernière étape de la procédure de mesure est consacrée à la détermination du facteur d'ajustement de valeur (Value Adjustment Factor). Ce facteur sera utilisé pour pondérer le total des points de fonction non ajustés déterminé à l'étape précédente, suivant des formules appropriées. Cette pondération est faite à la dernière étape de la procédure de mesure. L'on obtient alors le total des points de fonctions ajustés pour le comptage en cours.

Si les étapes une, deux et cinq nous semblent assez simples, il n'en est pas de même des étapes trois (Détermination du total des points de fonction non ajustés) et quatre (Détermination du facteur d'ajustement de valeur). Les règles régissant ces étapes [IFPUG 01, ISO/IEC 03b] font la part belle au jugement du mesureur. Dans bien des cas l'on risque d'aboutir à des interprétations différentes suivant les mesureurs. L'expérience du mesureur fait alors la différence ici. De plus ces règles semblent reposer sur l'hypothèse selon laquelle les artefacts du logiciel à mesurer sont suffisamment détaillés, ce qui n'est pas toujours le cas notamment à la phase d'analyse (par exemple à cette phase, l'on ne connaît pas nécessairement le détail complet des attributs pour un groupe de données).

3.2 MARK II FUNCTION POINT ANALYSIS (MK II FPA)

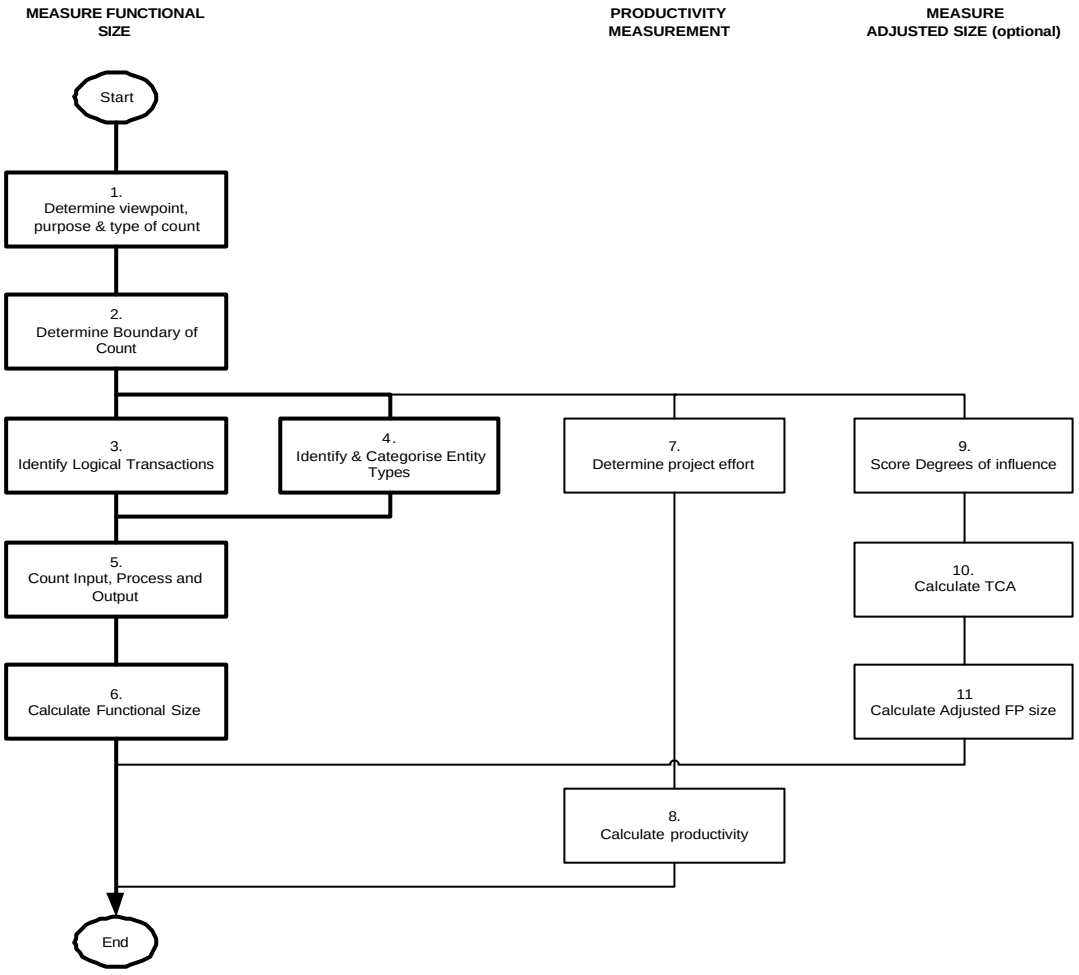


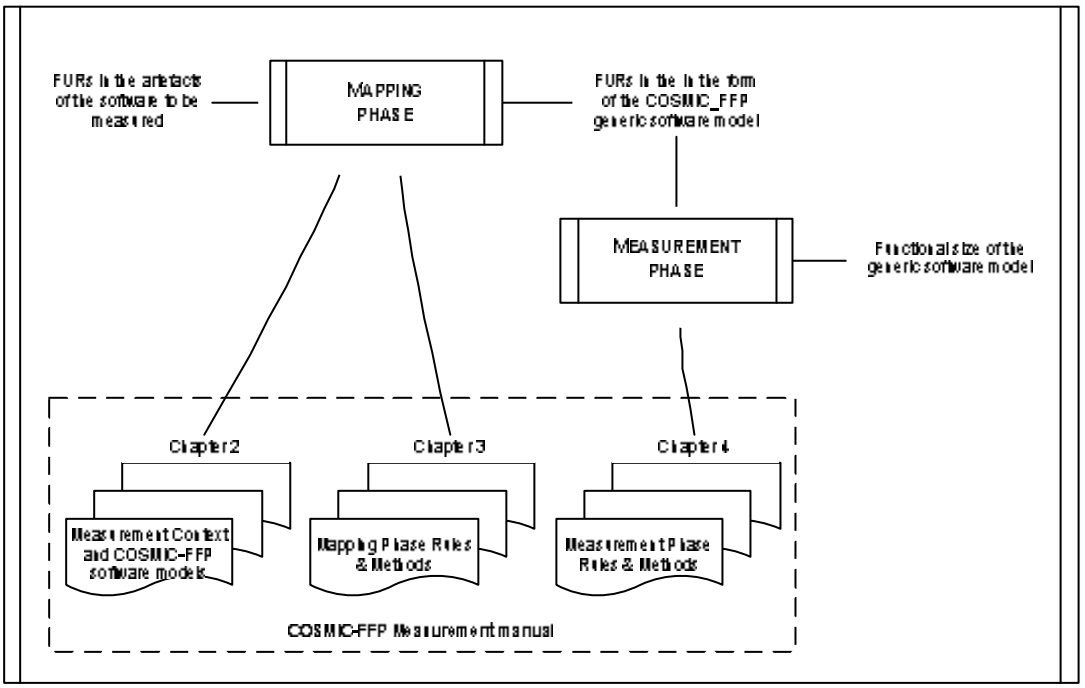
Figure 5 : Procédure de mesure MkII-FPA [UKSMA 00, ISO/IEC 02]

La mesure de la taille fonctionnelle d'un logiciel par le biais de la méthode Mk II FPA commence par la détermination de la perspective (viewpoint) et des objectifs de la mesure, du type de «comptage »/mesure (« comptage » pour projet de développement, pour projet d'amélioration/maintenance ou pour application). La perspective de la mesure précise le portfolio concerné par l'activité (travail d'un groupe de développeurs, fonctionnalités offertes à un utilisateur en particulier, ensemble des applications de comptabilité, ensemble des applications d'une entreprise, etc.). Les objectifs de la mesure (analyses de productivité, estimation de l'effort de développement pour un projet, etc.) et la perspective de mesure peuvent influencer l'identification de la frontière de «comptage », deuxième étape de la

procédure de mesure. La frontière permet de déterminer les fonctions qui doivent être incluses dans le «comptage » et celles qui doivent en être exclues. L'étape suivante de la procédure de mesure (étape 3) consiste en la détermination des transactions logiques contenues dans la frontière de «comptage ». Chaque transaction logique consiste en une entrée (Input Data Element Type), une sortie (Output Data Element Type) et un traitement (process). La détermination des éléments composants des transactions (entrée, sortie et traitement) se fait à l'étape 5 de la procédure de mesure, juste après la détermination des types d'entités contenus dans la frontière de « comptage » (étape 4 de la procédure de mesure). La dernière étape de la procédure de mesure (étape 6) est consacrée au calcul de la taille fonctionnelle suivant une formule appropriée.

Si les étapes une, deux et cinq nous semblent assez simples, il n'en est pas de même des étapes trois (Identification des transactions logiques) et quatre (Identification & Catégorisation des types d'entités). L'expérience du mesureur fait alors la différence ici.

3.3 COMMON SOFTWARE MEASUREMENT INTERNATIONAL CONSORTIUM - FULL FUNCTION POINTS (COSMIC-FFP)



La mesure de la taille fonctionnelle d'un logiciel par le biais de la méthode COSMIC-FFP comporte deux phases principales : une phase dite de « mapping » et une phase dite de « mesure ». La première phase est consacrée aux activités de mise en correspondance (« mapping ») entre les FURs (Functional User Requirements) identifiés dans les artefacts du logiciel à mesurer et les éléments du modèle générique de logiciel COSMIC-FFP (frontière, processus fonctionnel, mouvement de données, couches, etc.). L'on aboutit à une instance du modèle générique de logiciel COSMIC-FFP). Cette instance est utilisée comme entrée de la seconde phase de la procédure de mesure. Au cours de cette ultime phase, des valeurs numériques sont affectées à certains éléments de l'instance, suivant certaines règles fixées par la méthode, puis ces valeurs sont agrégées suivant une formule établie par la méthode. La valeur obtenue par agrégation correspond à la taille fonctionnelle de l'instance du modèle générique de logiciel COSMIC-FFP obtenue à l'issue de la phase de « mapping ». Cette valeur sera considérée comme la taille fonctionnelle du logiciel à mesurer. La taille ici est exprimée en *Cfsu* (Cosmic functional size units).

Il est à noter qu'on a ici une catégorisation des tâches associées à la procédure de mesure : l'on distingue les tâches dites de « mapping », des tâches dites de « mesure » (Cette catégorisation n'est pas aussi explicite pour les méthodes de mesure présentées plus haut, à savoir FPA et Mk II FPA). Ceci constitue à notre avis un plus dans la structuration de la procédure de mesure. Si les tâches de « mesure » semblent assez simples et facilement formalisables, il n'en est pas de même des tâches de « mapping » (détermination des frontières, des couches, etc.). L'expérience du mesureur pourrait faire la différence ici.

3.4 REMARQUES GENERALES

De manière générale, dans aucun des manuels de mesure pour les méthodes de mesure examinées plus haut, n'est donnée de façon explicite une vue globale des concepts et liens entre concepts que l'on retrouve dans les procédures de mesure. Aucune formalisation explicite des procédures de mesure n'est non plus disponible. A notre avis, celle-ci expliciterait davantage les procédures de mesure (meilleure compréhension) et surtout, serait d'une aide précieuse pour le développement d'outils d'aide à la mesure en général et des outils d'automatisation de la mesure en particulier. Dans cette thèse, nous proposons de compléter les manuels de mesure avec une formalisation ontologique des procédures de

mesure (les détails sont fournis au chapitre troisième). Nous pensons également que l'on pourrait généraliser dans un modèle les procédures de mesure, ce qui permettrait de construire des outils généraux d'aide à la mesure ou encore des outils généraux d'automatisation de la mesure. La procédure de mesure de COSMIC-FFP constitue à notre avis une base assez solide. Nous nous en inspirons pour proposer, dans le chapitre quatrième de cette thèse, un modèle cognitif décrivant une procédure de mesure. Ce modèle est à la base de l'approche que nous proposons, toujours dans ce chapitre, pour l'automatisation des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels. Dans la section qui suit, nous abordons la question d'automatisation de la mesure de la taille fonctionnelle des logiciels, question de mieux en saisir l'intérêt, l'état de l'art et les défis à relever.

4 L'AUTOMATISATION DE LA MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS

L'implantation des systèmes existants de mesure de la taille fonctionnelle des logiciels au niveau de l'industrie doit après la bataille administrative (acceptation des décideurs, motivation, engagement), faire face à deux problèmes majeurs d'ordre technique : la difficulté à appliquer les méthodes de mesure (ce qui rend fastidieuse la tâche du mesureur et nécessite parfois l'intervention d'un ou plusieurs experts malheureusement pas toujours disponibles) [Jones 96b], le manque ou la rareté d'outils d'assistance pour l'application des méthodes de mesure. Relativement à ce dernier point, Sue Black & David Wigg [Black et al. 99] relèvent que «l'industrie du logiciel a besoin d'outils de mesure pouvant être utilisés pour effectuer les mesures sur différentes plates-formes et pour différents langages, en vue d'offrir plus de *flexibilité* et améliorer l'*utilisabilité* ». L'idéal serait bien entendu de disposer d'outils automatisant la mesure. En tout cas, pour ce qui est de la mesure de la taille fonctionnelle des logiciels, de tels outils seraient les bienvenus. La recherche sur l'automatisation de *tout ou partie* du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels constitue à notre avis un pas assez considérable dans le sens de la facilitation (simplification) de l'application de la méthode, et pourrait contribuer à réduire de façon significative la subjectivité (due à l'interprétation des règles de mesure) bien souvent associée à l'application d'une méthode [Nishiyama 99, Abran et al. 01]. Dans cet ordre d'idées, parmi les caractéristiques désirables pour la «nouvelle génération» des

méthodes de mesure de taille des logiciels que propose le groupe COSMIC, figurent la facilité d'utilisation, la non subjectivité, la *répétabilité* (donc l'« automatisabilité ») [Symons et al. 99]. Notons que la *répétabilité* dont il est question ici fait référence au fait que deux mesureurs quelconques puissent aboutir à la 'même' taille fonctionnelle (avec une faible marge de différence) à partir des documents de spécifications du même logiciel. En effet, si cela est possible pour une méthode de mesure, alors cela suppose que la méthode est suffisamment claire et bien définie (sans ambiguïtés ou plutôt avec très peu d'ambiguïtés), et donc qu'il y a possibilité d'automatisation *complète ou partielle* de la procédure de mesure qui lui est associée.

Deux principales pistes de recherche existent à ce jour, en matière d'automatisation de la mesure de la taille fonctionnelle des logiciels. La première piste, qui prend appui sur la rétro-ingénierie des lignes de codes (dérivation de la taille fonctionnelle à partir des lignes de codes) pourrait être qualifiée d'approche « indirecte » (relativement aux fonctionnalités), tandis que la seconde, qui part des spécifications (mesure de la taille fonctionnelle à partir des documents de spécifications), pourrait être qualifiée d'approche «directe ». Un certain nombre de travaux ont été consacrés à la première approche [Edge et al. 97, Ho et al., 99, Paton 99, Edge 00], contrairement à la seconde approche pour laquelle très peu de travaux ont été consacrés [Rask 91, Gramantieri et al. 97, Diab et al. 01, Uemura et al. 01, Azzouz et al. 04]. La grande majorité des travaux sur l'automatisation du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications, pendant les vingt dernières années, ont porté sur la méthode de mesure FPA. Malheureusement toutes les tentatives d'automatisation complète du processus d'application de la méthode à partir des spécifications ont été des échecs quasi complets au niveau de l'industrie, et partiels pour ce qui est des recherches académiques [Edge 00]. Parmi les principaux défis à relever, nous soulignons la formalisation des procédures de mesure associées aux méthodes de mesure et la gestion de la nature, de l'incomplétude et de la granularité des spécifications. Relativement à l'aspect formalisation des procédures de mesure, le groupe COSMIC a entrepris ces dernières années (1999-2003) le «re-design » des méthodes de mesure de la taille fonctionnelle des logiciels, avec comme illustration concrète, la méthode de mesure COSMIC-FFP, dite de la deuxième génération des méthodes de mesure de la taille fonctionnelle des logiciels. Ce travail constitue une note d'espoir pour la question de formalisation des procédures de mesure. La présente thèse tire profit des premiers résultats

du travail et s'en veut un prolongement. De plus, la quasi-totalité des travaux ayant porté sur l'automatisation du processus d'application des méthodes de mesure de la taille fonctionnelle des logiciels à partir des spécifications se sont concentrés chacun sur une méthode de mesure en particulier et sur un langage/outil de spécifications bien déterminé (FPA - Rational Rose [Uemura et al. 01], FFP – ROOM [Diab et al. 01]...). **Nous proposons une étude plus globale de la question d'automatisation du processus d'application des méthodes de mesure de la taille fonctionnelle des logiciels à partir des spécifications, considérant les méthodes de mesure et les langages/outils de spécifications comme des paramètres pour l'étude.** Une telle étude a comme principaux défis à relever : l'analyse détaillée des procédures de mesure associées aux méthodes de mesure, l'examen de la question de formalisation de ces procédures et l'« implantabilité » dans un système logiciel des résultats de la formalisation. Cette thèse lève un pan de voile sur tous ces défis (chapitres 3 et 4).

La section suivante est consacrée à une synthèse et une analyse de haut niveau des questions de recherche auxquelles nous allons nous attaquer dans le projet.

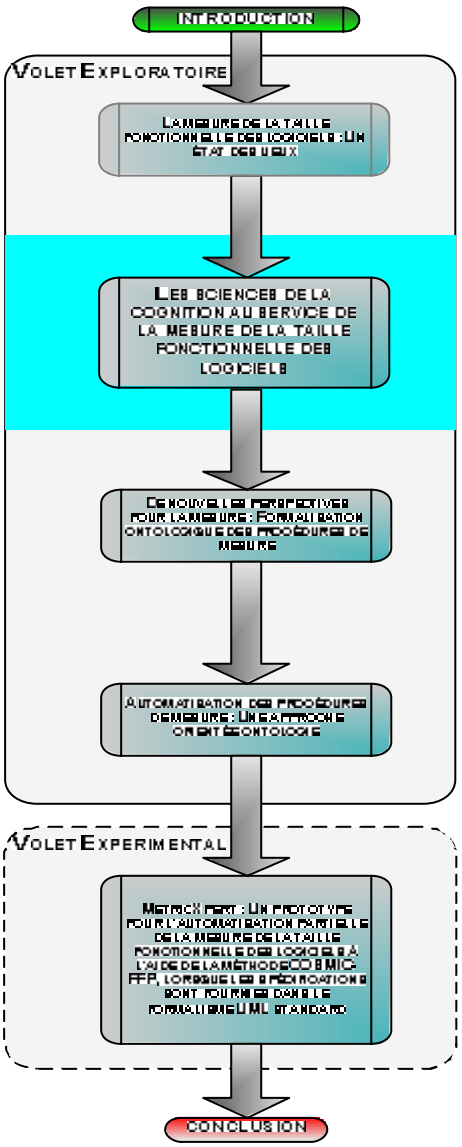
5 SYNTHÈSE

Force est de constater que la mesure de la taille fonctionnelle des logiciels est une approche de mesure en pleine expansion dans bien des pays dans le monde. Elle présente un intérêt certain dans les activités d'estimation (coût, effort) ou encore d'analyse de productivité, comparativement aux autres approches de mesure recensées (approche par lignes de codes par exemple) : elle peut être faite très tôt dans le cycle de vie du logiciel (phase d'analyse ou de conception), elle permet de mesurer la productivité économique [Jones 96b], la taille obtenue est indépendante de la technologie utilisée pour implanter les logiciels (plateforme de développement, langage de programmation, etc.), elle bénéficie d'un début de standardisation (les principales méthodes de mesure sont devenues des standards ISO). Par contre, l'application des méthodes demeure une tache non triviale. Elle requiert une expertise propre pour chacune des méthodes de mesure. De plus, l'on note une certaine subjectivité (due à l'interprétation des règles de mesure) lors de l'application d'une méthode de mesure [Nishiyama 99, Abran et al. 01]. L'automatisation complète ou partielle des procédures de mesure constitue à notre avis une piste prometteuse pour s'attaquer à une

bonne partie des problèmes auxquels fait face l'application d'une méthode de mesure de la taille fonctionnelle des logiciels aujourd'hui. Ce travail d'automatisation passe par une formalisation des procédures de mesure. Pour une meilleure plausibilité de l'automatisation (simulation du travail du mesureur) et dans la mesure où certains problèmes recensées sont de nature cognitive ou associés (interprétation des règles de mesure, mise en correspondance entre concepts de mesure et concepts de spécification, catégorisation de concepts de spécification aux fins de mesure, accessibilité et facilitation de la mesure, systématisation des procédures de mesure, recherche de consensus, etc.), une telle formalisation à notre avis tirerait profit des travaux de recherche en sciences de la cognition (représentation des connaissances, simulation, mapping/traduction, catégorisation/classification, etc.).

CHAPITRE 2.

LES SCIENCES DE LA COGNITION AU SERVICE DE LA MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS



FEUILLE DE ROUTE

1. *Introduction*
2. *Mesure et représentation des connaissances*
3. *Mesure et catégorisation/classification*
4. *Mesure et mapping/traduction*
5. *Mesure et simulation*
6. *Synthèse*

EN BREF …

Ce chapitre est consacré aux différents thèmes de recherche abordés dans les sciences de la cognition et qui pourraient trouver une oreille attentive dans le domaine de la mesure de la taille fonctionnelle des logiciels, et ouvrir de nouvelles perspectives dans la recherche de solutions à certains problèmes rencontrés dans le domaine (interprétation des règles de mesure, mise en correspondance entre concepts de mesure et concepts de spécification, catégorisation de concepts de spécification aux fins de mesure, accessibilité et facilitation de la mesure, systématisation des procédures de mesure, recherche de consensus, etc.). Les thèmes identifiés sont les suivants : la représentation des connaissances, la catégorisation/classification, la mapping/traduction et la simulation. Pour chacun de ces thèmes, le lien avec les activités de mesure est mis en exergue et la place occupée dans la thèse est précisée.

1 INTRODUCTION

Selon Keith J. Holyoak [Wilson et al. 99], la recherche en sciences de la cognition, dans des disciplines autres que la psychologie, a généralement des implications actuelles ou potentielles en psychologie. Pour ce qui est de cette thèse, la question psychologique générale à la base est la suivante : Comment l'homme évalue-t-il mentalement la taille des objets qu'il perçoit ? Plus spécifiquement, en nous rapprochant de notre domaine à savoir celui de la mesure du logiciel: Comment un mesureur (être humain) procède-t-il, à partir de spécifications à lui fournies d'un logiciel (informations), pour déterminer la taille fonctionnelle du logiciel (traitement sur les informations) ? Comme informaticien, nous relevons deux (2) autres questions sous-jacentes à cette question : (1) Dans quelle mesure peut-on construire un système informatique modélisant effectivement un mesureur (être humain) dans la tâche de mesure de la taille fonctionnelle d'un logiciel ? (2) Dans quelle mesure une modélisation, fut-elle sommaire, du mesureur (être humain) dans la tâche de mesure de la taille fonctionnelle d'un logiciel, sera-t-elle utile pour construire un système informatique (efficace, facile d'utilisation, etc.) qui mesure automatiquement ou assiste l'humain pour mesurer la taille fonctionnelle des logiciels ? Dans cette thèse, nous nous penchons uniquement sur la seconde question sous-jacente, car notre objectif principal n'est pas de mimiquer le mesureur à travers un système informatique, mais de comprendre le travail du mesureur dans le but d'améliorer la construction d'outils d'assistance a la mesure. La réponse à cette question détermine à bien des égards la pertinence d'une bonne partie de notre travail de recherche. Le chapitre quatrième de cette thèse apporte des éléments de réponse à la question. En effet dans le cadre de l'automatisation des procédures de mesure, il est question de substituer le mesureur par un programme informatique. Cela passe par une spécification explicite de la procédure de mesure qui pourrait être perçue comme un processus de conceptualisation, l'objet de la conceptualisation étant le logiciel. La particularité de ce processus est qu'il part de quelque chose d'abstrait, de conceptuel, savoir le logiciel (plus précisément les spécifications du logiciel). Ce processus est nourri par un certain nombre de « connaissances », qu'il convient de spécifier (représenter, décrire). Ainsi l'un des principaux liens existant entre cette thèse et les sciences de la cognition est relatif à la *représentation des connaissances*.

2 MESURE ET REPRÉSENTATION DES CONNAISSANCES

2.1 LA REPRÉSENTATION DES CONNAISSANCES : LE DÉBAT PHILOSOPHIQUE

Dans les débats philosophiques et relativement à la cognition, la question de la représentation des connaissances semble étroitement liée à l'une des trois (3) questions classiques[5] ayant marqué l'histoire de la philosophie, depuis les premiers débats de la Grèce antique : *La question de la nature et la structure de l'esprit et des connaissances.* Au courant du 17ème siècle, deux courants de pensée s'opposent alors sur la question : l'empirisme et le rationalisme. Pour les rationalistes (au rang desquels figurent Descartes, Leibniz, Spinoza ou encore Kant), la **raison** est la principale et même la seule source pertinente de connaissances chez l'humain. De façon spécifique, le rationalisme apparaît comme une théorie épistémologique qui met l'emphase sur l'importance des jugements, propositions, concepts, idées ou arguments ***à priori*** (indépendants de toute expérience sensorielle), comme fondements des connaissances significatives du monde. Kant postule même que les jugements synthétiques *à priori* sont des pré-conditions à toute expérience et donc constituent la base des connaissances mathématiques et scientifiques. Dans le camp empiriste (Bacon, Locke, Berkeley, Hume, Mill, etc.) l'on insiste sur l'**expérience** comme source des idées et des connaissances. L'empirisme apparaît comme une théorie épistémologique selon laquelle les connaissances sur le monde reposent sur des jugements, propositions, concepts, idées ou arguments ***à posteriori.*** Ces connaissances dérivent de nos interactions sensorielles, expérientielles ou empiriques avec le monde. Au vingtième siècle, **pragmatistes** (Peirce, James, Mead, Addams, Dewey, etc.) et **positivistes** (Hume, Russell, Wittgenstein, etc.) vont étendre et appliquer les principes de l'empirisme. L'on assiste également à l'émergence du débat entre nativistes, partisans de la modularité de l'esprit et connexionnistes.

Avec le développement dans les années 1930 de la notion de machine universelle et la formalisation de la notion de computation universelle, la question de la nature des traitements relevant de l'esprit (processus mentaux ou cognitifs entre autres) et par ricochet

[5] Relativement à la cognition, trois questions classiques peuvent être retenues comme ayant marqué l'histoire de la philosophie, depuis les premiers débats de la Grèce antique : La première question a trait à la relation qui existe entre le mental et le physique et est connue sous le vocable *problème corps-esprit*; La deuxième question est relative à la nature et la structure de l'esprit et des connaissances. Quant à la troisième question, elle aborde le problème des autres esprits, sous-entendu relativement à un esprit.

la nature de l'esprit, prend une place importante dans le débat : Les partisans de la thèse computationnelle (Warren McCulloch, Walter Pitts,...) font alors face à tous ceux qui s'y opposent. Pour les partisans de la thèse computationnelle, l'esprit est une machine de type computationnel [Fodor 75, Collins et al. 88]. En d'autres termes, l'esprit pourrait être perçu comme un «computeur », soit un système de manipulation de symboles ou de structures symboliques. Cependant, la nature et la structure du système ainsi que la nature des symboles ou structures symboliques, ne font pas toujours l'unanimité. Il existe à ce jour, un certain nombre de modèles qui prennent appui sur la computation[6] : On parle de «modèles computationnels ». Les modèles computationnels sont utilisés notamment pour la description des processus (pensée, apprentissage, perception, etc.). Ces processus étant étroitement liés aux structures mentales, la question de la formalisation desdites structures revêt un caractère essentiel. Cette préoccupation est omniprésente dans les paradigmes symbolique et subsymbolique.

Dans le paradigme symbolique, l'on considère que les structures mentales de la « psychologie ordinaire » (les buts, les croyances, les connaissances, etc.…) sont toutes « formalisables » en termes de « langage de la pensée », selon l'expression de Fodor (1975) [Fodor 75], sous forme de structures symboliques (constituées de symboles élémentaires dont chacun est susceptible d'être interprété sémantiquement par les concepts ordinaires dont nous nous servons pour conceptualiser le domaine). Sur ces structures symboliques opèrent des procédures de manipulation symboliques composées d'opérations primitives (la concaténation par exemple). Selon le paradigme symbolique, c'est à partir d'opérations de ce genre qu'il faut comprendre les processus cognitifs. Le « langage de la pensée » dont fait mention Fodor, est censé fournir une formalisation littérale de la « psychologie ordinaire ». Les règles permettant d'opérer sur ce langage sont pour l'essentiel « les lois de la pensée » de Boole (1954). Le vecteur des structures symboliques associées à ce langage est un SSP[7] (Système Symbolique Physique) sous-tendu à son tour par des niveaux d'implémentation inférieurs dans un dispositif de calcul. La genèse et le développement de *structures de connaissances* dans le SSP sont fixés à priori et suppose l'existence de prescriptions

[6] Les modèles mathématiques en sont une illustration parfaite

[7] Par *Système Symbolique Physique (SSP)*, il est entendu un système formel automatique et interprété. Un tel système doit disposer d'un ensemble de symboles (pièces ou unités), d'une position de départ et d'un ensemble de règles de manipulation des symboles; la manipulation des symboles doit être automatisée et les symboles, ainsi que les règles de manipulation, systématiquement interprétables.

normatives (interprétées par certains comme des « démons » ou « homoncules »). En général, dans un SSP, l'encodage d'une nouvelle *connaissance* revient par exemple à produire une nouvelle structure symbolique en combinant les symboles élémentaires d'un alphabet. Le niveau symbolique qui implémente les structures de connaissance est censé être exact et complet. Cela signifie que les niveaux inférieurs ne sont pas nécessaires pour fournir une description exacte de la cognition en termes d'éléments interprétables sémantiquement. Malheureusement, comme le souligne Paul Smolensky [Poirier et al. 01], du point de vue des neurosciences, le paradigme symbolique n'a pas permis de comprendre grand chose d'utile à l'organisation computationnelle du cerveau. Mais d'une certaine manière, cette remarque pourrait être perçue comme juste mais non pertinente, dans la mesure où la théorie symbolique n'est pas une théorie du cerveau mais une théorie du fonctionnement de l'esprit. Du point de vue de la modélisation du comportement le paradigme semble n'avoir donné satisfaction qu'à un niveau grossier. Du point de vue de l'IA, les règles symboliques et la logique qui permet de les manipuler tendent à produire des systèmes rigides et fragiles. En fait, quelques problèmes demeurent toujours sans solutions : (1) les êtres humains sont capables de générer de nouvelles primitives lorsqu'ils sont confrontés à de nouveaux objets; cette capacité et les problèmes soulevés par un alphabet soustrait à l'apprentissage, représente un problème majeur pour la modélisation symbolique (Schyns et Murphy, 1993); (2) la nature essentiellement symbolique et sérielle du traitement de l'information dans ce type d'approche rend les modèles particulièrement sujets à l'explosion combinatoire et donc difficilement applicables en dehors des univers et micro-mondes dans lesquels ils ont été définis (Robert Proulx, 1994 [Rialle et al. 94]); (3) dans un système symbolique, les symboles, en dépit de leur interprétabilité systématique, ne sont pas ancrés; leurs significations sont parasitaires de l'esprit d'un interprète (Stevan Harnad, 1994 [Gire 97]). (4) l'information sensorielle sur le monde physique est toujours supposée numérique (intensités lumineuses, forces, fréquences,…). Par conséquent, il devrait exister au moins une couche de computation non symbolique entre le monde réel et le paradigme des symboles purs; (5) la prise en compte de l'incertitude n'est pas chose évidente.

Dans le paradigme subsymbolique, il y est postulé que la formalisation de la cognition procède par abstraction à partir des structures neuronales. Ainsi, penser revient à « computer » comme le fait un réseau, c'est-à-dire massivement parallèle, « les comportement intéressants n'apparaissant qu'au niveau collectif, en émergeant du système

des interactions entre computeurs élémentaires simples » DUPUY (1995)(p.60) [Gordon et al. 97]. Dans le paradigme subsymbolique, les structures neuronales constituent la base (en un sens suffisamment abstrait du terme) du formalisme utilisé pour la description de l'intelligence, tandis que les structures mentales n'interviennent que dans les descriptions approximatives. Ici on ne conçoit pas les niveaux de la cognition par analogie avec les niveaux de systèmes informatiques : la cognition doit être conçue comme ayant lieu dans des systèmes dynamiques et non dans des ordinateurs numériques. De tels systèmes pourraient être perçus comme des ensembles d'unités élémentaires interconnectées, avec généralement une collection de variables dynamiques : Certaines correspondent au niveau d'activation pour chaque unité et les autres correspondent à la force de connexion pour chaque liaison. La relation entre le formalisme subsymbolique et le traitement psychologique est en partie déterminée par les constantes de temps qui interviennent dans les équations différentielles qui gouvernent l'activation et la modification de la force des connexions. Toute la connaissance contenue dans les connexions est utilisée en parallèle tandis qu'à l'échelle des processus cognitifs, comme la résolution de problème et le raisonnement non immédiat, on a affaire à un traitement séquentiel et même éventuellement à de l'inférence logique. D'après Paul Smolensky [Poirier et al. 01], en considérant les entités mentales comme des structures de niveau supérieur implémentées dans les systèmes connexionnistes, on obtient une vision nouvelle, plus complexe de ce que sont réellement ces structures mentales. Cependant quelques problèmes demeurent sans solution : (1) le modèle connexionniste dans son ensemble est dans l'incapacité de réaliser les propriétés formelles de systématicité, de composabilité, de productivité et d'inférencialité des systèmes computo-symboliques (Fodor & Pylyshyn); (2) il est difficile d'envisager un moyen de modéliser les niveaux supérieurs de la cognition. Pour remédier à cela, certains ont proposé des solutions qui fournissent une implantation neurale plausible des modèles symboliques de la cognition; (3) la modélisation des comportements qui s'étendent dans le temps (nécessite l'utilisation de «réseaux récurrents » (très peu maîtrisés pour le moment) pour mémoriser les états internes du réseau à chaque instant).

Au total, les descriptions de haut niveau des systèmes subsymboliques sont approximativement équivalentes aux explications symboliques, même si certains auteurs tel que Paul Smolensky, estiment que les concepts symboliques de la psychologie ordinaire fournissent une description approximativement exacte des phénomènes cognitifs, tandis que

les conceptions subsymboliques en fournissent une description plus précise. En fait, la formalisation de la cognition dans le paradigme subsymbolique procède par abstraction à partir des structures neuronales, plutôt qu'en opérant par formalisation symbolique des structures mentales comme c'est le cas dans le paradigme symbolique. En définitive, les paradigmes symboliques et connexionnistes semblent non antagonistes mais complémentaires. Les modèles connexionnistes pourraient gérer la cognition au niveau inférieur (exemple : reconnaissance de motifs visuels, reconnaissance de la parole,…) et devraient fournir un substrat pour les processus symboliques au niveau supérieur. A partir du début des années 80, les chercheurs en Intelligence Artificielle, Analyse de Décision, et Statistiques ont développé ce qui est connu sous le vocable «Réseaux Bayesiens » (Pearl 1988). Ces réseaux offrent une structure pour les bases de connaissances probabilistes par l'expression de relations d'indépendance conditionnelles entre les variables (la théorie de probabilité est maintenant largement acceptée comme base de calcul pour le raisonnement sous incertitude). Dans la plupart des cas, le nombre de probabilités qui doivent être spécifiées dans un réseau Bayesien croît seulement de façon linéaire par rapport au nombre de variables. De tels systèmes peuvent alors gérer des problèmes assez importants en taille et les applications sont nombreuses. De plus, des méthodes existent pour l'apprentissage d'un réseau Bayesien à partir de données de base, faisant de ces réseaux un pont naturel entre les approches symboliques et réseaux de neurones en Intelligence Artificielle. Des questions demeurent, notamment celle de savoir si c'est un bon modèle pour le raisonnement humain. De plus le problème de complexité (rencontré dès les tous débuts de l'utilisation du raisonnement probabiliste) n'est pas à négliger.

Une réflexion nous paraît assez pertinente par rapport au débat actuel sur la question du lien entre la computation et l'esprit. Il s'agit d'une réflexion de Vincent Rialle que nous partageons [Gordon et al. 97] : « L'esprit est-il computationnel ? En clair : le cerveau, si l'on admet l'indissociabilité corps-cerveau-esprit, fonctionne-t-il dans ses diverses activités comme une machine de traitement de symboles ? Si la réponse est positive, deux conséquences s'imposent alors : d'une part, on aura réussi à percer l'énigme de l'apparente dualité corps-esprit, qui est aujourd'hui l'un des points les plus discutés en philosophie de l'esprit; d'autre part, on saura construire une machine intelligente. Si *a contrario* la sphère

cognitive n'était pas computationnelle, serait-il encore possible d'en rendre compte 'scientifiquement'[8] ? »

En attendant d'avoir une réponse claire à ces questions, les travaux de recherche en Intelligence Artificielle (IA) relativement à la représentation des connaissances s'inspirent fortement des paradigmes présentés plus haut. Nous nous en rendrons compte dans la section suivante consacrée à la question de la représentation des connaissances en lien avec notre projet de recherche.

2.2 LA REPRÉSENTATION DES CONNAISSANCES EN IA ET DANS LE DOMAINE DE LA MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS

Comme le fait remarquer Jean-Guy Meunier [Meunier 92], « Les systèmes informatiques dits intelligents possèdent entre autres une base de connaissances à laquelle ils se réfèrent pour intégrer les intrants nouveaux, prendre des décisions et enfin effectuer des raisonnements, etc. ». La base de connaissances constitue pour ainsi dire un maillon essentiel dans un système informatique dit intelligent. Les connaissances qui y sont contenues sont présentées dans un formalisme bien précis et suivant des règles établies. Dans l'un de leurs multiples textes, Newell & Simon [Newell et al. 76], relèvent que toute activité intelligente (de la part d'un humain ou même d'une machine) nécessite les éléments suivants : La représentation des aspects significatifs du domaine d'un problème (on pourrait aussi parler de représentation des connaissances pertinentes associées au domaine d'un problème), la définition (ou la description) des opérations applicables sur les représentations obtenues afin de générer les potentielles solutions au problème, et enfin des stratégies de sélection d'une solution parmi les solutions potentielles générées. De par la complexité des problèmes qu'ils sont appelés à résoudre, les systèmes d'IA ont besoin non seulement de connaissances (relatives aux domaines des problèmes et autres), mais également de mécanismes et stratégies pour manipuler efficacement ces connaissances afin de fournir des solutions aux problèmes. ***Une bonne représentation des connaissances à manipuler est un atout majeur relativement à la performance et à l'efficacité des systèmes.*** G. F. Luger [Luger 02] place d'ailleurs la représentation et la fouille des connaissances au cœur de la recherche moderne en IA. Ici (en IA), les approches de représentation de connaissances

[8] Selon le principe de la pensée rationnelle du fonctionnement de l'esprit

varient entre autre en fonction du type de connaissances à représenter. Nous examinons dans la section suivante les principales catégories de connaissances identifiées en IA.

2.2.1 LES CATÉGORIES DE CONNAISSANCES RECENSEES EN IA

L'on pourrait regrouper l'ensemble des connaissances manipulées par un agent cognitif (artificiel notamment) en trois grandes catégories : Les connaissances factuelles/déclaratives, les connaissances « structurales » ou relationnelles et les connaissances procédurales.

Dans la catégorie des *connaissances factuelles,* l'on regroupe toutes les connaissances associées aux faits, qu'il s'agisse des faits[9] eux-mêmes (on parle aussi de données) ou encore des représentations des faits. On parle aussi de connaissances *déclaratives*. Dans les systèmes informatiques les connaissances factuelles sont utilisées comme paramètres pour les traitements effectués par le système (tuples dans une table de base de données, valeurs d'un champ d'une table de base de données, «fait »[10] dans une base de connaissances, événement dans un système temps-réel, etc.).

Pour ce qui est de la catégorie des *connaissances « structurales » ou relationnelles,* l'on regroupe ici tout ce qui est lien entre connaissances factuelles. Dans les systèmes informatiques, les connaissances «structurales » ou relationnelles sont nécessaires pour établir un lien entre des connaissances factuelles manipulées lors des traitements (ce sont des ensembles de propriétés décrivant des objets ou des concepts, des liens entre objets du monde réel ou entre concepts, etc.).

Quant à la catégorie des *connaissances procédurales, elles* regroupent toutes les connaissances relatives à la manipulation des connaissances factuelles et/ou « structurales ». On parle encore de *savoir-faire* ou de *connaissances dynamiques.* À ce niveau l'on pourrait identifier deux (2) sous-catégories : Celle regroupant les connaissances opérationnelles et celle regroupant les connaissances dites « inférentielles ». Dans les systèmes informatiques, les connaissances opérationnelles définissent les traitements à effectuer sur les connaissances factuelles (algorithmes, procédures, opérations dans un diagramme de classes,

[9] Lorsque j'insère dans un système la valeur '1' pour dire sexe masculin, '1' représente pour le système un fait. Mais pour moi, '1' n'est que la représentation du fait 'J'ai affaire à un individu de sexe masculin'

[10] Exemple : Homme(Pierre); Femme(Jeanne); cependant : spotted(x, y) -> saw(x, y) est plutôt une connaissance « inférentielle »

etc.). Quant aux connaissances «inférentielles », elles sont utilisées pour inférer sur les connaissances factuelles (règles de production, heuristiques, etc.).

Dans le domaine de la mesure de la taille fonctionnelle des logiciels, l'on retrouve toutes les catégories de connaissances recensées ci-dessus.

2.2.2 LES CATÉGORIES DE CONNAISSANCES IDENTIFIEES DANS LE DOMAINE DE LA MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS

Dans le domaine de la mesure de la taille fonctionnelle des logiciels et relativement aux procédures et méthodes associées, les concepts de mesure ou ceux associées à un langage de spécification de logiciel (par exemple UML), ainsi que les liens entre concepts, se rangent dans la catégorie des *connaissances « structurales » ou relationnelles.* Les tâches de mesure, les règles/procédures d'identification des instances de concepts de mesure ou encore les règles de mesure (règles d'assignation numérique et agrégation, règles contextuelles …) font partie de la catégorie des *connaissances dites « procédurales ».* Les intrants et les extrants des procédures de mesure se retrouvent quant à eux dans la catégorie des *connaissances dites «factuelles »* (modèle d'un logiciel à mesurer, spécifications d'un logiciel, résultats de mesure …).

Toutes les connaissances recensées et reliées aux procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels, ne sont pas toujours explicites. Dans certains cas elles sont tacites. C'est le cas de certaines règles de mesure (par exemple les règles/procédures d'identification des instances de concepts de mesure dans les spécifications d'un logiciel, notamment lorsque la qualité de ces spécifications laisse à désirer ou encore lorsque le langage de spécifications laisse libre court aux ambiguïtés). Ces connaissances sont dans bien des cas intuitives ou alors prennent racine dans les années d'expérience de mesure (heuristiques). Elles sont difficiles à formaliser ou à partager à travers l'espace et le temps, et donc vulnérables aux pertes. L'un des défis relevés par cette thèse est l'explicitation et la formalisation de certaines connaissances reliées aux procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels. L'objectif à terme étant de réduire considérablement (à défaut d'éliminer) les connaissances tacites reliées aux procédures de mesure.

De plus, il s'agit alors de trouver une façon appropriée, adéquate de représenter chacune des catégories de connaissances recensées, dans la perspective non seulement de

l'automatisation de la mesure, mais également d'une meilleure compréhension des méthodes de mesure. Dans la thèse, les approches de représentation de connaissances proposées en IA (Intelligence Artificielle) ont été mises à contribution.

2.2.3 LES PRINCIPALES APPROCHES POUR LA REPRÉSENTATION DES CONNAISSANCES EN IA

En matière de représentation des connaissances en IA, un certain nombre de défis sont à relever, selon la catégorie des connaissances représentées : Dans le cas des *connaissances « factuelles »*, il faut trouver un formalisme simple, expressif et accessible, qui tient compte des manipulations prévues sur les connaissances. Dans le cas des *connaissances « structurales » ou relationnelles*, en plus de certains défis recensés pour les *connaissances « factuelles »* (simplicité, expressivité, accessibilité), l'identification de toutes les propriétés pertinentes pour la description d'un objet/concept, l'identification des liens pertinents entre les propriétés identifiées et/ou entre les objets/concepts, la représentation des ensembles d'objets, la représentation des séquences d'états, le choix de la granularité de la représentation, constituent les défis majeurs. Pour ce qui est des *connaissances « procédurales »*, le principal défi est celui de l'efficacité des traitements effectués ou des inférences sur les connaissances; cette efficacité est étroitement liée à l'efficacité dans la localisation des connaissances nécessaires pour un traitement ou une inférence.

À ce jour, il existe une bonne brochette d'approches pour la représentation des connaissances en IA. L'on pourrait les regrouper sous deux (2) grands paradigmes : Le paradigme symbolique, le paradigme subsymbolique (ou connexionniste). En général, le choix du paradigme et de la forme de représentation pour un système donné, va dépendre du domaine d'application associé au système.

2.2.3.1 LE PARADIGME SYMBOLIQUE

Le point commun entre toutes les approches dans ce paradigme est l'utilisation d'un système de symboles pour représenter les connaissances, ainsi que la formalisation de la manipulation des symboles. Elles ont pour base philosophique l'Hypothèse du Système Symbolique Physique énoncée par Newell et Simon (1972). Elles pourraient être regroupées en deux (2) catégories d'approches de représentation : Les approches « orientées syntaxe » et les approches « orientées sémantiques ».

A. Approches « orientées syntaxe »

Dans cette catégorie, on retrouve principalement : Les approches basées sur la logique propositionnelle, les approches basées sur la logique des prédicats, les approches basées sur une logique pour raisonnement non monotone et les approches basées sur la logique floue.

A.1 Les approches basées sur la logique propositionnelle

Les *approches basées sur la logique propositionnelle* mettent l'emphase sur les faits ou liens entre faits du monde réel. Ces faits ou liens entre faits sont représentés à l'aide de propositions logiques (formules bien formées ou encore *Well-formed formulas*). Les liens sont établis grâce à des opérateurs logiques ($\wedge, \vee, \rightarrow, \neg$).

Illustration :

Soient les faits suivants : *« il pleut », « il fait soleil », « s'il pleut, alors il ne fait pas soleil »*

Ces faits seront respectivement représentés par les propositions logiques ci-après : *PLUVIEUX, ENSOLEILLÉ et PLUVIEUX ® Ø ENSOLEILLÉ*

Malgré leur simplicité, ces approches se heurtent aux limitations de la logique propositionnelle (certaines connaissances complexes ne peuvent pas être représentées efficacement : par exemple la capacité de généralisation est limitée). Pour pallier à ces insuffisances l'on fait appel à la logique des prédicats.

A.2 Les approches basées sur la logique des prédicats

Les *approches basées sur la logique des prédicats* font appel à ce qu'ils appellent des *prédicats* pour représenter les faits ou des liens entre faits. L'utilisation des quantificateurs (universel et existentiel) et des variables permet de généraliser (i.e. faire de la généralisation). En fait il, s'agit d'une extension des approches précédentes (approches basées sur la logique propositionnelle).

Illustration :

Soient les faits suivants : *« Socrates est un humain », « Socrates est mortel », « tout humain est mortel »*

Ces faits seront respectivement représentés par les propositions logiques ci-après : *HUMAIN(SOCRATES), MORTEL(SOCRATES), "x : HUMAIN(x) ® MORTEL(x)*

Contrairement à la logique propositionnelle, la logique des prédicats se heurte au problème de la *non décidabilité* (*il existe une procédure [qui se termine] pour trouver la*

preuve d'un énoncé, dans le cas où cet énoncé est un théorème; par contre on ne saurait garantir la terminaison d'une telle procédure dans le cas où l'énoncé n'est pas un théorème). D'autre part avec la logique des prédicats ou même la logique propositionnelle, on aboutit à des systèmes déductifs (qui déduisent de nouveaux énoncés à partir d'énoncés existants), pas inductifs (ils sont incapables par exemple de généraliser à partir d'un échantillon), ce qui ne correspond pas exactement au type de raisonnement qu'on retrouve chez l'humain. De plus, la question de la représentation des incertitudes et inconsistances demeure sans réponse adéquate. Avec une logique pour raisonnement non monotone (logique modale, logique temporelle, etc.), on va s'attaquer à certaines de ces limitations, notamment les dernières citées.

A.3 Les approches basées sur une logique pour raisonnement non monotone

Les *approches basées sur une logique pour raisonnement non monotone* sont généralement, des extensions d'approches basées sur la logique des prédicats, dans le but de tenir compte dans les représentations et dans le raisonnement, des connaissances incertaines, incomplètes ou évolutives [*Nonmonotonic Logic* (McDemott and Doyle, 80), *Default Logic* (Reiter, 80), *Autoepistemic Logic* (Moore, 85), etc.]. Par exemple dans le cas *Nonmonotonic Logic*, il est question d'une extension de la logique des prédicats du premier ordre avec un opérateur modal M (qui se lit « est consistant »). Le mode de raisonnement ici est le *raisonnement par défaut* (tirer des conclusions à partir de ce qui est le plus plausible).

> ***Illustration (Nonmonotonic Logic) :***
>
> La formule suivante : *"x, y : Entretient_Rapports(x, y) Ù M S'entend_Avec(x, y)* ➔ *Défendra(x, y)*, se lira : « Pour tout x et y, si x et y entretiennent des rapports et si le fait que *x s'entend avec y* est consistant avec toute autre croyance, alors déduire que x défendra y ».

En dépit du fait que cette approche permet de représenter un plus grand nombre de connaissances (relativement aux approches basées sur la logique propositionnelle ou la logique des prédicats), elle s'avère peu efficace sur le plan computationnel, et se heurte aussi au problème de la *non décidabilité* [Rich et al 91]. De plus, tout comme en logique classique, des concepts tels que "*grand*" ou "*petit*" ou "*un peu chaud*" ne peuvent pas être représentés adéquatement. Les approches basées sur la logique floue s'attaque entre autres à cette catégorie de connaissances.

A.4 Les approches basées sur la logique floue

Les *approches basées sur la logique floue* prennent appui sur la notion d'*ensemble flou*, avec une modification de la fonction classique d'appartenance à un ensemble. Un ensemble flou A est un ensemble de paires {(x, mA(x))}, où mA est une fonction d'appartenance dont la valeur est comprise entre 0 et 1 (indiquant une sorte de « degré d'appartenance »), plutôt qu'une valeur booléenne comme c'est le cas avec la fonction classique.

Illustration :

Soit A un ensemble flou représentant la notion d'*individu de grande taille*,
alors on aurait A = {(x, mA(x))} où x représente la taille d'un individu, mA est une fonction de d'appartenance, mA(x) indiquant le «degré d'appartenance » de x à A (par exemple : mA(1,50)=0; m(1,60)=0,4; m(1,70)=0,7; m(1,80)= 0,9 ...);

Grâce à la logique floue, il est possible de représenter certaines connaissances incertaines, en introduisant notamment la notion de «degré d'appartenance », obtenue à l'aide d'une fonction d'appartenance qui opère une distribution de possibilités (telle que définie par Lotfi Zadeh (1983) dans sa *théorie des possibilités*). Plutôt que de mesurer l'aléatoire (l'aspect aléatoire de l'information), comme c'est le cas dans la théorie des probabilités, Zadeh suggère de mesurer l'imprécision, le flou (qui entoure l'information). La théorie des ensembles flous qu'il propose inclut des règles pour la combinaison des mesures de possibilités lorsqu'on a affaire à une expression contenant plusieurs variables floues. Les systèmes à base de logique floue sont spécialement efficaces et robustes face à l'imprécision des mesures, notamment dans le domaine du contrôle en ingénierie.

Dans la section suivante nous nous intéressons à la deuxième catégorie d'approches dans le paradigme symbolique : Les approches « orientées sémantiques ».

B. Approches « orientées sémantique »

Ici, on sacrifie la rigueur (mais également la rigidité) du raisonnement logique au profit d'une simulation psychologique et linguistique inspirée du modèle humain. L'*associationnisme* est utilisé pour essayer de juguler la complexité de certains concepts. C'est ce que l'on va retrouver dans les approches basées sur les réseaux sémantiques, les approches basées sur les graphes conceptuels, ou encore les approches basées sur les réseaux Bayesiens.

B.1 Les approches basées sur les réseaux sémantiques (frames)

Les théories associationnistes sur la signification des objets constituent la base théorique pour les *approches basées sur les réseaux sémantiques (frames)* : Les associationnistes définissent la signification d'un objet en termes de réseaux d'associations (ou liens) avec d'autres objets. Ces associations (ou liens) contribuent à la formation d'une compréhension des propriétés et comportements de l'objet considéré.

Illustration :

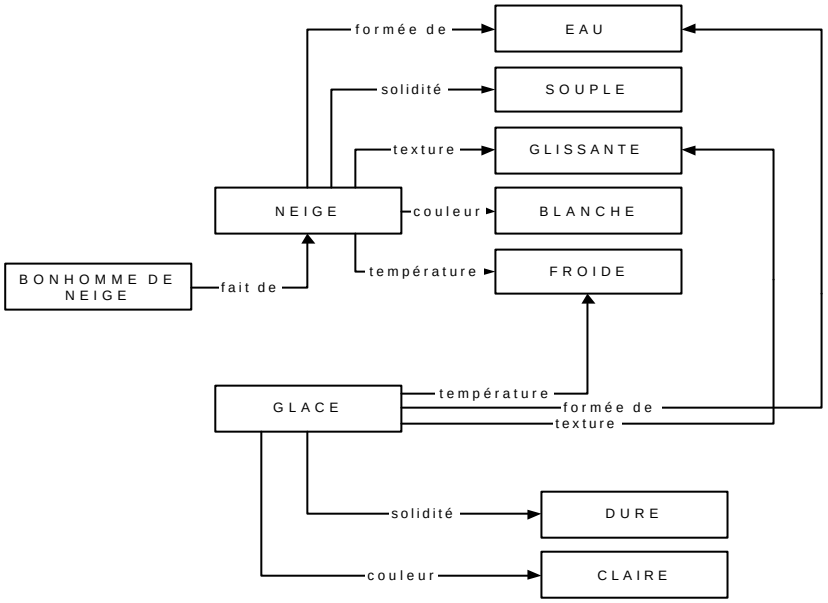


Figure 7 : Réseau sémantique représentant des propriétés de la neige et de la glace

La Figure 7 traduit le fait qu'avec l'expérience, nous associons le concept de *neige* avec d'autres concepts tels que : *froid, blanc, bonhomme de neige, glace*, etc. Notre compréhension du concept de *neige* et de la valeur de vérité des propositions telles «*la neige est blanche* », se manifeste à travers ce réseau d'associations.

Dans la mesure où ils permettent de représenter explicitement les associations (ou liens) grâce aux concepts d'*arc* et de *nœud*, les graphes vont s'avérer particulièrement appropriés pour formaliser les théories associationnistes relativement aux connaissances. Ainsi, un réseau sémantique va représenter la connaissance sous forme de graphe, les nœuds correspondant aux faits ou concepts, tandis que les arcs renvoient aux associations ou liens entre concepts (ou faits). Dans un tel réseau, l'inférence se fait en suivant les liens entre concepts. Les premiers systèmes implantant les réseaux sémantiques datent des années

1960 : Masterman (1961), Quinllian (1967), Wilks (1972), etc.; ces systèmes sont pour la plupart orientés vers la linguistique (traduction machine, traitement du langage naturel, etc.). Dans leur forme basique, les réseaux sémantiques sont simplement des collections de nœuds interconnectés : Les connaissances représentées souffrent du manque d'organisation, de structuration. Les *frames* vont étendre les réseaux sémantiques et essayer de pallier entre autre à cette insuffisance, permettant ainsi de représenter un objet assez complexe comme une unique entité (frame), plutôt qu'un réseau complexe de propriétés. Une autre extension ramenant les associations (ou liens) au niveau des nœuds plutôt que sur les arcs aboutira aux *graphes conceptuels*.

B.2 Les approches basées sur les graphes conceptuels

Dans les *approches basées sur les graphes conceptuels*, un graphe conceptuel désigne un graphe fini, connecté, bipartite (les nœuds du graphe sont soit des concepts, soit des relations conceptuelles [associations]). Contrairement aux réseaux sémantiques, les arcs du graphe ne sont plus étiquetés et ne représentent plus les associations (ou liens) entre concepts (ou faits). Il n'existe d'arc qu'entre un concept et une relation conceptuelle. Les concepts peuvent être concrets (chat, téléphone, restaurant, etc.) ou abstraits (amour, beauté, performance, etc.). Une même relation conceptuelle peut être reliée à un ou plusieurs concepts.

Illustration :

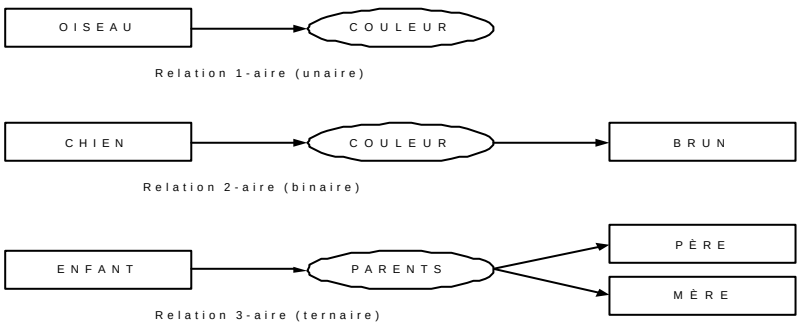


Figure 8 : Relation conceptuelles d'arités différentes

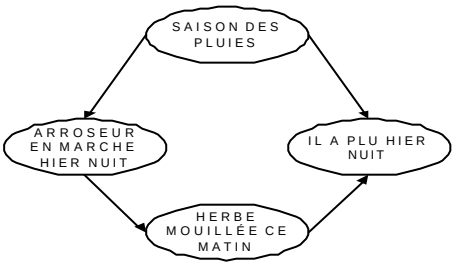
Les graphes conceptuels permettent de représenter les notions de *généralisation* et *spécialisation* chères aux approches orientées objet (les diagrammes de classe, ou encore les diagrammes entité/association sont des formes très populaires de graphes conceptuels utilisées en informatiques). Ils peuvent être transformés en expression logique : Certains auteurs (Sowa, 1984) ont d'ailleurs proposé des algorithmes pour la transformation d'un

graphe conceptuel en expression du calcul de prédicats. Cependant la question de représentation de connaissances incertaines ou incomplètes reste encore sans réponse adéquate. L'approche basée sur les réseaux Bayesiens s'attaque à cette question.

B.3 Les approches basées sur les réseaux Bayesiens

Dans les *approches basées sur les réseaux Bayesiens,* l'option dite *stochastique* pour s'attaquer à la question des connaissances incertaines est mise en avant : on raisonne sur la fréquence des événements, en prenant appui sur le théorème de Bayes. On utilise des DAG (Direct Acyclic Graph) pour la représentation. Dans les graphes, sont représentées les relations causales entre les variables du graphe (les variables du graphe pourraient être des propositions, ou des concepts). Pour chaque valeur d'un nœud parent du graphe, il faut spécifier, à l'aide de probabilités conditionnelles, les croyances fournies à propos des valeurs que le nœud fils est susceptible de prendre.

Illustration :



Attribut	Probabilité
p(HerbeMouilleeCeMatin | ArroseurEnMarcheHierNuit, IlAPluHierNuit)	0.95
p(HerbeMouilleeCeMatin | ArroseurEnMarcheHierNuit, Not IlAPluHierNuit)	0.9
p(HerbeMouilleeCeMatin | Not ArroseurEnMarcheHierNuit, IlAPluHierNuit)	0.8
p(HerbeMouilleeCeMatin | Not ArroseurEnMarcheHierNuit, Not IlAPluHierNuit)	0.1
p(ArroseurEnMarcheHierNuit | SaisonDesPluies)	0.0
p(ArroseurEnMarcheHierNuit | Not SaisonDesPluies)	1.0
p(IlAPluHierNuit | SaisonDesPluies)	0.9
p(IlAPluHierNuit | Not SaisonDesPluies)	0.1
P(SaisonDesPluies)	0.5

Figure 9 : Exemple de réseau Bayesien & Probabilités conditionnelles associées

Ces approches sont fréquemment utilisées dans des domaines pour lesquels on a besoin d'inférence abductive, ou encore ceux associés à l'apprentissage du langage naturel, la classification, etc.

Toutes les approches présentées jusqu'à présent sont basées sur une architecture centralisée et des traitements sériels ou séquentiels. De plus, la syntaxe occupe dans bien des cas une place très importante (on leur reproche bien souvent le manque d'ancrage des symboles manipulés). Nous examinons dans la suite des approches basées sur une architecture distribuée et des traitements parallèles.

2.2.3.2 LE PARADIGME SUBSYMBOLIQUE

Le paradigme subsymbolique, prend appui sur les modèles connexionnistes, et se propose de modéliser à la fois les processus perceptifs de bas niveau et les processus de haut niveau tels que la reconnaissance d'objets, la résolution de problèmes, la planification et la compréhension du langage. La formalisation de la cognition dans le paradigme subsymbolique procède par abstraction à partir des structures neuronales.

Ici, l'on identifie fondamentalement une structure à deux couches pour la représentation des connaissances : Une couche de description du système formel au niveau inférieur et une couche d'interprétation sémantique au niveau supérieur. Les entités susceptibles de recevoir une interprétation sémantique sont des «configurations d'activité » sur un grand nombre d'unités du système, et les entités manipulées par des règles formelles sont l'ensemble des activations individuelles des cellules du réseau (les règles portent sur la propagation de l'activation ou la modification de la force des connexions; elles sont donc de nature profondément différentes de règles de manipulation symbolique).

Cette approche suscite beaucoup d'intérêt aujourd'hui dans la recherche en général et la recherche en IA en particulier : L'on s'accorde à reconnaître aux modèles connexionnistes une puissance computationnelle et une efficacité certaines dans la résolution de certains problèmes étiquetés « complexes ».

L'existence d'une pléiade d'approches pour représenter les connaissances en IA pose le problème du choix de l'approche adéquate pour un domaine spécifié. Dans notre projet de recherche, nous avons opté pour une combinaison entre les approches « orientées sémantiques » (approche par graphes conceptuels) et une approche permettant d'exprimer l'incertain/imprécis (approche basée sur la théorie de la certitude de Stanford). Notre choix trouve sa justification dans l'approche orientée ontologie pour laquelle nous avons optée dans le projet : Nous mettons un point d'honneur sur l'expressivité des modèles produits. L'aspect raisonnement incertain/imprécis sera développé dans nos travaux futurs; nous en jetons uniquement les bases dans la thèse.

2.2.4 LA REPRÉSENTATION DES CONNAISSANCES DANS LE DOMAINE DE LA MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS

Tel que mentionné plus haut, dans cette thèse nous jetons les bases de la représentation des connaissances dans le domaine de la mesure de la taille fonctionnelle des logiciels. Nous

nous inspirons fortement des travaux de recherche menés en Intelligence Artificielle (IA) en matière de représentation des connaissances. Le corpus de connaissances associé au processus d'application de chaque méthode de mesure de la taille fonctionnelle des logiciels, dont les bases sont jetées dans cette thèse (à travers les ontologies), englobe aussi bien les *connaissances « structurales » ou relationnelles*, que les *connaissances « procédurales »* et les *connaissances « factuelles »* relatives au processus d'application de chaque méthode de mesure de la taille fonctionnelle des logiciels. L'approche et le formalisme adoptés pour la représentation de ces connaissances sont d'une importance certaine, compte tenu de l'approche ontologique que nous préconisons. Si les approches formelles (basées sur la logique - "orientées syntaxes") semblent appropriées pour l'automatisation (car se prêtent mieux aux inférences), l'on reconnaît aux approches à base de graphes (« orientées sémantiques ») une bonne expressivité, une simplicité et une accessibilité non négligeables. En général, il s'agit de trouver un compromis entre expressivité, simplicité, accessibilité et efficacité eu égard aux traitements. Nous avons opté pour une approche à base de graphes (approche basée sur les graphes conceptuels) et le formalisme orienté-objet UML (diagramme de classes, diagramme de composants). La simplicité, l'expressivité, l'accessibilité et la popularité du langage UML sont mises à profit. Une mise à contribution de la *théorie de la certitude de Stanford* est envisagée dans les prochaines itérations, pour prendre en compte l'imprécision et/ou l'incertitude associées à certains intrants des tâches de mesure (en l'occurrence les spécifications). Cette théorie émet des hypothèses pour créer des ***mesures de confiance*** et propose des ***règles pour la combinaison des résultats de mesure***. Elle est caractérisée principalement par l'utilisation d'un *facteur de certitude* qu'on associe à chaque expression, pour refléter la confiance (ou plutôt le «degré de confiance ») qu'on attribue à l'expression. On distingue la « confiance pour » de la « confiance contre ». Ainsi on a toujours deux (2) mesures : MB(H|E) qui mesure la croyance associée à l'hypothèse H sachant E, MD(H|E) qui mesure la non croyance associée à l'hypothèse H sachant E, avec soit 1>MB(H|E)>0 tandis que MD(H|E)=0, soit 1>MD(H|E)>0 tandis que MB(H|E)=0. Le facteur de certitude se calcule grâce à la formule suivante : CF(H|E) = MB(H|E) - MD(H|E). Plus le facteur de certitude se rapproche de 1, plus la confiance pour l'hypothèse H est forte, tandis que plus le facteur de certitude se rapproche de -1, plus la confiance contre l'hypothèse H est forte.

Nous comptons ainsi associer à chaque instance de concept de mesure une croyance (*facteur de certitude*) que nous allons appeler **pertinence** de l'instance (cette croyance dépend de l'expérience du mesureur et de la qualité des spécifications). Ces pertinences seront utilisées dans le calcul de la précision des résultats de mesure, en combinant les pertinences suivant les règles proposées par la théorie sus-citée.

Après la représentation des connaissances, nous abordons dans la section suivante la question de catégorisation/classification en précisant sa portée dans le domaine de la mesure de la taille fonctionnelle des logiciels.

3 MESURE ET CATEGORISATION/CLASSIFICATION

3.1 LA CATÉGORISATION/CLASSIFICATION DANS LES SCIENCES DE LA COGNITION

La catégorisation peut être perçue comme le processus cognitif d'identification/formation/production et description de catégories (chaque catégorie pouvant être vue comme une abstraction représentant une collection de choses/entités concrètes ou abstraites, ayant des propriétés en commun). La ligne de démarcation avec le processus de classification n'est pas toujours évidente, mais elle existe : Dans la catégorisation, les catégories ne sont pas connues au départ (elles sont produites), tandis que dans la classification, les instances sont associées aux catégories existantes, résultant éventuellement en un raffinement des catégories. Catégorisation et classification sont considérées par certains auteurs comme faisant partie des plus importantes habilités dont peut disposer un organisme [Ashby et al., 01]. Dans la même lancée, John Sowa relève que « la catégorisation et la classification sont les fondements de l'intelligence » [Sowa 03]. Il n'est donc pas surprenant que ce processus suscite un intérêt sans cesse croissant dans la communauté scientifique en sciences de la cognition, toutes disciplines confondues, qu'il s'agisse de la philosophie, de la psychologie, des neurosciences cognitives, de l'informatique cognitive ou encore de la linguistique. Une multitude de thèmes de recherche font l'objet de travaux en cours. Ils portent de manière générale sur la représentation des catégories, l'apprentissage des catégories, la simulation du processus de catégorisation, la sémantique des catégories, le changement conceptuel, la catégorisation dans les différents processus mentaux/cognitifs (perception, langage, apprentissage, raisonnement …). Les

questions connexes sont relatives entre autres à la reconnaissance de patrons, la nature des représentations internes (iconiques, symboliques, …) ou encore la mémoire sémantique (comment sont stockées et exploitées les connaissances chez l'humain). Les questions liées à la représentation des catégories ou du processus de catégorisation figurent parmi les grandes questions de recherche sur la catégorisation en informatique cognitive (notre domaine d'intérêt).

3.2 LA CATÉGORISATION EN INFORMATIQUE COGNITIVE

L'informatique cognitive, en tant que faisant partie de l'ensemble des disciplines qui constituent les sciences de la cognition, s'intéresse bien entendu à la cognition et à l'intelligence, ou plutôt aux comportements intelligents. En effet, l'informatique cognitive fournit aux sciences cognitives un cadre et un support grâce auxquels les chercheurs expriment leurs théories sur l'activité mentale. La modélisation et les modèles occupent ici une place centrale. Ainsi, bien des travaux de recherche sont centrés sur la modélisation de l'architecture de la cognition ou encore la gestion des connaissances (représentation, organisation, manipulation, génération …). Ces modèles se doivent d'être non seulement assez réalistes (fidèles à ce qu'ils représentent/décrivent : plausibilité psychologique ou linguistique), mais également exploitables par un ordinateur (interprétables, manipulables avec une complexité raisonnable : contrainte d'ingénierie). Malheureusement, dans bien des cas, la première préoccupation est sacrifiée au profit de la seconde. La problématique de la représentation des connaissances et donc des catégories, qui se veut un pont entre la logique, l'informatique, la psychologie cognitive et perceptuelle, la linguistique et d'autres branches des sciences de la cognition, est incontournable dans les débats en IA sur la catégorisation. On note depuis un certain temps un engouement majeur pour les ontologies (qui contiennent les catégories [Sowa, 00]) ou encore la simulation du processus de catégorisation (avec le data mining). Les ontologies, en tant que collections de concepts formalisés, pourraient être perçues comme une réponse de l'IA à la question de représentation/description des catégories/connaissances incluant les liens entre elles. Les concepts ici renvoient aux représentations "mentales" (ou supposées) des catégories, incluant éventuellement les procédures pour leur identification ou classification. Le formalisme utilisé dans les descriptions revêt un caractère essentiel pour l'exploitabilité des ontologies : Il doit non

seulement intégrer les aspects syntaxiques, mais aussi les aspects sémantiques de ce qui est décrit. Ici s'établit la jonction avec la sémiotique dans ses trois branches : (1) syntaxe (relations entre signes[11]); (2) sémantique (liens entre signes et choses du monde réel); (3) pragmatique (liens entre signes et agents qui les utilisent). En effet, si l'on considère comme Pierce que la sémiotique est la science qui étudie l'utilisation des signes par « toute intelligence scientifique » (« toute intelligence capable d'apprendre par expérience ») [Sowa 00], alors l'on pourrait en déduire que les techniques informatiques de traitement des bases de connaissances et de données sont des techniques de sémiotique computationnelle.

L'un des projets majeurs de catégorisation d'envergure internationale en informatique a été initié en 1984 par Doug Lenat [Sowa, 03]. Il s'intitulait Cyc et visait la mise en place d'une base de connaissances générale pour un être humain moyen. On a ainsi pu identifier environ 600 000 catégories, décrites à l'aide de 2 000 000 d'axiomes. Malheureusement, il se pose encore la question d'exploitation de ces connaissances en vue de bâtir des théories (implantation des techniques d'abduction), lesquelles théories seraient utilisées pour la prédiction (implantation des techniques de déduction). Le raisonnement par analogie pourrait avoir une place prépondérante ici comme le suggère John Sowa [Sowa 03]. Ce mode de raisonnement pourrait s'avérer pertinent dans les tâches de classification.

Dans la section suivante, nous abordons la question de catégorisation/classification relativement au domaine de la mesure de la taille fonctionnelle des logiciels.

3.3 LA CATÉGORISATION/CLASSIFICATION DANS LE DOMAINE DE LA MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS

Dans notre projet de recherche, il est question d'ontologie : Nous proposons, tel que mentionné plus haut, une formalisation ontologique des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels, dans le but de faciliter l'application des méthodes de mesure et le développement d'outils pour la mesure ou l'assistance à la mesure. Une telle formalisation passe par une bonne compréhension du « design » d'une méthode de mesure de la taille fonctionnelle des logiciels. Nous postulons un lien entre la catégorisation et le design d'une méthode de mesure de la taille fonctionnelle des logiciels.

[11] Quelque chose (par exemple un icône) qui fait référence a quelque chose pour quelqu'un

3.3.1 CATÉGORISATION ET DESIGN D'UNE MÉTHODE DE MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS

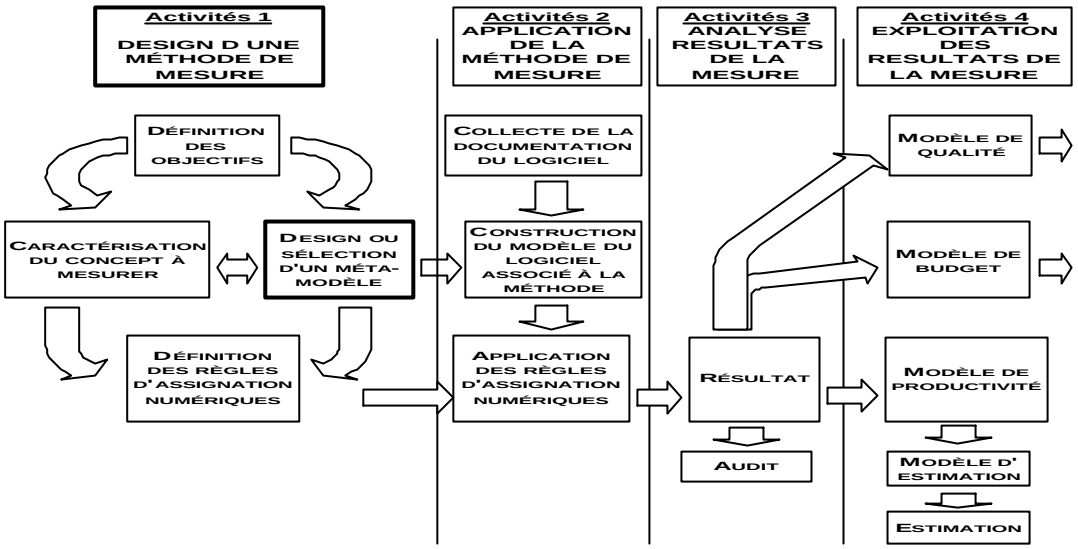


Figure 10 : Les activités de mesure : Adaptation du modèle détaillé du processus de mesure (Traduction de J.P. Jacquet & A. Abran, 1997) [Jacquet et al., 97]

Les activités de mesure peuvent être regroupées en quatre principales catégories (Figure 10) :

C. les activités de « design » des méthodes de mesure (définition des objectifs de mesure [estimation de coût, d'effort …], caractérisation du concept à mesurer [la taille fonctionnelle par exemple], « design » ou sélection d'un « meta-modèle » ("static model" [Bévo et al. 01], "generic software model"/"context model" [Abran et al. 03, ISO/IEC 03a]) représentatif de l'objet de la mesure [le logiciel par exemple], définition des règles d'assignation numériques),

D. les activités d'application des méthodes de mesure (collecte de la documentation d'un logiciel [documents de spécification, …], construction du modèle du logiciel relativement à une méthode de mesure et application des règles d'assignation numérique au modèle construit en vue de déterminer la taille fonctionnelle),

E. les activités d'analyse des résultats de mesure, par exemple pour des besoins d'audit des méthodes de mesure,

F. les activités d'exploitation des résultats de mesure (utilisés dans des modèles de qualité, de budget, de productivité ou encore d'estimation de coût/effort).

Le « design » d'une méthode de mesure de la taille fonctionnelle du logiciel passe par la précision des objectifs de la mesure et une bonne compréhension du concept de «taille fonctionnelle » du logiciel : Que veut-on mesurer ? De quel point de vue veut-on mesurer ? À quoi va servir le résultat de la mesure ? Voilà autant de questions auxquelles on doit répondre. La «taille fonctionnelle » étant un attribut du logiciel (objet de la mesure), il convient alors de proposer une façon de modéliser le logiciel de façon à bien cerner cet attribut, mettre l'emphase sur ce qui est pertinent par rapport à la mesure (dans la littérature, on parle de design d'un « méta-modèle »). À ce niveau intervient le gros du travail d'analyse du « designer » d'une méthode de mesure de la taille fonctionnelle des logiciels. Il doit alors répondre à une série de questions : En quoi consiste un logiciel ? Quelles sont les différentes façons de le décrire (le modéliser, le représenter) ? Comment le décrire (représenter, modéliser) de façon à faire ressortir les éléments « représentatifs », « pertinents » relativement au concept de «taille fonctionnelle » ? Que représente le concept de «taille fonctionnelle » d'un logiciel ? Quels en sont les concepts sous-jacents ? Comment identifier dans une description du logiciel donnée, ce qui est «représentatif» du concept de «taille fonctionnelle » et ce qui ne l'est pas ? Comment tenir compte, dans le «méta-modèle » à proposer, des objectifs essentiels de la mesure de la taille fonctionnelle du logiciel (entre autre la facilité d'application) ?

Le problème auquel est confronté le concepteur d'une méthode de mesure lorsqu'il doit produire un « meta-modèle » associé à la méthode de mesure, peut être perçu en partie comme un problème de catégorisation. En effet, il s'agit de :

- Partir d'un ensemble d'éléments descriptifs d'un logiciel (spécifications du logiciel, …), d'opérer des regroupements et déduire des classes d'éléments (catégories) pertinentes relativement aux objectifs de la mesure et au concept à mesure. Par exemple, pour la mesure de la taille fonctionnelle des logiciels, la norme ISO [ISO 93, ISO/IEC 98, ISO/IEC 97a, ISO/IEC 97b, Abran et al. 99] qui sert de base de travail pour les « designers », regroupe au plus haut niveau les éléments descriptifs d'un logiciel en deux catégories : les éléments qualitatifs ou techniques et les éléments fonctionnels. Chaque catégorie identifiée pourrait être décomposée en sous- catégories et ainsi de suite. Par exemple dans le cas de la méthode de mesure FPA, les éléments fonctionnels (appelés FUR) sont regroupés en deux catégories : les éléments de données et les éléments de transactions. Les éléments de données sont à leur tour regroupés en deux

catégories : les groupes de données logiques (ILFs) et les données externes (EIFs). Quant aux éléments de transactions, ils sont regroupés en trois catégories : les entrées (EI), les sorties (EO) et les requêtes (EQ). Les autres méthodes de mesure (COSMIC-FFP, MkII-FPA) proposent des catégories différentes.

- Synthétiser dans un modèle (« meta-modèle ») les catégories recensées avec les relations entre elles (conceptualisation). Le résultat de ce travail est ce que l'on pourrait qualifier de «modèle » de la méthode de mesure qu'on veut bâtir. Selon Walter G. Vincenti [Vincenti 90] «le besoin de design entraîne l'émergence d'une sorte particulière de connaissance ». Ainsi, l'on devrait retrouver dans ce « modèle », le gros des « connaissances » dites «statiques » manipulées lors de l'application d'une méthode de mesure de la taille fonctionnelle du logiciel.
- Proposer une façon d'identifier sans ambiguïté les membres de chacune des catégories recensées. Il s'agit en fait de proposer une procédure de classification ou à défaut des règles permettant, étant donnée un élément descriptif d'un logiciel, de déterminer la catégorie à laquelle il appartient. Cette procédure de classification ou ces règles d'identification sont très utiles pour l'application des méthodes de mesure (« connaissances » dites « procédurales » nécessaires pour l'application d'une méthode de mesure de la taille fonctionnelle du logiciel). Ainsi l'activité d'application d'une méthode de mesure de la taille fonctionnelle du logiciel pourrait être perçue comme une activité de classification.

3.3.2 CLASSIFICATION ET APPLICATION D'UNE MÉTHODE DE MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS

En effet, dans cette activité de mesure, l'on déroule un algorithme, une procédure (la procédure de mesure associée à une méthode de mesure).

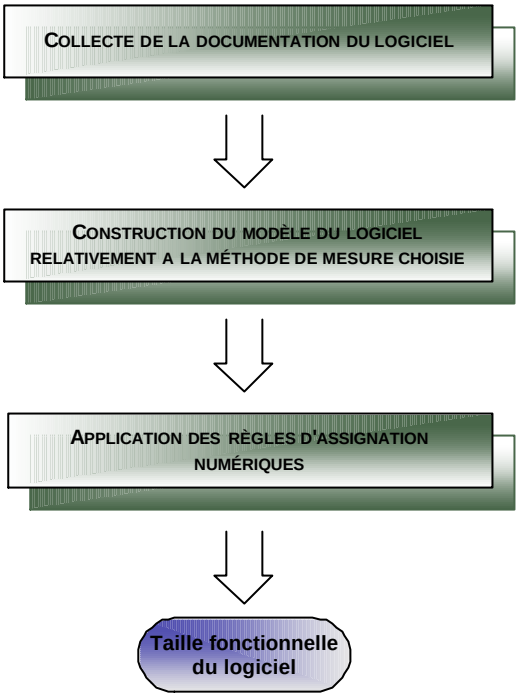


Figure 11 : Une abstraction de la procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels (*Adaptée de [Jacquet et al., 1997]*)

Il apparaît clairement sur la Figure 11 ci-dessus que la procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels comporte deux (2) principales phases: (1) une phase d'"identification/classification et modélisation" (identification/classification des éléments de spécification pertinents relativement à la mesure et construction du modèle du logiciel à mesurer, conformément à la méthode de mesure considérée), et (2) une phase de "mesure" (assignation de valeurs numériques à certains éléments du modèle obtenu précédemment et agrégation de ces valeurs pour dériver la taille fonctionnelle du logiciel à mesurer). La première phase est la plus déterminante des deux. En effet, d'elle va dépendre la qualité des résultats de mesure. Elle consiste en partie en une classification de certains éléments de spécification du logiciel à mesurer (ceux tirées de la documentation du logiciel et jugés pertinents relativement à la mesure) : Pour chaque élément de spécification identifié, il s'agit de déterminer à quelle catégorie il correspond relativement à la méthode de mesure (les catégories dont il est question ici sont celles qui sont décrites dans le «meta-modèle » associé à la méthode de mesure). Ce travail sera d'autant plus aisé s'il existe une correspondance (un « mapping ») entre les concepts de mesure et les concepts de spécification, auquel cas ce travail pourrait même être automatisé.

Notons que les concepts de mesure sont ceux qu'on retrouve dans le « meta-modèle » associé à une méthode de mesure, tandis que les concepts de spécification sont ceux que l'on retrouve dans le langage de spécification utilisé dans la documentation du logiciel. Malheureusement, la correspondance entre ces concepts est rarement disponible, ce qui n'est pas pour faciliter le travail du « mesureur ». La synthèse de ce travail de classification est réalisée dans un modèle du logiciel à mesurer, qui n'est qu'une instanciation du «méta-modèle » associé à la méthode de mesure.

En somme, une procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels, opère dans sa première phase le passage d'un formalisme dans lequel est décrit le logiciel (dans les documents de spécifications, par exemple le formalisme UML) à un formalisme associé à la méthode de mesure considérée. Un tel passage suppose nécessairement une mise en correspondance (« mapping ») entre les éléments du formalisme associé à la méthode de mesure et les éléments du formalisme dans lequel est décrit le logiciel (dans les documents de spécifications). Ici l'on est confronté à la question épistémologique du « mapping », qui constitue un autre lien entre notre projet de recherche et les sciences de la cognition.

4 MESURE ET « MAPPING »/TRADUCTION

L'on ne saurait parler de **« mapping »**, entendu comme **traduction** ou encore établissement d'une correspondance entre deux «choses » (objet, concepts, structure, etc.), sans évoquer le raisonnement analogique qui le sous-tend et en fournit la sémantique. « Mapping » et analogie bien que deux notions différentes, s'apparentent au niveau du principe conducteur. En effet, autant la question analogique est celle de trouver une transformation (map) d'une source vers un but, autant l'objectif principal dans le « mapping » est de trouver une transformation (map) établissant un lien entre deux « choses » (objet, concepts, structure, etc.). La section qui suit est consacrée à la question analogique. Le rapprochement avec la notion de «mapping » y est fait.

4.1 « MAPPING » ET ANALOGIE

« La forme la plus précise, et sans doute le point de départ logique de la notion d'analogie, est dans l'idée d'une égalité de rapports au sens mathématique » [Dorolle 49]. Ce constat est bien illustré dans la métaphore suivante, «l'homme le plus grand est plus grand que la plus grande femme, comme l'homme est plus grand que la femme »; il y apparaît une « égalité » de rapports de tailles : rapport de tailles entre l'homme le plus grand et la plus grande femme *versus* rapport de tailles entre l'homme et la femme. Dans cette autre métaphore, «La vieillesse est à la vie comme le soir au jour », l'on souligne une « égalité » de rapports plus abstraits : rapport entre la vieillesse et la vie *versus* rapport entre le jour et le soir. L'égalité ici ne doit pas être prise au sens strict, mais plutôt dans un sens proche de la similitude, de la «similarité » ou encore de la ressemblance. Selon Kant, « l'analogie conclut de la ressemblance particulière de deux choses à la ressemblance totale, d'après le principe de la spécification » [Dorolle 49]. Ainsi, l'analogie permet de généraliser et se caractérise dans une certaine mesure comme une induction. En effet, elle pourrait être perçue comme une comparaison systématique entre structures, exploitant les propriétés et relations entre objets d'une structure source pour inférer sur les propriétés et relations entre objets d'une structure cible. Ce genre de traitement se retrouverait au niveau de la cognition humaine, dans des tâches telles que la perception visuelle de haut niveau, l'apprentissage ou encore le changement conceptuel [Gentner et al. 97]. Il n'est donc pas étonnant qu'il soit au cœur des activités de recherche en sciences de la cognition en général et en IA en particulier. En IA, l'intégration du traitement analogique dans les systèmes apparaît comme un enjeu crucial dans le souci sans cesse pressant de les doter d'un brin d' « intelligence » (flexibilité et puissance du raisonnement s'inspirant de l'humain). L'un des principaux points de mire c'est la modélisation computationnelle de l'analogie. Une classification suivant l'approche de représentation adoptée identifie trois (3) principales classes de modèles computationnels de l'analogie [French 02] : les modèles dits symboliques (utilisant les règles de production, les cadres, les prédicats, etc.; adéquates pour des structures relationnelles), les modèles connexionnistes (utilisant par exemple les réseaux de neurones artificiels; adéquates pour des structures émergentes) et les modèles hybrides (combinaison entre modèles symboliques et modèles connexionnistes). Un certain nombre de techniques de raisonnement exploitant ces modèles ont été mises en œuvre dans des systèmes informatiques. Parmi les techniques

les plus populaires, citons le raisonnement à base de cas (en anglais *Case Based Reasoning*) [Schank 82, Aamodt et al. 94], avec ses variantes : le raisonnement à base de problèmes (en anglais *Problem Based Reasoning*) [Kolodner et al. 96, Hmelo 95, Barrows et al. 95], le raisonnement à base d'exemples (en anglais Examplar Based Reasoning) [Bareiss 89, Kibler et al. 87, Porter et al 86], etc. Dans le raisonnement à base de cas, l'idée centrale est la résolution de nouveaux problèmes par adaptation de la solution à un problème «similaire » déjà résolu (appelé un «cas »). La «similarité » entre problèmes et «cas » ici, est établie suivant le principe de l'analogie. Le raisonnement à base de cas et ses variantes peuvent être perçus comme sous-tendus par des formes d'*analogie intra domaine*. Qu'il s'agisse de l'analogie intra domaine ou de l'analogie inter domaine, elles se traduisent par l'établissement d'une correspondance (ou « mapping ») entre une source et un but (« mapping » à l'intérieur d'un même domaine pour l'une et « mapping » entre deux domaines pour l'autre). Dans le cas de deux (2) langages ou encore de deux ontologies, il s'agit d'établir une correspondance entre les concepts des deux (2) langages/ontologies. Un tel « mapping » occupe une place centrale dans le processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications présentées dans un formalisme donné (UML par exemple). C'est l'objet de la section qui suit.

4.2 LE « MAPPING » DANS LE PROCESSUS D'APPLICATION D'UNE MÉTHODE DE MESURE DE LA TAILLE FONCTIONNELLEDES LOGICIELS

4.2.1 LE "MAPPING ONTOLOGIQUE"

Le processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels opère une transformation (map) permettant de passer d'une description d'un logiciel dans un formalisme donné (langage de spécifications) à une description du même logiciel dans un autre formalisme (formalisme associé à une méthode de mesure). Une telle transformation utilise un ensemble de correspondances entre les deux formalismes en jeu. Nous pensons qu'il serait judicieux d'expliciter ces correspondances (résultats du « mapping » entre les deux formalismes) pour simplifier la transformation. Cette tâche incomberait aux concepteurs des méthodes de mesure. A la lumière de ce que proposons dans cette thèse, leur tâche se ramènerait à une mise en correspondance entre ontologies : nous parlons de **« mapping ontologique »**. Il s'agit de mettre en correspondance une

ontologie associée à une méthode de mesure (par exemple COSMIC-FFP) et toute ou partie d'une ontologie définie par un langage de spécifications de logiciels (par exemple UML). Les ontologies ciblées sont les ontologies de domaine (concepts et relations entre concepts). De façon plus précise, il s'agit d'établir des correspondances entre les concepts de telles ontologies, tel qu'indiqué sur la Figure 12 ci-dessous.

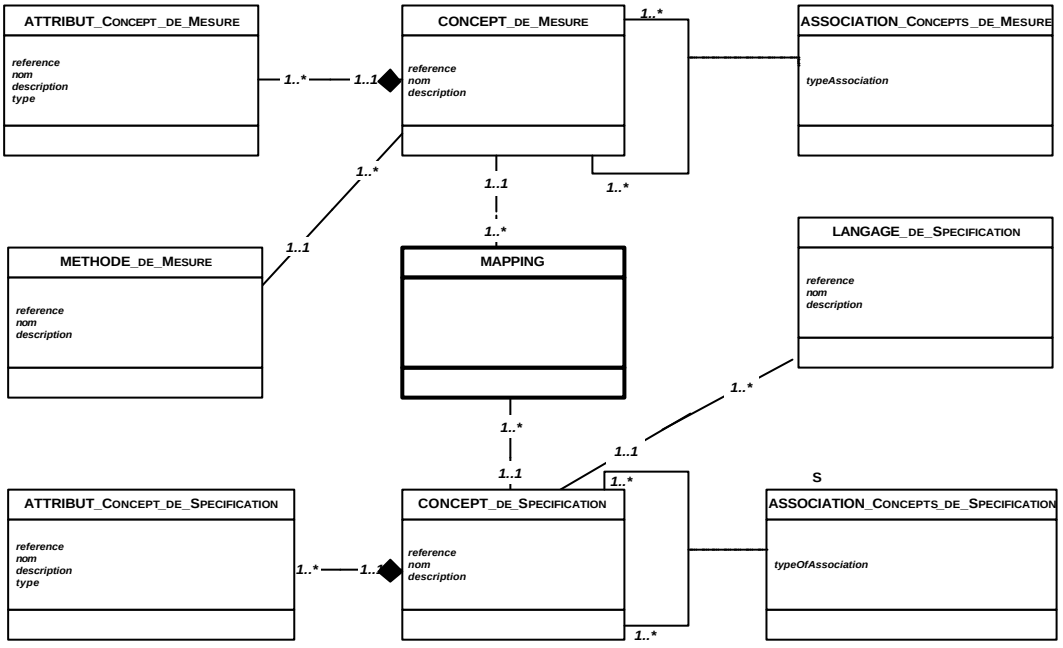


Figure 12 : Mapping entre ontologies de domaine associées aux méthodes de mesure et celles associées aux langages de spécifications.

Sur la Figure 12, l'ensemble des correspondances entre concepts de mesure et concepts de spécification est matérialisée par la classe association (MAPPING) qui lie les classes CONCEPT_DE_MESURE et CONCEPT_DE_SPECIFICATION. L'existence d'un lien entre instances de ces classes est synonyme d'existence d'une correspondance entre concepts représentés par ces instances. L'ensemble des liens entre concepts de mesure est matérialisé par la classe association ASSOCIATION_CONCEPT_DE_MESURE (il peut s'agir de liens entre concepts d'une même méthode de mesure ou alors entre concepts de deux méthodes de mesure distinctes). L'ensemble des liens entre concepts de spécification est matérialisé par la classe association ASSOCIATION_CONCEPT_DE_SPECIFICATION. La classe ATTRIBUT_CONCEPT_DE_MESURE matérialise l'ensemble des attributs des différents

concepts de mesure tandis que la classe ATTRIBUT_CONCEPT_DE_SPECIFICATION matérialise l'ensemble des attributs des différents concepts de spécification. Chaque concept de mesure est relatif à une méthode de mesure (METHODE_DE_MESURE) et chaque concept de spécification est relatif à un langage de spécification (LANGAGE_DE_SPECIFICATION).

Dans la suite, nous proposons une approche pour la réalisation du « mapping ontologique » susmentionné.

4.2.2 REALISATION DU "MAPPING ONTOLOGIQUE"

Tel que mentionné plus haut, un rapprochement peut être fait entre le « mapping » et le *raisonnement par analogie*. Le «mapping ontologique » ne déroge pas à ce constat. En effet, l'objectif principal dans le «mapping ontologique » dont il est question ici, est de trouver une transformation (map) permettant de passer d'une description d'un logiciel dans un formalisme donné (langage de spécifications) à une description du même logiciel dans un autre formalisme (formalisme associé à une méthode de mesure). Ainsi, le travail de « mapping » ici pourrait s'inspirer fortement de l'ossature utilisée pour créer une analogie :

- **Recueil de concepts** : Identification des concepts de spécifications et des concepts de mesure qui seront mis en correspondance. Ces concepts sont disponibles dans les ontologies de domaine spécifiques au langage de spécifications et à la méthode de mesure considérés. L'on aboutit à une liste de concepts de spécification et une liste de concepts de mesure;

- **Élaboration :** Spécification explicite et de façon détaillée de chacun des concepts sélectionnés. En réalité, ce travail a déjà été effectué dans le cadre du développement des ontologies de domaine spécifiques au langage de spécifications et à la méthode de mesure considérés. Il s'agit donc de récupérer les spécifications;

- **Transformation (map) et inférences** : Proposition de correspondances entre concepts de mesure et concepts de spécifications. Ces correspondances constituent l'essence même de la transformation (map) permettant de passer d'une description d'un logiciel dans un formalisme donné (langage de spécifications) à une description du même logiciel dans un autre formalisme (formalisme associé à une méthode de mesure);

- **Justification** : Documentation et détermination de la validité de la transformation. La validité de la transformation va dépendre en partie de la pratique courante de la mesure. Pour cela une ou plusieurs études de cas peuvent être réalisées, en vue de vérifier que les résultats obtenus (modèles de logiciels suivant la méthode de mesure considérée) en exploitant la transformation, sont identiques ou se rapprochent de ceux obtenus par des mesureurs experts;
- **Représentation** : Trouver une façon de représenter les correspondances obtenues, dans la perspective de l'automatisation de la transformation (un aspect très important de l'automatisation d'une procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications).

Un exemple présentant une mise en correspondance (« mapping ») entre concepts UML et concepts COSMIC-FFP est présenté en Annexe E de ce document. L'idéal serait que l'on puisse réaliser le « mapping » automatiquement, c'est-à-dire que l'on fournisse à un système les deux ontologies (l'une associée aux spécifications et l'autre associée aux méthodes de mesure), et qu'il soit capable (par apprentissage) de produire les correspondances. Mais il faut faire face aux *questions épistémologiques* soulevées par le « mapping », qui s'apparentent à bien des égards à celles soulevées par la traduction en linguistique. Comme le fait remarquer Jayant Madhavan [Madhavan et al. 02], le processus de « mapping » peut difficilement être automatisé au complet. Dans le cadre de la thèse, nous nous contentons d'examiner la possibilité de fournir au système les correspondances, et qu'il soit capable de les représenter dans un format utilisable automatiquement pour la mesure (sous forme de règles, sous forme de procédures stockées, etc.). Le principal défi se trouve alors au niveau de la représentation appropriée des correspondances. Nous avons opté pour l'approche base de données. A chaque correspondance, est associée dans la base de données une **table** avec deux champs qui stockent les références aux concepts mis en correspondance. Les tuples de cette table sont automatiquement insérés par un **déclencheur** (« trigger »), également associé à la correspondance.

Nous postulons que si l'on dispose d'un mapping adéquat entre une ontologie associée à une méthode de mesure (par exemple COSMIC-FFP) et toute ou partie d'une ontologie définie par un langage de spécifications de logiciels (par exemple UML), alors il est possible d'automatiser une bonne partie du processus d'application d'une méthode de mesure, et

donc de simuler par un programme ce processus. La simulation constitue un autre thème de recherche en liaison avec notre projet de recherche.

5 MESURE ET SIMULATION

5.1 LA SIMULATION: PRINCIPE THÉORIQUE

La simulation est indissociable du travail de modélisation qui la précède. En effet, on ne saurait parler de simulation d'un système du monde réel sans évoquer le modèle dudit système ayant servi à la simulation. Dans le cas de la simulation sur ordinateur qui constitue notre centre d'intérêt dans cette thèse, le modèle doit être implanté dans un ordinateur. L'ordinateur est utilisé comme outil de simulation. Dans la suite nous employons l'expression «processus de simulation» pour faire référence à l'activité de modélisation suivie de la simulation proprement dite d'un système du monde réel sur un ordinateur.

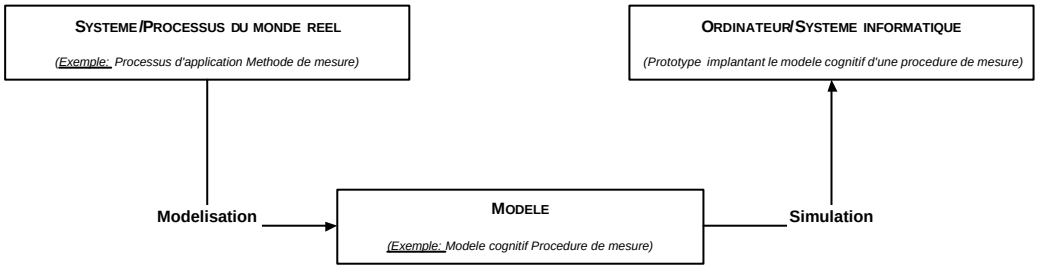


Figure 13 : Les éléments de base d'un « processus de simulation » [Zeigler 76]

Comme l'indique la Figure 13, on a trois (3) principaux éléments dans un « processus de simulation » :

- un **système du monde réel** (une partie du monde réel à simuler). Il peut être naturel ou artificiel. *Exemple : le processus de développement des fleurs, l'organisation sociale d'une ruche d'abeilles, ou encore dans le cadre de cette thèse, le processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications, etc.;*
- un **modèle** (abstraction du système du monde réel). Il doit pouvoir intégrer les aspects structurels et «comportementaux» (ou plutôt fonctionnels) du système du monde réel, avec les hypothèses émises relativement à chacun de ces aspects (notamment lorsqu'ils sont inaccessibles). Il est décrit dans un langage formel ou informel. L'expressivité du

langage est un atout pour la compréhension du modèle. La description informelle d'un modèle consiste en une spécification (dans un langage non formel) des composants (avec leurs interactions), des variables descriptives (incluant les paramètres), et la présentation des principales hypothèses impliquées par la spécification. Si elle présente l'avantage de pouvoir communiquer la nature essentielle du modèle grâce a l'expressivité de certains langages non formels, elle se trouve confrontée à quelques problèmes intrinsèques : l'incomplétude (on ne peut pas penser à toutes les situations pertinentes et produire une description complète), l'inconsistance et les ambiguïtés. Pour ce qui est de la description formelle d'un modèle, elle consiste en une transformation de la description informelle en une spécification système (dans un langage formel). Elle a la vertu d'être concise, permettant de réduire voire éliminer les ambiguïtés inhérentes à la description informelle, d'en détecter les omissions et incohérences. Cependant, elle sacrifie l'interfaçage naturel avec l'intuition du lecteur. Pour une simulation sur ordinateur la description formelle (ou au moins semi formelle) est incontournable (dans bien des cas sous forme de règles).

- un **ordinateur** dans lequel l'on implante le modèle et sur lequel se déroule la simulation.

Deux contraintes doivent être satisfaites pour mener à bien le « processus de simulation » : (1) le modèle doit être valide relativement au système du monde réel (il doit être une abstraction aussi fidèle que possible du système du monde réel); (2) le logiciel d'ordinateur résultant de l'implantation du modèle sur ordinateur doit « réaliser » aussi fidèlement que possible le modèle et générer le comportement prévu dans le modèle. Ces contraintes sont prises en compte dans la validation d'une simulation. Une « bonne » simulation doit entre autre être :

- *Réplicativement valide* : Les données résultant de la simulation doivent correspondre aux données existantes produites avec le système du monde réel;
- *Prédictivement valide* : Les données résultant de la simulation doivent correspondre aux données produites avec le système du monde réel après coup;
- *Structurellement valide :* La simulation doit reproduire le comportement observé du système du monde réel et refléter la façon dont le système du monde réel procède pour produire son comportement.

De la définition donnée par Jean-Louis Lemoigne de la « modélisation » (*voir Glossaire en Annexe A*), il appert que modélisation et simulation se complètent dans la compréhension d'un phénomène perçu complexe. Nous pensons que ceci s'applique aussi dans le domaine de la mesure, relativement au processus d'application d'une méthode de mesure de la taille fonctionnelle du logiciel. Dans la section qui suit, nous abordons la question de simulation de l'application d'une méthode de mesure de la taille fonctionnelle des logiciels.

5.2 LA SIMULATION DE L'APPLICATION D'UNE MÉTHODE DE MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS

Un système qui automatise la procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels peut être perçu comme un système qui simule le travail du « mesureur » lorsqu'il mesure la taille fonctionnelle d'un logiciel. La simulation suppose que l'on puisse fournir au système les connaissances nécessaires dans un format et un formalisme qui lui sont accessibles. Dans notre cas ces connaissances sont rassemblées dans les ontologies proposées dans la phase exploratoire du projet.

La taille fonctionnelle d'un logiciel[12], obtenue à la fin processus d'application d'une méthode mesure de la taille fonctionnelle des logiciels, doit être perçue comme la résultante d'un processus de conceptualisation bien déterminé, l'objet de la conceptualisation étant le logiciel. Il s'agit donc dans le cas de l'automatisation de la mesure, de simuler ce processus dans un programme informatique. La particularité de ce processus est qu'il part de quelque chose d'abstrait, de conceptuel, à savoir le logiciel. Il intègre un certain nombre de tâches de mesure relativement indépendantes, plus ou moins complexes, qui doivent être accomplies par le « mesureur », ou alors, dans le cas de l'automatisation du processus de mesure, par un logiciel. Ces tâches font appel à une certaine expertise (en termes de connaissances relatives à la méthode de mesure et au processus d'application de la méthode). Parmi les tâches, citons celle d'analyse des éléments de spécification du logiciel à mesurer (modèles, diagrammes, texte si possible …), afin d'identifier ce qui est pertinent du point de vue de la mesure et ce qui ne l'est pas, en se basant sur le « mapping » entre concepts de mesure et concepts pertinents pour la mesure et issus du langage utilisé pour spécifier le logiciel à

[12] Correspond à notre connaissance empirique et objective du logiciel considéré.

mesurer. Cette tâche combinée avec d'autres tâches, permet d'aboutir à une représentation, une description du logiciel à mesurer, dans un formalisme propre à la méthode de mesure (instanciation du *modèle statique* [Bevo et al., 01] de la méthode de mesure considérée). Le *modèle statique* de la méthode de mesure sert ici de «*gabarit* » pour la représentation. Ce « *gabarit* » doit être fourni à l'outil logiciel dans le cas de l'automatisation. Pour ce qui est du « mapping » entre formalisme associé à la méthode de mesure et formalisme dans lequel est décrit le logiciel (dans les documents de spécifications), l'outil doit, à défaut de le réaliser, disposer au moins des correspondances et pouvoir les exploiter. A partir d'une instanciation du modèle statique d'une méthode et des règles d'assignation numérique proposées par la méthode, une autre tâche consiste à déterminer la taille fonctionnelle.

Remarque :

Si la dernière tâche sus-présentée (détermination la taille fonctionnelle) semble aisément *simulable* sur un ordinateur, la tâche de « mapping » entre formalisme associé à une méthode de mesure et formalisme dans lequel est décrit le logiciel dans les documents de spécifications, le paraît moins (l'on se heurte ici à la question épistémologique du « mapping », de la traduction). Dans cette thèse, **nous n'avons pas la prétention de pouvoir simuler entièrement sur un ordinateur tout le processus d'application d'une méthode mesure de la taille fonctionnelle des logiciels**. La simulation dont il est question ici est d'abord une simulation exploratoire, avant d'être expérimentale. Elle a pour but principal de mieux expliciter le processus, le rendre accessible au plus grand nombre d'utilisateurs. Elle ne vise pas la construction d'un système informatique modélisant effectivement un mesureur (être humain) dans la tâche de mesure de la taille fonctionnelle d'un logiciel. Le modèle cognitif que nous proposons (chapitre quatrième) synthétise cette simulation exploratoire. Nous ne pensons pas qu'en l'état actuel des travaux de recherche en sciences cognitives, la simulation expérimentale puisse être complète.

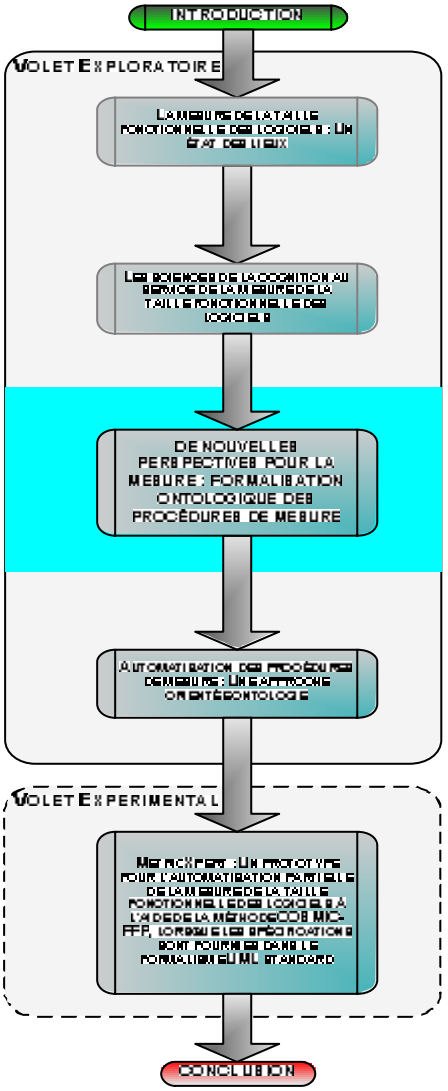
6 SYNTHÈSE

Au total, il ressort de ce chapitre qui s'achève, que la mesure de la taille fonctionnelle des logiciels tirerait bel et bien profit des travaux de recherche menés en sciences de la cognition, plus particulièrement sous l'angle des divers travaux reliés à représentation des connaissances. Le « design » et l'application d'une méthode de mesure tireraient par

exemple profit des travaux sur la catégorisation et la classification, notamment dans la perspective d'une systématisation des procédures de mesure. Il est à noter que les travaux sur la simulation pourraient être exploités dans une telle systématisation, qui pourrait être perçue comme une simulation du travail du « mesureur » lorsqu'il mesure la taille fonctionnelle d'un logiciel. D'autre part, les travaux sur le « mapping » et par extension la traduction ou encore l'analogie trouveraient un champ d'application en mesure. En effet, une procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels, opère dans sa première phase le passage d'un formalisme dans lequel est décrit le logiciel (dans les documents de spécifications, par exemple le formalisme UML) à un formalisme associé à la méthode de mesure considérée. Un tel passage suppose nécessairement une mise en correspondance (« mapping ») entre les éléments du formalisme associé à la méthode de mesure et les éléments du formalisme dans lequel est décrit le logiciel (dans les documents de spécifications). Le domaine de la mesure en génie logiciel et celui de la mesure de la taille fonctionnelle des logiciels étant relativement jeunes, la question de consensus autour des concepts et règles de mesure constitue l'une des préoccupations majeures des chercheurs du domaine, et pourrait bénéficier des travaux sur la représentation des connaissances. De plus, les travaux sur la représentation des connaissances pourraient apporter un peu plus d'intelligibilité aux procédures de mesure (explicitation des procédures de mesure, notamment des règles de mesure). Ce dernier point est au cœur du chapitre suivant, qui aborde la notion de formalisation ontologique des méthodes de mesure, par nous introduite dans cette thèse.

CHAPITRE 3.

DE NOUVELLES PERSPECTIVES POUR LA MESURE : FORMALISATION ONTOLOGIQUE DES PROCÉDURES DE MESURE



FEUILLE DE ROUTE

1. *Introduction*
2. *Les ontologies : Un survol*
3. *Formalisation ontologique des procédures de mesure*
4. *Etudes de cas (FPA, MKII-FPA, COSMIC-FFP) + validation des ontologies*
5. *Synthèse*

EN BREF …

Ce chapitre introduit la notion de formalisation ontologique d'une procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels (définition et principe). Les perspectives ouvertes relativement à la mesure par une telle formalisation des procédures de mesure sont explicitées. Trois études de cas sont fournies : elles sont relatives à la formalisation ontologique des procédures de mesure associées aux méthodes de mesure FPA, MkII-FPA et COSMIC-FFP respectivement. La question de la validation des résultats de la formalisation (ontologies) est abordée.

1 INTRODUCTION

Le développement des ontologies au cours de cette dernière décennie suscite un engouement sans précédent, non plus seulement dans les laboratoires de recherche en Intelligence Artificielle (IA) mais de plus en plus parmi les experts de divers domaines et sur le web (World-Wide Web). Dans plusieurs disciplines aujourd'hui, l'on assiste au développement d'ontologies standardisées pour faciliter les échanges entre experts et expliciter les connaissances du domaine. En médecine par exemple, on se tourne vers la production de gros lexiques structurés et standardisés tels que Snomed [Price et al. 00], et le développement d'une terminologie standard pour le domaine [Spackman et al. 00] (c'est par exemple l'un des objectifs principaux visés par le projet Unified Medical Language System [Humphreys et al. 93]). En Génie Logiciel, le projet SWEBOK[13] (initié en 1998 et soutenu par le IEEE Computer Society, a permis de jeter les bases d'un corpus de connaissances généralement acceptées du domaine, une sorte de méga-ontologie du domaine du génie logiciel. Certaines organisations internationales ne sont pas en reste. C'est le cas du PNUD (Programme des Nations Unies pour le Développement), qui en association avec Dun & Bradstreet, a développé une ontologie qui synthétise la terminologie pour les produits et services[14] [Noy et al. 01]. Sur le web, les ontologies prennent dans bien des cas la forme de grosses taxonomies (par exemple : sur Yahoo! on a une taxonomie catégorisant les sites web; sur Amazon.com on a une taxonomie de produits avec leurs caractéristiques; etc.).

Dans tous les cas, grâce aux ontologies, des agents humains ou logiciels peuvent partager la même compréhension des informations relevant d'un domaine considéré, la réutilisation des connaissances d'un domaine est standardisée, certaines hypothèses tacites dans un domaine sont rendues explicites, l'analyse des connaissances d'un domaine peut être plus rigoureuse [Noy et al. 01]. Pour toutes ces raisons, nous pensons que le domaine de la mesure en général, de la mesure de la taille fonctionnelle des logiciels en particulier pourrait tirer profit d'une «ontologisation » (développement d'ontologies associées). D'où l'idée de formalisation ontologique des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels que nous préconisons dans cette thèse.

[13] http://www.swebok.org/)

[14] http://www.unspsc.org/search.asp

Avant d'aborder la question de formalisation ontologique des procédures de mesure, nous proposons dans la section suivante un tour d'horizon des ontologies (définition, catégories d'ontologies, développement d'ontologies), question de mieux cerner les contours de la formalisation ontologique.

2 LES ONTOLOGIES

2.1 DÉFINITIONS

Étymologiquement, le terme **ontologie** est utilisé pour désigner l'étude philosophique de ce qui existe. Aujourd'hui, il n'est plus uniquement utilisé dans les débats philosophiques, certains auteurs lui ayant donné au cours de l'histoire des connotations se rapprochant de leurs domaines d'étude respectifs. En Intelligence Artificielle (IA) par exemple, le terme fait référence à un *ensemble de catégories, avec les liens entre ces catégories*. Il est bien souvent utilisé comme synonyme de *terminologie* dans un domaine spécifié (terminologie + interprétation sémantique des termes). L'on considère qu'une ontologie peut être spécifique à des tâches ou à des domaines[15].

L'une des définitions les plus populaires du terme est celle donnée par Grüber [Grüber 93], qui présente une ontologie comme une *spécification explicite d'une conceptualisation.* Mais celle-ci reste très générale, en tout cas moins spécifique que celle donnée par Albert (1993) (proche du domaine de la gestion des connaissances), qui considère une ontologie comme un *corpus de connaissances* relatives à une tâche particulière ou un domaine particulier (taxonomie de concepts pour la tâche ou le domaine, définissant l'interprétation sémantique de la connaissance). Ainsi, elle pourrait servir de plate-forme pour l'évolution vers un consensus dans un domaine considéré.

Dans la thèse, nous combinons différentes perceptions, en mettant l'accent sur les définitions de Grüber et Albert. Celle de Albert est en accord avec notre objectif de jeter les bases d'un corpus de connaissances relatives à l'application d'une méthode de mesure de la

15 Terminological ontologies : Specifie les termes utilisés pour représenter la connaissance dans un domaine (exemple : UMLS = Unified Medical Language System; Lindberg et al., 1993, Page: 66
).
Information ontologies : Structures des enregistrements de bases de données (exemple : schéma d'une BD)
Knowledgemodeling ontologies: specify conceptualizations of the knowledge.

taille fonctionnelle des logiciels. Quant à celle de Grüber, elle constitue pour nous l'épine dorsale du travail que nous avons entrepris. En effet, il est en grande partie question dans cette thèse de spécification explicite des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle (considérées comme des conceptualisations).

Cependant, comme nous allons le constater dans la section suivante, il existe plusieurs categories d'ontologies selon le domaine considéré ou le besoin comblé par l'ontologie. Chacune des catégories a des particularités bien spécifiques.

2.2 CATÉGORIES D'ONTOLOGIES

La question de la catégorisation des ontologies a été abordée par bien des auteurs dans la littérature [Van Heijst et al. 97, Guarino 98, Mizoguchi 02]. Pour certains, l'on pourrait catégoriser les ontologies suivant la nature de la conceptualisation sous-jacente à chaque ontologie. La Figure 14 présente les principales catégories identifiées par bon nombre d'auteurs.

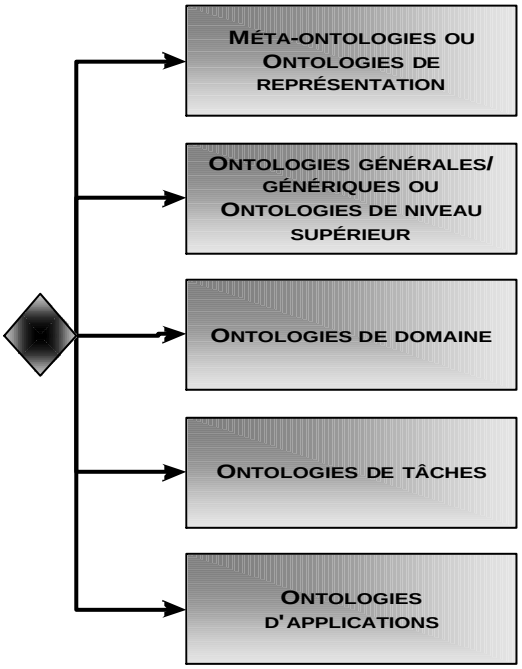


Figure 14 : Les principales categories d'ontologies [Van Heijst et al. 97, Mizoguchi 02]

Les **méta-ontologies** ou **ontologies de représentation** sont des ontologies qui servent à la formalisation des connaissances dans un système de représentation de connaissances. C'est le cas par exemple du Frame ontology [Gruber 93], qui définit les termes/concept pour

la capture/spécification des conventions utilisées dans les systèmes orientés objet de représentation des connaissances (concept de classe, d'association, de fonction, de relation unaire, de relation binaire, etc.).

Pour ce qui est des **ontologies générales/génériques** ou **ontologies de niveau supérieur,** elles classifient les différentes catégories d'entités existant dans le monde. L'on synthétise dans cette catégorie d'ontologie des notions très générales, indépendantes de tout domaine ou problème particulier. Par exemple, sur le serveur Ontolingua [Farquhar et al. 96] l'on a une ontologie du temps qui spécifie entre autres le concept d'instant, d'intervalle de temps, de durée, etc.

Contrairement aux ontologies générales, les **ontologies de domaine** sont plus spécifiques. Elles synthétisent les connaissances spécifiques à un domaine particulier *(exemple : ontologie d'éléments chimiques, ontologie du domaine de l'aviation civile, ontologie du domaine du génie logiciel, etc.)* ou à un processus. Les concepts du domaine ou du processus considéré, ainsi que les relations entre concepts et les théories gouvernant le domaine ou le processus y sont spécifiés. Dans bien des cas l'on a une organisation hiérarchique des concepts. Certaines ontologies de cette catégorie prennent la forme de thesaurus ou encore de taxonomies de termes (d'aucun parlent d'ontologies terminologiques). C'est le cas par exemple de l' ontologie *WordNet* [Patel et al. 03].

Tout comme les ontologies de domaine, les **ontologies de tâches** sont spécifiques à un domaine ou un processus particulier. Elles spécifient les tâches du domaine ou du processus considérée *(exemple : ontologie de tâches associée au processus de production dans une unité de production, ontologie de tâches associée au montage de véhicules sur une chaîne de montage, ontologie tâches associée au processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels, etc.)*

Les **ontologies d'applications** quant à elles spécifient les connaissances liées à la fois à une tâche et à un domaine particuliers. Elles sont associées à des méthodes de résolution de problèmes. Elles mettent en correspondance les concepts du domaine considéré avec les concepts faisant partie de la description d'une méthode de résolution de problèmes considérée. Le rôle joué par chaque concept du domaine considéré dans la méthode de résolution de problèmes considérée, y est explicité

Dans cette thèse nous nous orientons vers les ontologies de domaine et de tâches. Nous en développons quelques unes. Dans la section qui suit, nous abordons la question de

développement des ontologies. Nous examinons les principales méthodologies/approches existantes, les outils disponibles et les formalismes de représentation utilisés.

2.3 DÉVELOPPEMENT D'ONTOLOGIES

2.3.1 MÉTHODOLOGIES/APPROCHES DE DEVELOPPEMENT

Au cours de la dernière décennie, l'on a assisté à l'émergence d'un certain nombre de méthodologies pour le développement et la maintenance des ontologies. Dans bien des cas, elles sont le reflet des expériences personnelles des auteurs dans la construction d'ontologies [Visser et al. 98]. Nous passons en revue dans la suite, les principales méthodologies parmi les plus connues. Il s'agit des méthodologies TOVE, *Enterprise Model Approach*, METHONTOLOGY et IDEF5. Nous examinons aussi un certain nombre d'approches qui, à défaut d'être des méthodologies à part entière de développement d'ontologies apportent un éclairage et proposent de grandes lignes directrices pour le développement d'ontologies. Le travail réalisé par Visser [Visser et al. 98] nous sert de base (c'est le plus complet que nous ayons trouvé dans la littérature à ce moment). Nous en proposons une synthèse.

2.3.1.1 LA MÉTHODOLOGIE TOVE (GRUNINGER & FOX) [GRUNINGER ET AL. 95]

Elle est basée sur les expériences de développement d'une entreprise torontoise du même nom, TOVE (TOronto Virtual Enterprise) [Gruninger et al. 94a, Gruninger et al. 94b, Gruninger et al. 95, Uschold et al. 96]. Elle préconise les étapes suivantes pour le développement d'une ontologie :

- Une étape de *motivation* au cours de laquelle l'on fait une synthèse des problèmes rencontrés au niveau de l'entreprise et pouvant tirer profit du développement d'une ontologie. Une description des tâches pouvant être supportées par une ontologie est alors produite;
- Une étape de *description informelle des exigences* pour l'ontologie à développer, sous forme de questions informelles auxquelles l'ontologie devrait répondre;
- Une étape de *spécification terminologique* qui prévoit la spécification formelle des types objets que l'on retrouve dans l'ontologie, avec leurs attributs et les associations entre eux;

- Une étape de *description formelle des exigences* pour l'ontologie à développer, sous forme de terminologie formellement définie;
- Une étape de *spécification des axiomes* et des *contraintes d'interprétation* associés aux définitions des termes que l'on retrouve dans l'ontologie. La logique des prédicats du premier ordre est utilisée pour la spécification;
- Une étape d'*évaluation* dédiée à l'évaluation de l'ontologie produite au moyen de théorèmes de complétude.

L'un des points les plus intéressants de cette méthodologie est l'emphase mise sur l'évaluation des ontologies (une étape y est consacrée). Cependant, elle s'oriente plus vers le développement d'ontologies pour des entreprises.

2.3.1.2 LA MÉTHODOLOGIE « ENTERPRISE MODEL APPROACH » (USCHOLD ET AL. 95)

Elle est basée sur l'expérience de développement de l'ontologie *Enterprise* [Uschold et al. 95]. Elle préconise quatre (4) étapes principales dans le développement d'une ontologie :

- une étape d'*identification de la raison d'être* de l'ontologie et du *niveau de formalisation* requis pour l'ontologie;
- une étape d'*identification de l'envergure/étendue*, des *frontières* de l'ontologie. Les informations et «choses» (« conceptualisation ») à spécifier à travers l'ontologie sont identifiées. L'on pourrait faire appel ici aux étapes 1 et 2 de la méthodologie TOVE;
- une étape de *formalisation* des définitions et axiomes associées aux informations et « choses » identifiées précédemment comme faisant partie du champ d'intérêt de l'ontologie;
- une étape d'*évaluation* de l'ontologie produite.

Cette méthodologie simplifie considérablement le processus (quatre étapes au lieu de six comme dans TOVE). Cependant, elle s'oriente plus vers le développement d'ontologies pour un domaine spécifique (celui des affaires). De plus, comme la plupart des méthodologies existantes pour le développement d'ontologies, aucune indication n'est donnée sur la façon d'identifier des concepts ontologiques.

2.3.1.3 LA MÉTHODOLOGIE METHONTOLOGY [GOMEZ-PEREZ ET AL. 96]

Cette méthodologie identifie quatre (4) principales étapes dans le développement d'une ontologie :

- une étape dite de « *spécification* » qui en fait, est consacrée à l'identification de la raison d'être et l'envergure/étendue (les *frontières)* de l'ontologie. A cette étape se fait également l'acquisition des connaissances à spécifier;
- une étape de *conceptualisation* au cours de laquelle est produite une description informelle des concepts avec leurs propriétés et les liens entre concepts;
- une étape d'*intégration* prévue pour l'uniformisation de l'ontologie développée avec les autres ontologies existantes, évitant ici des incohérences par rapport à l'existant;
- une étape d'*implantation* dédiée à la description formelle des extrants de la conceptualisation;

Les activités d'*évaluation* et de *documentation* sont conduites tout au long du processus de développement. Les techniques d'évaluation sont celles utilisées dans la validation et la vérification des systèmes à base de connaissances. Cette méthodologie nous semble plus précise au niveau des étapes. Elle est assez bien documentée.

2.3.1.4 LA MÉTHODOLOGIE IDEF5 [KBSI 94]

Cette méthodologie définit une procédure générale et un ensemble de lignes directrices pour la création, la modification et la maintenance des ontologies [KBSI 94]. Elle identifie cinq (5) principales étapes dans le développement d'une ontologie :

- une étape d'*organisation* et de *détermination d'envergure* au cours de laquelle sont établis la raison d'être, le point de vue, le contexte du développement et les frontières de l'ontologie;
- une étape de *recueil de données* consacrée à l'acquisition des données nécessaires qui seront exploitées pour le développement de l'ontologie, grâce à des techniques d'acquisition de connaissances (analyses de protocoles, interviews d'experts, etc.);
- une étape d'*analyse des données recueillies* en vue d'en extraire les éléments ontologiques;

- une étape de *développement initial* de l'ontologie (production d'une ontologie préliminaire dans un langage, une notation graphique);
- une étape de *raffinement* de l'ontologie préliminaire et de *validation* des résultats;

Dans la suite, nous présentons un certain nombre d'approches qui, à défaut d'être des méthodologies à part entière de développement d'ontologies apportent un éclairage et proposent de grandes lignes directrices pour le développement d'ontologies.

2.3.1.5 LES AUTRES APPROCHES

L'approche Ontolingua

Cette approche est spécifiée dans le guide d'utilisation du serveur Ontolingua [Farquhar et al. 95, Farquhar et al. 96, Farquhar et al. 97]. Elle apporte un éclairage sur l'affichage, le développement, la maintenance et le partage des ontologies. L'accent est mis sur la réutilisation des ontologies.

L'approche CommonKADS

Conçue en 1985 par Anne Brooking (United Kingdom), Joost Breuker et Bob Wielinga (Pays-bas), la méthodologie KADS (initialement *Knowledge Acquisition and Documentation Structuring,* et devenue ensuite *Knowledge Analysis and Design System/Support*) est une méthodologie d'analyse et de modélisation des connaissances partagées, dans laquelle le cycle de vie de la connaissance et l'interaction système/utilisateurs prennent une place centrale. Cette approche est largement utilisée pour le développement des systèmes à base de connaissances dans lesquels les ontologies jouent un rôle important. Elle préconise le principe de développement modulaire, qui est populaire en ingénierie ontologique. Elle met également l'accent sur la réutilisation des ontologies [Wielinga et al. 94, Schreiber et al. 95]. Elle offre une approche structurée pour la documentation des connaissances. Au fil des années, elle a connu une évolution considérable, avec notamment le développement d'un langage de représentation formelle. Sa version actuelle baptisée CommonKADS, est considérée comme un standard européen de facto pour l'analyse des connaissances et le développement de systèmes orientés connaissances. La notation UML est mise à contribution pour : (1) modéliser le flux d'informations et de contrôles relativement à une procédure, un processus ou une tâche (*diagramme d'activités*) ; (2) modéliser le comportement dynamique de certains objets (*diagramme d'état-transition*) ; (3) modéliser la structure statique des objets (*diagramme de*

classes) ; (4) présenter une vue de haut niveau des services/fonctionnalités offert(e)s (*diagramme des cas d'utilisation*). La méthodologie CommonKADS est mise à contribution dans cette thèse, dans le cadre de la formalisation ontologique des procédures de mesure.

L'approche ONIONS (ONtological Integration Of Naive Sources)

Elle met l'accent sur l'acquisition des ontologies et des problèmes associées (A quel moment doit-on arrêter le raffinement d'une ontologie ? Comment déterminer ce qui est conceptuellement pertinent ?) [Steve et al. 96, Gangemi et al. 96]. Cette approche est motivée par le problème d'intégration des connaissances.

L'approche Mikrokosmos

Elle naît du développement de l'ontologie Mikrokosmos [Mahesh et al. 95, Mahesh 96] au cours duquel trente (30) lignes directrices ont été mises au point pour le développement. Ces lignes directrices apparaissent donc comme des heuristiques qu'on peut utiliser lors du développement d'ontologies.

L'approche SENSUS

Elle naît du développement de l'ontologie SENSUS [Swartout et al. 97], résultant de l'agrégation d'un certain nombre d'ontologies existantes, dont *Wordnet* [Patel et al. 03], *Ontos*, etc., avec des catégories sémantiques issues de dictionnaires électroniques.

Au delà de la multitude des méthodologies proposées, les étapes suivantes (Figure 15) nous semblent les plus pertinentes dans le développement d'ontologies.

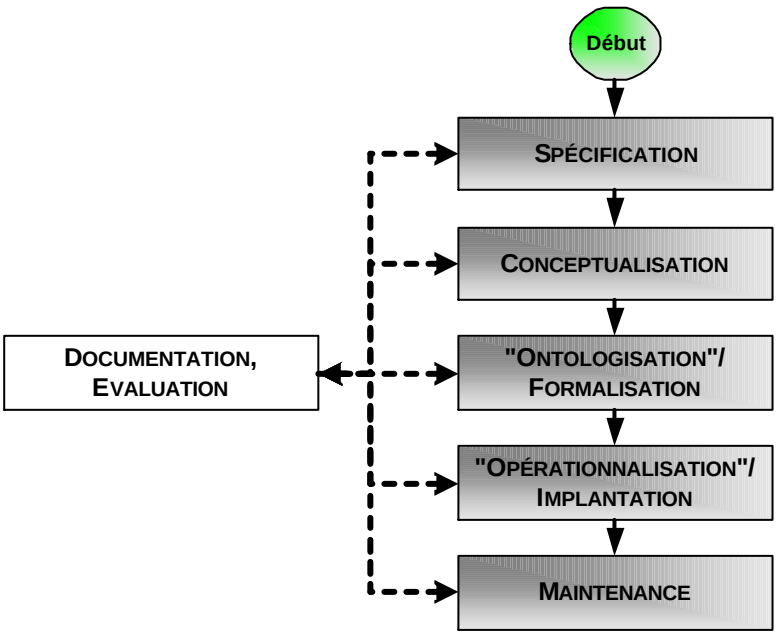


Figure 15 : Principales étapes du développement d'une ontologie

La première étape dans le développement d'une ontologie (étape de **spécification**) est consacrée à l'identification de la raison d'être (*le pourquoi*) et de l'envergure/étendue (*utilisations et utilisateurs*) de l'ontologie. A la deuxième étape (étape de **conceptualisation**), il s'agit d'identifier les « conceptualisations » à spécifier (*concepts, tâches, liens entre concepts, etc.*), en tenant compte de la raison d'être et l'envergure de l'ontologie (précisées de l'étape précédente). La troisième étape quant à elle (étape d'**« ontologisation »/formalisation**) est dédiée à la spécification des « conceptualisation » identifiées précédemment, dans un langage formel ou semi-formel (*par exemple UML*). A la quatrième étape (étape d'**« opérationnalisation »/Implantation**), il faut transcrire les extrants de l'« ontologisation » dans un langage formel et opérationnel lorsque c'est possible. La dernière étape (étape de **maintenance**) est consacrée aux mises à jour éventuelles de l'ontologie, question de corriger des erreurs ou intégrer de nouveaux besoins (*utilisations, utilisateurs, etc.*). Dans chacune des cinq (5) étapes il faut inclure des activités de **documentation** et d'**évaluation**.

Si la méthodologie de développement utilisée pour le développement d'une ontologie revêt un caractère important si l'on veut garantir une certaine traçabilité des résultats, le formalisme utilisé pour représenter et stocker les éléments ontologiques revêt un caractère essentiel pour l'exploitabilité de l'ontologie. Nous examinons dans la suite les différents formalismes utilisés pour la représentation et le stockage en ce qui a trait aux ontologies.

2.3.2 FORMALISMES DE REPRÉSENTATION & STOCKAGE POUR LES ONTOLOGIES

Plusieurs formalismes sont proposés dans la littérature pour la représentation des connaissances dans le cadre d'ontologies. Certains de ces formalismes sont assez populaires : **formalismes relationnels** (orientés objet, cadres, réseaux sémantiques, graphes conceptuels …), **formalismes à base de logique** (logique propositionnelle, logique des prédicats, logique floue …). Si les formalismes à base de logique se prêtent bien aux programmes informatiques, les formalismes relationnels se caractérisent par un bon degré d'expressivité et de simplicité. En général il s'agit de trouver un compromis entre expressivité, simplicité et efficacité/computabilité.

Au dessus de ces formalismes ont été développés des langages spécifiques, adaptés au stockage et à l'exploitation des ontologies par des systèmes informatiques. Ces langages peuvent être regroupées en deux (2) categories :

1. Les langages dits « classiques » : Ontolingua[16] [Farquhar et al. 96], LOOM[17] [Brill 93], OCML[18] - Operational Conceptual Modelling Language [Domingue et al. 99], F-Logic[19] - Deductive Object Oriented Database Language [Frohn et al. 99], etc.

2. Les langages à balises: XML – Extensible Markup Language [Harold 01], SHOE - Simple HTML Ontology Extensions [Heflin et al. 99], XOL –Ontology eXchange Language [Karp et al. 99], RDF - Resource Description Framework [Lassila et al. 99, Broekstra et al. 01, Brickley et al. 03, Beckett 04, Brickley et al. 04], DAML+OIL[20] - DARPA Agent Markup Language + Ontology Inference Layer [Horrocks 02], OWL - Web Ontology Language [McGuinness et al. 04], etc.).

Avec l'influence grandissante du Web, les langages pour ontologies reposent de plus en plus sur les technologies W3C[21] telles que RDF Schema, XML et RDF. XML et OWL. Ces technologies sont en passe de devenir des standards, respectivement pour les échanges de données et les ontologies sur le Web

Dans la thèse, nous avons choisi le formalisme orienté objet pour la spécification des éléments ontologiques (diagramme de classes UML et diagramme de composants UML sont utilisés respectivement pour les ontologies de domaine et les hiérarchies de tâches). La simplicité, l'expressivité et la popularité de UML ont motivé ce choix (l'indépendance par rapport à tout outil d'implantation a été également pris en compte). XML a été adopté pour le stockage des ontologies. La portabilité et l'exploitabilité des ontologies produites ont été des critères décisifs dans le choix. La popularité grandissante de XML et le fait qu'il soit en passe de devenir un standard ont également pesé dans la balance. De plus, l'exploitation des ontologies dans des SGBDs (systèmes de gestion de bases de données) est possible, grâce aux interfaces d'import/export de données dans le format XML qu'offrent actuellement la

[16] http://www.ksl.stanford.edu/software/ontolingua/

[17] http://www.isi.edu/isd/LOOM/documentation/manual/quickguide.html

[18] http://kmi.open.ac.uk/projects/ocml/

[19] http://www.informatik.uni-freiburg.de/~dbis/florid/

[20] http://www.daml.org/

[21] World Wide Web Consortium (W3C)

majorité des SGBDs. Enfin, bon nombre d'outils CASE utilisés pour le développement d'applications et exploitant le formalisme UML s'alignent progressivement sur le standard XMI, qui est basé sur XML (*Rational Rose*[22], *Together*[23], *Tau*[24], *Objecteering*[25], *Enterprise Architecht*[26], etc.). Il est donc stratégiquement intéressant de se tourner vers XML, étant donné que le processus pour lequel nous construisons des ontologies (processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels) utilise les artefacts du développement logiciel (nécessité de « mapping ontologique »). Finalement, il existe sur le marché une panoplie d'outils de développement d'ontologies, qui permettent de générer dans le format XML les ontologies développées. Dans la suite nous examinons certains outils parmi les principaux conçus pour développement d'ontologies.

2.3.3 OUTILS DE DÉVELOPPEMENT

Les outils de développement d'ontologies qui existent sur le marché aujourd'hui sont divers et variés à bien des égards. Cet état de choses suscite beacoup d'interrogations lorsque vient le moment d'en choisir un pour construire une nouvelle ontologie [Gómez-Pérez et al. 02] : L'outil offre-t-il une assistance au développement ? Comment sont stockées les ontologies construites (dans une base de données, des fichiers XML, etc.) ? L'outil dispose-t-il d'un moteur d'inférence ? Quels langages d'ontologies l'outil supporte-t-il ? L'outil permet-il d'importer/exporter des ontologies, si oui, dans quel format ? L'outil offre-t-il un support à la réutilisation d'ontologies existantes ? L'outil offre-t-il un support à l'évaluation de la qualité des ontologies construites ? L'outil permet-il de documenter les ontologies construites ? L'outil offre-t-il un support graphique à la construction des ontologies ? L'outil est-il stable, convivial, « mature » ? L'outil est-il bien documenté ? Les réponses à toutes ces questions pourraient s'avérer décisives dans le choix de l'un ou l'autre outil. Dans cette section nous passons en revue (dans l'ordre alphabétique) les principaux outils disponibles.

[22] http://www-306.ibm.com/software/rational/

[23] http://www.togethersoft.com/products/

[24] http://www.telelogic.com/products/tau/uml/index.cfm

[25] http://www.objecteering.com/

[26] http://www.sparxsystems.com.au/

2.3.3.1 APOLLO

Apollo[27] est un outil implanté en java qui dispose de son propre langage interne de stockage d'ontologies. Il représente les connaissances suivant l'approche orientée-objet (classes, instances, fonctions, associations, etc.). La vérification des modèles se fait à chaud (en même temps qu'ils sont construits). Il permet d'exporter les ontologies construites dans différents formats.

2.3.3.2 LINKFACTORY®

LinKFactory®[28] [Ceusters et al. 01] est un système formel de gestion des ontologies, développé en java, orienté vers la construction et la gestion d'ontologies formelles de très grande taille, complexes et qui ne sont pas liées à un langage de spécification particulier. Il est multi plateformes et multi-utilisateurs. Les ontologies sont stockées dans une base de données relationnelle. Il dispose de mécanismes de « mapping » entre ontologies. Plusieurs mécanismes d'assurance qualité sont également intégrées (gestion des versions, hiérarchisation et suivi des utilisateurs, etc.).

2.3.3.3 OILED

OILEd [Bechhofer et al. 01] est un éditeur graphique d'ontologies développé en java, qui utilise le formalisme DAML+OIL. Les cadres sont également exploités pour la modélisation. Le RDF Schema de DAML+OIL est utilisé pour le chargement et le stockage des ontologies. L'outil dispose de mécanismes pour la classification et le contrôle de la cohérence des ontologies. La version 3.4 de OILEd est gratuite et disponible sur le site web de OILEd.

2.3.3.4 ONTOEDIT

OntoEdit [Sure et al. 02] est un environnement graphique d'ingénierie des ontologies (développement et maintenance), disposant d'un modèle ontologique interne. Il est disponible en versions gratuite et professionnelle. La version professionnelle inclut le contrôle de la cohérence, l'ingénierie collaborative et le partage d'ontologies.

---

[27] http://apollo.open.ac.uk/docs/user-guide.ps.gz

[28] **http://www.isi.edu/~blythe/kcap-interaction/papers/LinKFactoryXWhiteXPaperXfinal.doc**

2.3.3.5 ONTOLINGUA

Le serveur Ontolingua [Farquhar et al. 96] rassemble un ensemble d'outils et services pour supporter la construction d'ontologies partagées entre différents groupes. Il a été développé par le Knowledge Systems Laboratory (KSL) de l'université de Standford. Il supporte plusieurs langages et dispose de traducteurs permettant de passer de l'un à l'autre. Il est aussi doté d'une bibliothèque d'ontologies, accessible à distance ou localement via des éditeurs d'ontologies ou des applications.

2.3.3.6 ONTOSAURUS

Développé par le Information Sciences Institute (ISI) de l'Universite de Californie du sud, Ontosaurus [Swartout et al. 97] compte deux (2) modules : Un serveur d'ontologies pour le stockage (le formalisme Loom est utilisé) et un serveur d'édition d'ontologies qui crée dynamiquement des pages HTML pour la présentation des ontologies. Il dispose de traducteurs intégrés pour convertir les ontologies dans le formalisme KIF, Ontolingua, KRSS et C++.

2.3.3.7 KNOME

KnoME[29] [Rogers et al. 01] est un ensemble d'outils mis au point à l'Université de Manchester, permettant le développement collaboratif d'ontologies. Le formalisme utilisé pour les spécifications est le GRAIL Concept Modelling Language [Rector et al. 97], conçu pour la terminologie du domaine médical. Par conséquent, il est plus orienté vers le développement d'ontologies pour la terminologie médicale. Il dispose d'un module permettant l'acquisition rapide des connaissances des experts pour un domaine donné. La version 5.4 du paquetage qui a été implanté en Smalltalk, est gratuite et disponible sous le nom OpenKnoME à l'adresse http://www.topthing.com.

2.3.3.8 PROTÉGÉ-2000

Protégé-2000 [Noy et al. 01, Noy et al. 00] est un outil d'acquisition et gestion de connaissances développé à l'Université de Stanford. Il est gratuit et disponible à l'adresse http://protege.stanford.edu. Il est doté d'un environnement graphique et interactif pour la conception d'ontologies et le développement de bases de connaissances. C'est un système

[29] http://www.topthing.com

ouvert, modulaire. Il est extensible et ajustable : il offre la possibilité à un utilisateur d'incorporer de nouvelles fonctionnalités par ajout de « plugins » appropriés qu'il a créés. Une bibliothèque de « plugins » est prévue à cet effet). Certains «plugins » déjà développés permettent l'import/export d'ontologies existantes stockées dans le format RDF Schema, XML et DAML+OIL. D'autres «plugins » permettent d'afficher sur l'interface utilisateur de l'outil des images dans le format GIF, d'inclure le son et la vidéo. Il est également possible grâce à certains « plugins » de visualiser la base de connaissances sous différentes moutures, de fusionner des ontologies, d'inférer sur les connaissances, de gérer les versions, etc. L'outil est doté d'un moteur d'inférence et un langage d'axiomes permettant d'exprimer des contraintes sur les données.

2.3.3.9 SYMONTOX

SymOntoX[30] (Symbolic Ontology Manager XML savvy) [Missikoff et al. 02] est un prototype développé par le LEKS (Laboratory for Enterprise Knowledge and Systems - IASI-CNR) pour la gestion d'ontologies de domaine. Les concepts du domaine et les associations entre concepts sont modélisés dans le formalisme OPAL (Object, Process, and Actor modelling Language) conçu par le LEKS. Toutes les données sont stockées dans le format XML. L'outil dispose d'une interface graphique pour l'édition et la visualisation des ontologies. Il est également doté d'un composant pour la validation des ontologies suivant les axiomes définis par le formalisme OPAL.

2.3.3.10 WEBODE

WebODE[31] [Arpírez et al., 01] est un environnement d'ingénierie ontologique. Il est extensible (il permet l'ajout de nouveau services crées par les utilisateurs). Il prend appui sur la méthodologie METHONTOLOGY [Fernández-López et al., 99]. Les ontologies sont stockées dans une base de données relationnelle. Il permet l'import/export d'ontologies existantes dans le format XML et la traduction dans les formats RDF, OIL, DAML+OIL, CARIN et F-Logic. L'outil dispose d'une interface graphique pour l'édition et la visualisation des ontologies, d'un module de contrôle de cohérence et vérification de

[30] http://www.symontox.org/

[31] http://webode.dia.fi.upm.es/

contraintes, d'un moteur d'inférence, d'un module de construction d'axiomes et d'un module de documentation des ontologies. Il permet l'édition coopérative d'ontologies.

2.3.3.11 WEBONTO

WebOnto[32] [Domingue 98] est un outil de création, visualisation et édition coopérative d'ontologies, développé par le KMI (Knowledge Media Institute - Open University, England). Il est doté d'une interface graphique et permet la modélisation de tâches. Le langage de modélisation des connaissances OCML (Operational Conceptual Modelling Language) est utilisé pour la spécification des ontologies. Il dispose d'un module de contrôle de cohérence. Il est gratuit et disponible à l'adresse http://webonto.open.ac.uk.

Dans le cadre de la formalisation ontologique des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels, nous préconisons l'outil Protégé 2000. La flexibilité, l'extensibilité, la popularité de cet outil, ainsi que la disponibilité de la documentation associée, ont guidé notre choix. Le fait qu'il permette l'import/export d'ontologies dans le format RDF Schéma, XML et DAML+OIL, constitue également un avantage certain. De plus, l'outil est doté d'un moteur d'inférence et d'un langage d'axiomes permettant d'exprimer des contraintes sur les connaissances décrites à travers les ontologies. Malheureusement, au moment où nous avons produit les ontologies présentées dans la thèse nous ne disposions pas de l'outil. L'outil nous aurait permis d'aboutir beaucoup plus facilement et plus rapidement aux ontologies dans divers formats standards et exploitables dans certains SGBDs[33] (nous pensons au format XML par exemple). Nous nous sommes contentés de l'outil *Power Designer* de Sybase qui permet de générer du script SQL[34] pour SGBDs relationnels. Ainsi nous n'avons pas tiré profit des facilités et possibilités offertes par l'outil. Dans la section qui suit nous abordons la question de formalisation ontologique des procédures de mesure.

[32] http://webonto.open.ac.uk

[33] Systèmes de Gestion de Bases de Données

[34] Structured Query Language

3 FORMALISATION ONTOLOGIQUE DES PROCÉDURES DE MESURE

Abran relève à juste titre dans sa thèse [Abran 94], que «les mesures doivent être associées à un processus de modélisation ». Nous sommes persuadés du besoin de contribution des sciences de la cognition en vue d'améliorer les systèmes de mesure, notamment en ce qui a trait à la représentation, la modélisation des « connaissances » associées aux méthodes de mesure (leurs procédures de mesure plus précisément) ou à l'objet de la mesure (c'est-à-dire le logiciel). Comme le fait remarquer Jean-Louis Lemoigne [Lemoigne 90], les modèles résultants d'un travail de modélisation sont «*susceptibles de rendre intelligible un phénomène perçu complexe, et d'amplifier le raisonnement de l'acteur projetant une intervention délibérée au sein du phénomène…* ». Ainsi, relativement au « design » des méthodes de mesure, et dans l'optique d'apporter un plus à l'« intelligibilité » du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels, **nous jetons dans cette thèse (à travers un travail de conceptualisation et de modélisation), les bases d'une plate-forme consensuelle pour la structuration, la représentation, l'échange et l'interprétation d'information, de « choses » (« concepts ») associées aux procédures de mesure de la taille fonctionnelle des logiciels. Il s'agit en fait des bases d'un corpus de « connaissances » associées au processus d'application de chaque méthode de mesure de la taille fonctionnelle des logiciels**. L'approche adoptée pour la spécification d'un tel corpus de « connaissances » est l'approche ontologique : Nous parlons de **formalisation ontologique des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels**. Nous partons du principe selon lequel les ontologies permettent non seulement de capturer mais aussi d'expliciter les connaissances relatives à un domaine ou un processus, aboutissant à une sorte de plateforme consensuelle sur la compréhension du domaine/processus, offrant des perspectives de réutilisation, de partage, d'échange et d'opérationnalisation desdites connaissances. Le principe général d'une telle formalisation fait l'objet de la section suivante.

3.1 PRINCIPE GENERAL DE LA FORMALISATION ONTO LOGIQUE DES PROCÉDURES DE MESURE

La formalisation ontologique de la procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels consiste en la production de deux (2)

ontologies (suivant les définitions fournies par Mizoguci [Mizoguchi et al. 00, Mizoguchi 02]) associées à la procédure :

- Une **ontologie de domaine** qui spécifie formellement et explicitement tous les concepts référencés dans la procédure en précisant les liens entre eux
- Une **ontologie de tâches** qui spécifie formellement et explicitement toutes les tâches identifiées dans la procédure en précisant les liens entre elles (hiérarchies de tâches)

La production de ces ontologies passe par une analyse des procédures de mesure associées aux méthodes considérées, des échanges avec des experts des méthodes et des expériences personnelles de mesure avec les méthodes. De l'analyse des procédures de mesure résulte la liste des concepts pertinents de mesure et des tâches de mesure. Les définitions des concepts identifiés sont extraites des manuels de mesure. Les liens entre concepts et entre tâches de mesure sont ensuite identifiés. Les concepts et liens entre concepts sont alors représentés dans un diagramme de classes UML, tandis que la hiérarchie des tâches est représentée dans un diagramme de composants UML. Finalement, la documentation des concepts et des tâches est fournie dans un moule inspiré de la méthodologie CommonKADS [Schreiber et al. 99]. Nous exploitons la façon dont les tâches, concepts et relations entre concepts sont décrits dans la méthodologie.

Une version XMI (XML Metadata Interchange) des ontologies est fortement suggérée. Des outils appropriés de production d'ontologies, tels que ceux recensés plus haut, peuvent être exploités à cet effet. Ceci pourrait s'avérer utile pour des outils d'analyse et de documentation des résultats de mesure, ou encore pour des outils d'assistance à la validation des résultats de mesure. L'aide des experts des méthodes considérées est toujours nécessaire et souhaitée pour le suivi et la maintenance des ontologies.

Comme piste de recherche future, nous pensons à une contribution de la théorie de la certitude de Stanford [Luger 02] dans la prise en compte de l'incertain qui est une réalité dans la mesure, notamment lors de l'identification de certains éléments de mesure dans les spécifications. Ceci permettrait d'associer formellement un **intervalle de confiance** aux mesures. En effet, la précision des résultats de mesure fournis par les outils de mesure et même par les mesureurs humains, est une préoccupation pour bon nombre d'utilisateurs de résultats de mesure, et pas seulement dans le domaine du logiciel. Dans le contexte de la mesure de la taille fonctionnelle des logiciels, notre champ d'intérêt, nous proposons

d'associer à chaque identification d'élément de mesure dans les spécifications, un degré de certitude. Un tel degré de certitude est intimement lié à la qualité des documents de spécification, à la granularité des spécifications, etc. Un outil diagnostic tel que celui proposé par Desharnais dans sa thèse [Desharnais 04], pourrait être utilisé pour produire ces degrés de certitude. La précision associée à un résultat de mesure serait obtenue par combinaison de ces degrés de certitude. La combinaison tiendrait compte des liens existants entre les éléments de mesure identifiés. La théorie de la certitude de Stanford par exemple, nous offrirait un cadre théorique pour une telle combinaison.

En attendant, il appert que la formalisation ontologique des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels, ouvre de nouvelles perspectives pour la mesure. Nous les examinons dans la section suivante.

3.2 DE NOUVELLES PERSPECTIVES POUR LA MESURE

Dans les manuels de mesure des principales méthodes de mesure de la taille fonctionnelle des logiciels, le langage naturel est utilisé pour décrire les procédures de mesure ou encore certains éléments qui leur sont associés (concepts, tâches,…). Il en résulte une faible uniformité dans l'interprétation lors de l'application des méthodes [Nishiyama 99], surtout parmi les non experts. Une présentation plus formelle, laissant moins de place aux ambiguïtés serait bénéfique. De plus, le développement d'outils de mesure pour une méthode donnée requiert présentement l'assistance d'experts de la méthode, en particulier pour étayer certains éléments significatifs qui ne sont explicites dans les manuels (certaines règles de mesure, certains liens entre concepts de mesure …). Des ontologies claires et spécifiques (suivant les définitions fournies par Mizoguci [Mizoguchi et al. 00, Mizoguchi 02]) pourraient aider à réduire de façon significative une telle dépendance. De telles ontologies présentées dans un formalisme approprié synthétiseraient le gros des connaissances associées aux procédures de mesure.

Un tel corpus de «connaissances» associées au processus d'application de chaque méthode de mesure de la taille fonctionnelle des logiciels pourrait donc être exploité non seulement dans le cadre de l'aide à une meilleure de la compréhension des méthodes de mesure, mais aussi dans le cadre du développement d'outils d'aide à la mesure, notamment

les outils d'automatisation d'une bonne partie du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels.

XML (Extended Markup Language) étant approprié pour l'échange de documents, les bases de données historiques de projets logiciels pourraient être étendues pour stocker des données plus détaillées sur la taille fonctionnelle des projets. Par exemple, si nous considérons le système qu'utilise l'ISBSG (International Software Benchmarking Standards Group) pour la collection des données historiques sur des projets logiciels, plutôt que de stocker uniquement des descriptions textuelles et des valeurs numériques indiquant la taille fonctionnelle des projets, l'on pourrait stocker dans le format XML (suivant un moule XMI défini), certaines données sur les projets (notamment les modèles et données relatifs à la taille fonctionnelle). De plus, il est possible d'envisager les échanges de données entre outils de mesure de la taille fonctionnelle des logiciels (par exemple dans le cas où l'on a plusieurs outils de mesure pour la même méthode de mesure).

Finalement, une nouvelle approche pour s'attaquer à la question de convertibilité inter-méthodes est envisagée. Cette approche est également orientée ontologie: les liens entre ontologies de domaine associées aux procédures de mesure pour les méthodes considérées, doivent être formellement établis. Les travaux actuels sur les liens entre ontologies seront mis à contribution [Madhavan et al. 02]. Ce travail est déjà en chantier. COSMIC-FFP et MkII-FPA ont été sélectionnées pour les tests.

Dans la suite, nous présentons quelques études de cas de formalisation ontologique de procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels. Nous avons choisi les trois (3) méthodes devenues standard ISO en 2003 (FPA, MkII-FPA et COSMIC-FFP).

3.3 ETUDES DE CAS (FPA, MKII-FPA, COSMIC-FFP)

A la date d'aujourd'hui, nous avons produit les ontologies associées aux processus d'application des méthodes COSMIC-FFP [Bevo et al. 03a], MkII-FPA [Bevo et al. 03b] et FPA [Bevo et al. 03c].

3.3.1 FORMALISATION ONTOLOGIQUE DE LA PROCÉDURE DE MESURE ASSOCIÉE À LA MÉTHODE DE MESURE COSMIC-FFP

3.3.1.1 ONTOLOGIE DE DOMAINE ASSOCIÉE À LA PROCÉDURE DE MESURE DE COSMIC-FFP

Le manuel de mesure ISO de COSMIC-FFP v2.2 [Abran et al. 03, ISO/IEC 03a], ainsi que notre expérience personnelle de mesure à l'aide de COSMIC-FFP, ont été nécessaires pour produire cette ontologie. Nous avons procédé à une analyse détaillée de la procédure de mesure associée à COSMIC-FFP. Tous les concepts jugés pertinents et associés à la procédure de mesure ont été identifiés et leurs définitions respectives extraites du manuel de mesure. Les liens entre les concepts ont été explicitement déterminés. Finalement, les concepts identifiés et liens entre concepts ont été représentés dans le formalisme choisi (formalisme orienté-objet [diagramme de classes UML [Booch et al. 99]). Pour la documentation des concepts et liens entre concepts, la méthodologie CommonKADS a été mise à contribution, avec l'approche de description des concepts et liens entre concepts qui y est proposée. L'aide d'experts COSMIC-FFP est toujours requise pour raffiner et consolider ce travail.

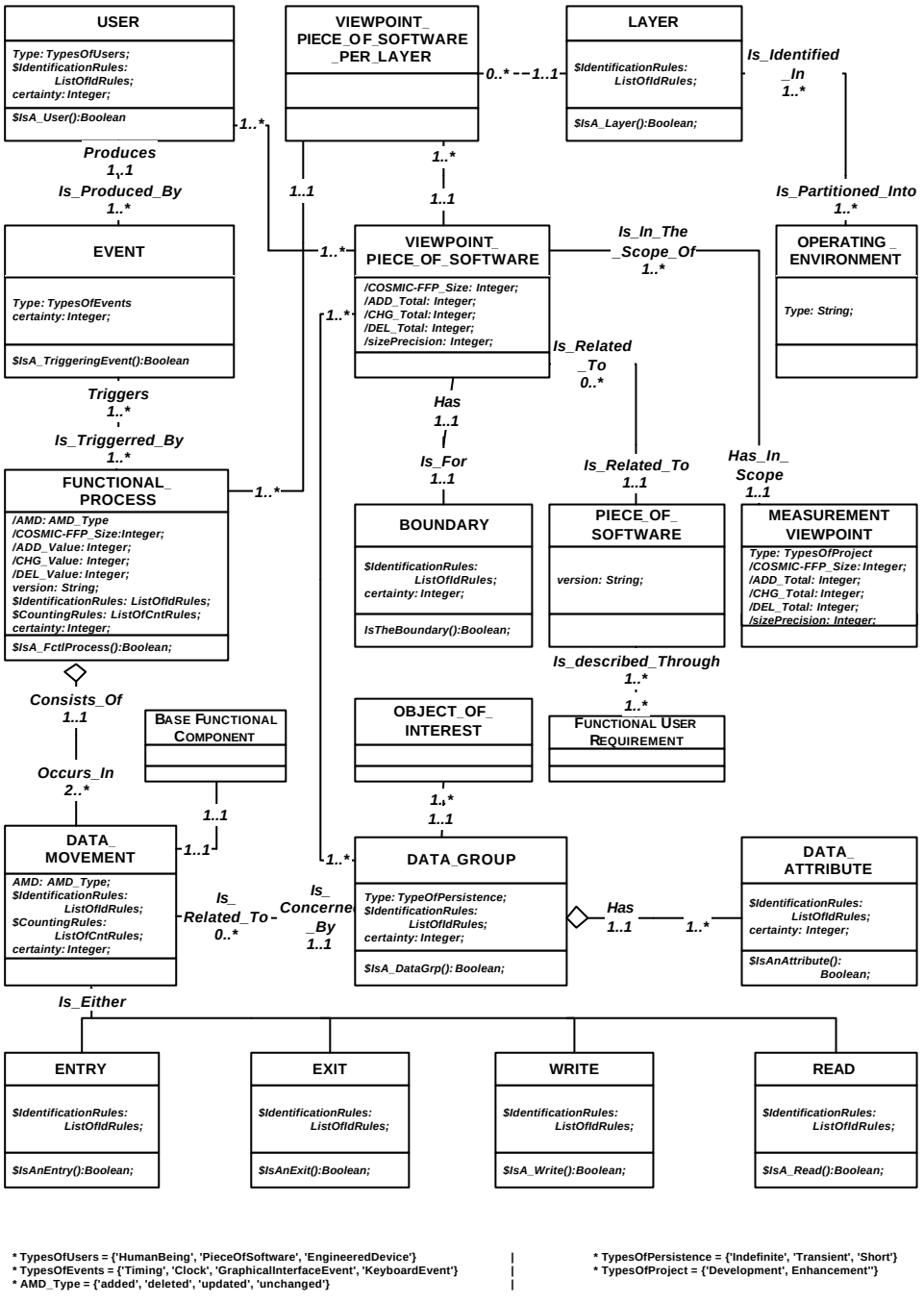


Figure 16: Ontologie de domaine associée à la procédure de mesure de COSMIC-FFP (*Adaptée de [Bevo et al 01]*)

Ci-dessous, nous présentons un exemple de description de concepts apparaissant dans la Figure 16.

- **Concepts**

NB: Toutes les définitions sont extraites du manuel de mesure ISO de COSMIC-FFP v2.2 [Abran et al. 03, ISO/IEC 03a].

F.1 Environnement opérationnel (Operating_Environment)

Définition:

*L'**environnement opérationnel** d'un logiciel désigne l'ensemble des logiciels opérant concurremment avec ledit logiciel sur un système informatique donné.*

Propriétés:

- *Type : {String};*
- *Description : {String};*

F.2 Couche (Layer)

Définition:

*Une **couche** dans un environnement logiciel résulte du partitionnement fonctionnel dudit environnement logiciel tel que tous les processus fonctionnels inclus sont considérés comme étant à un même niveau d'abstraction.*

Propriétés:

- *Name : {String};*
- *Description : {String};*
- *IdentificationRules : {IdRule}*

***Concept:** IdRule*

Condition: {expression logique}

Action: {procédure}

F.3 Morceau de logiciel (Piece of software)

Définition:

*Un **morceau de logiciel** est une partie d'un logiciel. Un logiciel étant considéré comme un ensemble d'instructions, de données, de procédures et éventuellement de documentation, opérant comme un Tout en vue de combler un ensemble de besoins spécifiques, lesquelles spécifications peuvent être décrites d'un point de vue fonctionnel à travers un ensemble fini de FUR (Requis Fonctionnels d'Utilisateurs), de requis d'ordre technique ou qualitatif.*

Propriétés:

- *name : {String};*
- *Description: {String};*
- *Version : {String};*

F.4 ProcessusFonctionnel (Functional Process)

Définition:

*Un **processus fonctionnel** est un composant élémentaire d'un FUR (Requis Fonctionnels d'Utilisateurs) comprenant un ensemble unique, cohésif et indépendamment exécutables de mouvement de données. Il est déclenchée par un ou plusieurs événements déclencheurs soit directement, soit indirectement via un acteur. Son exécution s'achève lorsqu'elle a accompli tout ce qui doit être fait en réponse à un événement déclencheur.*

Propriétés:

- *AMD (AddedModifiedDeleted) : {AMD_Type}; /* AMD_Type = {'added', 'deleted', 'updated'} */*
- *Title : {String};*
- *Description : {String};*
- *Version : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *MeasurementRule[ ] : {CntRule}*
- *Certainty : {Decimal}; /* Cette valeur indique le degré de certitude qu'a le mesureur dans l'identification du processus fonctionnel. Elle sera utilisée dans le calcul de la précision associée à la taille fonctionnelle du logiciel. */*
- *ADD_Value : {Integer}; /* Cette valeur indique le nombre de points de fonction comptabilisés relatifs à l'ajout de fonctionnalités pour le processus fonctionnel (projet de développement ou de maintenance) */*

- *CHG_Value : {Integer}; /* Cette valeur indique le nombre de points de fonction comptabilisés relatifs à la mise à jour de fonctionnalités, pour le processus fonctionnel (projet de maintenance) */*
- *DEL_Value : {Integer}; /* Cette valeur indique le nombre de points de fonction comptabilisés relatifs à la suppression de fonctionnalités, pour le processus fonctionnel (projet de maintenance) */*
- *COSMIC-FFP_Size : {Integer}; /* Cette valeur indique le nombre total de points de fonction comptabilisés pour le processus fonctionnel */*

Concept: *IdRule*

Condition: {expression logique}

Action: {procédure}

Concept: *CntRule*

Condition: {expression logique}

Action: {procédure}

NB: Tous les autres concepts sont décrits en annexe B de ce document.

Un exemple de description de liens entre concepts apparaissant dans la Figure 16, est proposé dans la section suivante.

G. Liens entre concepts

G.1 Environnement Opérationnel – Couche (Operating_Environment - Layer)

Description:

La partition d'un environnement opérationnel en couches.

Arguments:

Environnement Opérationnel (Operating_Environment)

Rôles: *Le nombre d'environnements opérationnels dans lesquels l'on retrouve une couche donnée [l'on pourrait retrouver une couche donnée dans plusieurs environnements opérationnels]*

Cardinalités: *min 1; max n;*

Couche (Layer)

Rôles: *Le nombre de couches résultant de la partition d'un environnement opérationnel donné [Un environnement opérationnel pourrait être partitionné en plusieurs couches]*

Cardinalités: *min 1; max n;*

G.2 Morceau de logiciel – FUR (Piece Of Software - FUR)

Description:

Les FURs (Requis Fonctionnels d'utilisateur) pour un morceau de logiciel.

Arguments:

Morceau de logiciel (Piece Of Software)

Rôles: *Le nombre de morceaux de logiciel implantant un FUR donné [Un FUR pourrait être implanté dans plusieurs morceaux de logiciel (par exemple plusieurs versions d'un même morceau de logiciel)]*

Cardinalités: *min 1; max n;*

FUR (Requis Fonctionnel d'Utilisateur)

Rôles: *Le nombre de FURs implantées dans un morceau de logiciel donné [Un morceau de logiciel pourrait implanter plusieurs FURs]*

Cardinalités: *min 1; max n;*

G.3 Groupe de Données– Objet d'Intérêt (Data_Group - Object_Of_Interest)

Description:

Les objets d'intérêt associés à un groupe de données.

Arguments:

Groupe de Données (Data_Group)

Rôles: *Le nombre de groupes de données associés à un objet d'intérêt [Un objet d'intérêt correspond à un groupe de données au maximum]*

Cardinalités: *min 1; max 1;*

Objet d'Intérêt (Object_Of_Interest)

Rôles: *Le nombre d'objets d'intérêt associés à un groupe de données [Un groupe de données peut être issu du regroupement de plusieurs objets d'intérêt]*

Cardinalités: *min 1; max n;*

G.4 Mouvement de Données– BFC (Data_Movement - BFC)

Description:

Pour la méthode COSMIC-FFP un BFC (Composant fonctionnel de Base) correspond à un mouvement de données.

Arguments:

Mouvement de Données (Data_Movement)

***Rôles**: Le nombre de mouvements de données par BFC [Un BFC correspond à un mouvement de données]*

***Cardinalités:** min 1; max 1;*

BFC (Base Functional Component)

***Rôles:** Le nombre de BFCs pour un mouvement de données. [Un mouvement de données correspond à un BFC]*

***Cardinalités:** min 1; max 1;*

G.5 Morceau de logiciel – Perspective de mesure (Viewpoint_Piece_Of_Software)

Description:

Partie d'un morceau de logiciel devant être prise en compte dans une perspective de mesure.

Arguments:

Perspective de mesure (Measurement_Viewpoint)

***Rôles**: Le nombre perspectives de mesure incluant un morceau de logiciel donné [Un morceau de logiciel peut faire partie entièrement ou partiellement de plusieurs perspectives de mesure]*

***Cardinalités:** min 0; max n;*

Morceau de logiciel (Piece_Of_Software)

***Rôles:** Le nombre de morceaux de logiciel associés à une perspective de mesure donnée. [A une perspective de mesure l'on peut associer plusieurs morceaux de logiciel]*

***Cardinalités:** min 1; max n;*

NB: Tous les autres liens entre concepts sont décrits en annexe B de ce document.

La section suivante est consacrée à un exemple d'ontologie de tâches associée à une procédure de mesure.

3.3.1.2 ONTOLOGIE DE TÂCHES ASSOCIÉE À LA PROCÉDURE DE MESURE DE COSMIC-FFP

Le manuel de mesure ISO de COSMIC-FFP v2.2 [Abran et al. 03, ISO/IEC 03a], ainsi que notre expérience personnelle de mesure à l'aide de COSMIC-FFP, ont été nécessaires pour produire cette ontologie. Nous avons procédé à une analyse détaillée de la procédure de mesure associée à COSMIC-FFP. Toutes les tâches jugées pertinentes et associées à la procédure de mesure ont été identifiées. Les liens entre les tâches ont été explicitement déterminés en vue de produire une hiérarchie de tâches. Finalement, les tâches identifiées et liens entre tâches ont été représentés dans le formalisme choisi (formalisme orienté-objet [diagramme de composants UML] [Booch et al. 99]). Le diagramme d'activités UML a été également utilisé pour la spécification des tâches. Pour la documentation des tâches et liens entre tâches, la méthodologie CommonKADS a été mise à contribution, avec l'approche de description des tâches et liens entre tâches qui y est proposée. L'aide d'experts COSMIC-FFP est toujours requise pour raffiner et consolider ce travail.

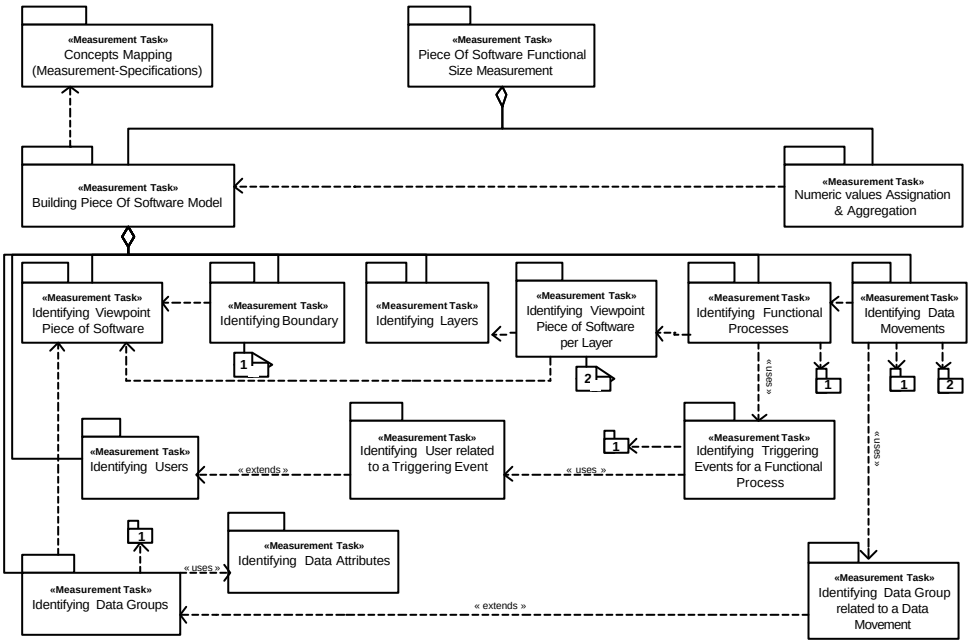


Figure 17 : Hiérarchie de tâches associée à la procédure de mesure de COSMIC-FFP (*Basée sur [Bevo et al 01]*)

Ci-dessous, nous présentons un exemple de description de tâches apparaissant dans la Figure 17.

H. Les tâches

H.1 Mesure de la taille fonctionnelle d'un morceau de logiciel (Piece of software Functional Size Measurement)

Définition :

Finalité : *Etant données un ensemble de spécifications pour un morceau de logiciel un environnement opérationnel et une perspective de mesure, la tâche a pour finalité la production d'un modèle COSMIC-FFP pour le morceau de logiciel et la détermination de sa taille fonctionnelle en unités de taille fonctionnelle COSMIC-FFP (CFPUs). Cette tâche se décompose en deux (2) sous tâches complémentaires: « Construction Modèle Morceau Logiciel » (« Building Piece of Software Model »), et « Assignation de valeurs numériques & Agrégation » (« Numeric Values Assignment/Association & Aggregation »).*

Entrées : *Ensemble_de_specifications_pour_un_morceau_de_logiciel,*
Environnement_operationnel,
Perspective_de_la_mesure,
Ensemble_de_correspondances_specifications_mesure

Sorties : *Modele_COSMIC-FFP_pour_le_morceau_de_logiciel,*
Taille_fonctionnelle_du_morceau_de_logiciel,
[précision/certitude_globale_pour_la_mesure]

Corps :

Type : *composition*

Sous-Tâches : *« Construction Modèle Morceau Logiciel » (« Building Piece of Software Model »), et « Assignation de valeurs numériques & Agrégation » (« Numeric Values Assignment/Association & Aggregation »)*

Exécution : *exécuter séquentiellement :*

- *Construction_Modele_Morceau_Logiciel*
  (+Ensemble_de_specifications_pour_un_morceau_de_logiciel,
  +Perspective_de_la_mesure,
  +[Environnement_Operationne]l,
  +Ensemble_de_correspondances_specifications_mesure,
  -Modele_COSMIC-FFP_pour_le_morceau_de_logiciel),

- *Assignation_Valeurs_Numériques_Agrégation (+Modele_COSMIC-FFP_pour_le_morceau_de_logiciel, -Taille_fonctionnelle_du_morceau_de_logiciel, -précision/certitude_globale_pour_la_mesure)*

H.2 Construction du modèle COSMIC-FFP d'un morceau de logiciel (Building a COSMIC-FFP Piece of software model)

Définition:

Finalité : *Etant données un ensemble de spécifications pour un morceau de logiciel un environnement opérationnel, une perspective de mesure et un ensemble de correspondances entre concepts de mesure et concepts d'un langage de specification, la tâche a pour finalité la production d'un modèle COSMIC-FFP pour le morceau de logiciel. Cette tâche se décompose en huit (8) sous tâches complémentaires: « Identification des couches » (« Identifying Layers »), « Identification Morceaux Logiciel Relatifs a une Perspective de Mesure » (« Identifying Viewpoint Piece Of Software »), « Identification Frontières» (« Identifying boundary »), « Identification Morceaux Logiciel Relatifs a une Perspective de Mesure pour chaque Couche » (« Identifying Viewpoint Piece Of Software Per Layer »), « Identification Utilisateurs » (« Identifying users »), « Identification Groupes de Données » (« Identifying Data Groups »), « Identification Processus Fonctionnels » (« Identifying Functional Processes ») et « Identification Mouvements de Données » (« Identifying data movements »). Cette tâche utilise les extrants de la tâche « Mise en Correspondance de Concepts » (« Concepts Mapping »)*

Entrées : *Ensemble_de_specifications_pour_un_morceau_de_logiciel,*
Environnement_operationnel,
Perspective_de_la_mesure,
Ensemble_de_correspondances_specifications_mesure

Sorties : *Modele_COSMIC-FFP_pour_le_morceau_de_logiciel*

Corps :

Type : *composition*

Sous-Tâches : *« Identification des couches » (« Identifying Layers »), « Identification Morceaux Logiciel Relatifs a une Perspective de Mesure » (« Identifying Viewpoint Piece Of Software »), « Identification Frontières» (« Identifying boundary »), « Identification Morceaux Logiciel Relatifs a une Perspective de Mesure pour chaque Couche » (« Identifying Viewpoint Piece Of Software Per Layer »), « Identification Utilisateurs » (« Identifying users »), « Identification Groupes de Données » (« Identifying Data Groups »), « Identification Processus Fonctionnels » (« Identifying Functional Processes »), « Identification Mouvements de Données » (« Identifying data movements »).*

Exécution : *exécuter séquentiellement :*

- *Identification_Couches (*
  +Environment_Operationnel,
  -Couches[ ])
- *Identification_Morceaux_Logiciel_Relatifs_a_une_Perspective_Mesure(*
  +Ensemble_de_specifications_pour_un_morceau_de_logiciel,
  +Perspective_de_la_mesure,
  -Morceaux_Logiciel_Relatifs_a_une_Perspective_Mesure)
- *Identification_Frontieres (*
  +Morceaux_Logiciel_Relatifs_a_une_Perspective_Mesure,
  -Frontieres)
- *Identification_Morceaux_Logiciel_Relatifs_a_Perspective_Mesure_Par_Couche(*
  + Morceaux_Logiciel_Relatifs_a_une_Perspective_Mesure,
  +Couches [ ],
  -
  Morceaux_Logiciel_Relatifs_a_une_Perspective_Mesure_Par_Couche[ ])
- *Identification_Utilisateurs(*
  +Morceaux_Logiciel_Relatifs_a_une_Perspective_Mesure_Par_Couche,
  +Ensemble_de_correspondances_specifications_mesure,
  -Utilisateurs[ ])
- *Identification_Groupes_Donnees (*
  +
  Morceaux_Logiciel_Relatifs_a_une_Perspective_Mesure_Par_Couche,

+Frontieres,
+ Ensemble_de_correspondances_specifications_mesure,
-Groupes_Donnees[ ])

- *Identification_Processus_Fonctionnels(*
  +Morceaux_Logiciel_Relatifs_a_une_Perspective_Mesure_Par_Couche,
  +Frontieres,
  + Ensemble_de_correspondances_specifications_mesure,
  - Processus_Fonctionnels[ ])
- *Identification_Movements_Donnees(*
  +Morceaux_Logiciel_Relatifs_a_une_Perspective_Mesure_Par_Couche,
  +Frontieres,
  + Ensemble_de_correspondances_specifications_mesure,
  +Processus_Fonctionnels[ ]
  -Movements_Donnees[])

H.3 Assignation de valeurs numériques et agrégations (Numeric Values Assignment/Association & Aggregation)

Définition :

Finalité : *Etant donné un modèle COSMIC-FFP d'un morceau de logiciel, la tâche a pour finalité l'assignation de valeurs numériques à certains éléments du modèle (les mouvements de données en loccurrence) et l'agrégation des valeurs assignées en vue de déterminer la taille fonctionnelle du morceau de logiciel en Cfsu (Cosmic Functional Size Units) suivant le point de vue considérée. L'agrégation doit tenir compte du paramètre d'incertitude associé à certains éléments du modèle (processus fonctionnels, groupes de données, mouvements de données, etc.), en vue de déterminer la précision associée au résultat de mesure.*

Entrées : *Modele_COSMIC-FFP_pour_le_morceau_de_logiciel*

Sorties :*Taille_fonctionnelle_du_morceau_de_logiciel,*
[précision/certitude_globale_pour_la_mesure]

Corps :

Type : *composition*

Sous-Tâches : *« Assignation_Valeurs_Numériques » (« Numeric values Assignment/Association »), « Agrégation » (« Aggregation »)*

« Construction Modèle Application » (« Building Application Model »), «Calcul Taille Fonctionnelle Application» (« Calculating Application Functional size »)

Exécution : *exécuter séquentiellement :*

- *Assignation_Valeurs_Numériques_Agrégation(*
  +Modele_COSMIC-FFP_pour_le_morceau_de_logiciel,
  -ModelePlus_COSMIC-FFP_pour_le_morceau_de_logiciel)
- *Agregation (*
  +ModelePlus_COSMIC-FFP_pour_le_morceau_de_logiciel,
  -Taille_fonctionnelle_du_morceau_de_logiciel,
  -précision/certitude_globale_pour_la_mesure)

H.4 Mapping de concepts (Concepts Mapping)

Définition :

Finalité : *Etablir des correspondances entre concepts de mesure et concepts d'un langage de spécification (par exemple UML).*

Entrées : *Concepts COSMIC-FFP, Concepts d'un langage de spécifications*

Sorties : *Ensemble_de_correspondances_specifications_mesure*

Corps :

Type : *simple*

Sous-Tâches : *;*

Exécution : *exécuter:*

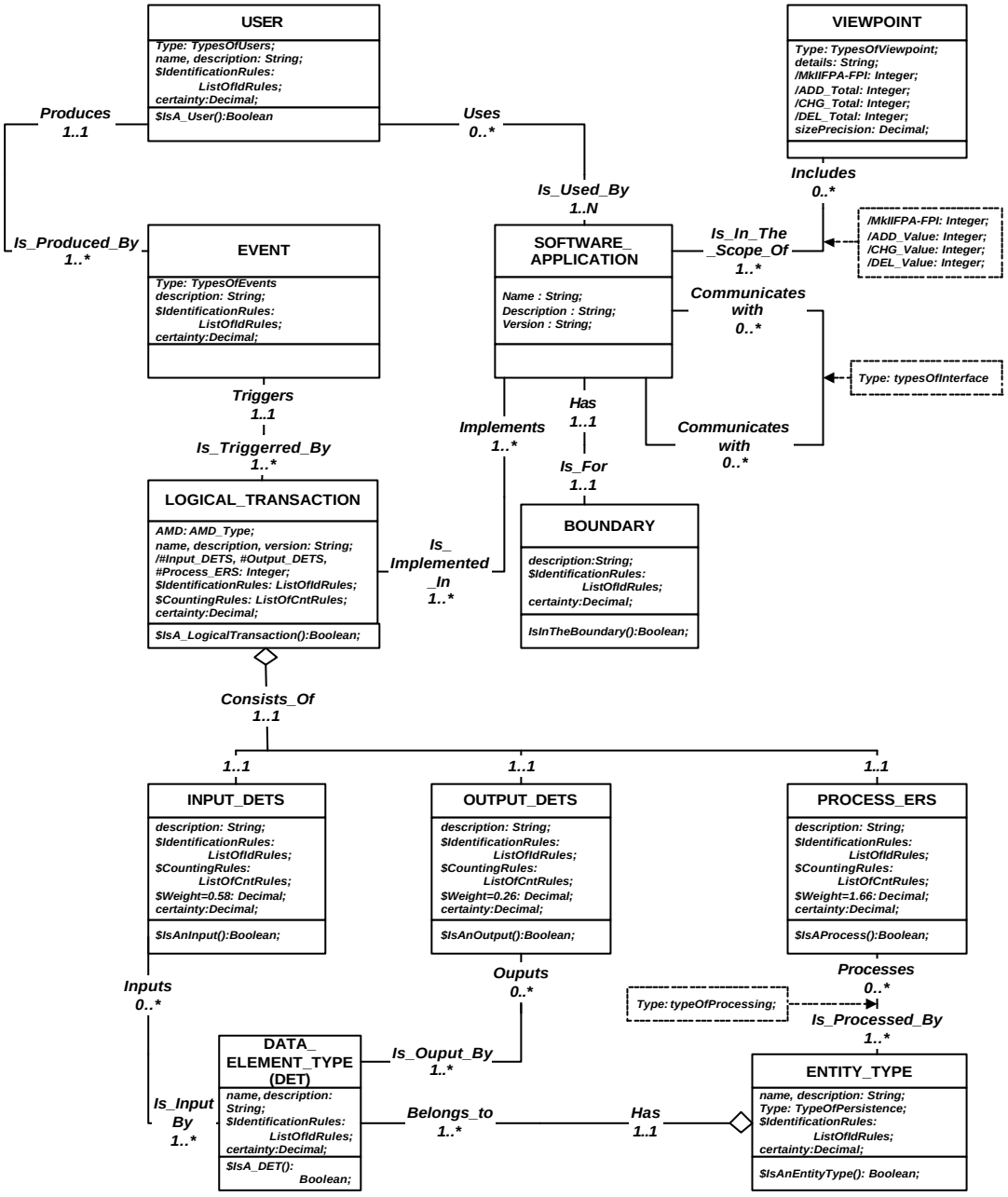
- *Etablir_Correspondances (*
  +Concepts_COSMIC-FFP[ ],
  +Concepts_Langage_Spécifications[ ],
  -Ensemble_de_correspondances_specifications_mesure[ ])

NB: Toutes les autres tâches sont décrites en annexe B de ce document.

3.3.2 FORMALISATION ONTOLOGIQUE DE LA PROCÉDURE DE MESURE ASSOCIÉE À LA MÉTHODE DE MESURE MKII-FPA

3.3.2.1 ONTOLOGIE DE DOMAINE ASSOCIÉE À LA PROCÉDURE DE MESURE DE MKII-FPA

Le manuel de mesure de MkII-FPA version 1.3.1 [UKSMA 00, ISO/IEC 02], ainsi que notre expérience personnelle de mesure à l'aide de MkII-FPA, ont été nécessaires pour produire cette ontologie. Nous avons procédé à une analyse détaillée de la procédure de mesure associée à MkII-FPA. Tous les concepts jugés pertinents et associés à la procédure de mesure ont été identifiés et leurs définitions respectives extraites du manuel de mesure. Les liens entre les concepts ont été explicitement déterminés. Finalement, les concepts identifiés et liens entre concepts ont été représentés dans le formalisme choisi (formalisme orienté-objet [diagramme de classes UML [Booch et al. 99]). Pour la documentation des concepts et liens entre concepts, la méthodologie CommonKADS a été mise à contribution, avec l'approche de description des concepts et liens entre concepts qui y est proposée. L'aide d'experts MkII-FPA est toujours requise pour raffiner et consolider ce travail.



*** TypesOfUsers = {'HumanBeing', 'PieceOfSoftware', 'EngineeredDevice'}**
*** TypesOfEvents = {'Timing', 'Clock', 'GraphicalInterfaceEvent', 'KeyboardEvent'}**
*** TypesOfViewpoint = {'Project Viewpoint', 'Application Manager Viewpoint', 'Business Enterprise Viewpoint'}**
*** AMD_Type = {'Added', 'Deleted', 'Updated'}**
*** TypesOfProcessing = {'Create', 'Delete', 'Update', 'Read'}**

Figure 18 : Ontologie de domaine associée a la procédure de mesure de MkII-FPA (*Adaptée de [Bevo et al 01]*)

Ci-dessous, nous présentons un exemple de description de concepts apparaissant dans la Figure 18.

- **Concepts**

NB: Toutes les définitions sont extraites du manuel de mesure MkII-FPA, version 1.3.1 [UKSMA 00, ISO/IEC 02].

H.5 Frontière (Boundary)

Définition:

*La **frontière** pour un logiciel donné fait référence à l'interface conceptuelle entre le logiciel et ses utilisateurs. Elle détermine les fonctions à inclure dans le comptage des points de fonction et celles qu'il faut exclure.*

Propriétés:

- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty: {Décimal}; /* Cette valeur indique la certitude du mesureur pour cet élément de fonctionnalité (c'est-à-dire la certitude que le mesureur a qu'il s'agit de la frontière pour l'application mesurée). Elle sera utilisée dans le calcul de la précision associé e à la taille fonctionnelle de l'application mesurée. */*

***Concept:** IdRule*

Condition: {expression logique}

Action: {procédure}

H.6 Transaction logique (Logical Transaction)

Définition:

*Une **transaction logique** est un composant fonctionnel de base pour la méthode Mk II FPA. C'est la plus petite unité complète et significative de traitement d'information pour les utilisateurs dans un secteur d'activités. Elle est déclenchée par un événement du monde réel ou par une requête. Elle est constituée d'une entrée, d'un traitement et d'une sortie. Elle doit laisser l'application à laquelle elle appartient dans un état cohérent.*

Propriétés:

- *AMD (AddedModifiedDeleted) : {AMD_Type}; /* AMD_Type = {'added', 'deleted', 'updated'} */*
- *name : {String};*
- *Description: {String};*

- *Version : {String};*
- *Certainty : {Décimal}; /* Cette valeur indique la certitude du mesureur pour cet élément de fonctionnalité (c'est-à-dire la certitude que le mesureur a qu'il s'agit d'une transaction logique pour l'application mesurée). Elle sera utilisée dans le calcul de la précision associée à la taille fonctionnelle de l'application mesurée. */*
- *MkII-FPA_Size : {Integer}; /* Cette valeur indique le nombre total de points de fonction comptabilisées pour cette transaction logique */*
- *IdentificationRule[]: {IdRule}*
- *MeasurementRule[]: {CntRule}*

Concept: *IdRule*

Condition: {expression logique}

Action: {procédure}

Concept: *CntRule*

Condition: {expression logique}

Action: {procédure}

H.7 Application logicielle (Software Application)

Définition:

*Une **application logicielle** est un ensemble d'instructions, de données, de procédures et éventuellement de documentation, opérant comme un Tout en vue de combler un ensemble de besoins spécifiques, lesquelles spécifications peuvent être décrites d'un point de vue fonctionnel à travers un ensemble fini de FUR (Requis Fonctionnels d'Utilisateurs), de requis d'ordre technique ou qualitatif.*

Propriétés:

- *name : {String};*
- *Description: {String};*
- *Version : {String};*

NB: Tous les autres concepts sont décrits en annexe C de ce document.

Un exemple de description de liens entre concepts apparaissant dans la Figure 18, est proposé dans la section suivante.

I. Liens entre concepts

I.1 Perspective de mesure – Application_logicielle (Viewpoint – Software_Application)

Description:

Applications logicielles pour une perspective de mesure déterminée.

Arguments:

Application_logicielle (Software_Application)

Rôles: *Le nombre d'applications logicielles associées à une perspective de mesure déterminée. [A une perspective de mesure donnée l'on peut associer plusieurs applications logicielles]*

Cardinalités: *min 1; max n;*

Perspective (Viewpoint)

Rôles: *Le nombre de perspectives de mesure auxquelles est associée une application logicielle donnée. [Une application logicielle donnée peut être associée à plus d'une perspective de mesure]*

Cardinalités: *min 0; max n;*

I.2 Application logicielle – Frontière (Software_Application – B oundary)

Description:

Frontière de mesure pour une application logicielle.

Arguments:

Frontière de mesure (Boundary)

Rôles*: Le nombre frontières pour une application logicielle donnée [A une application logicielle donnée est associée une et une seule frontière de mesure]*

Cardinalités: *min 1; max 1;*

Application logicielle (Software_Application)

Rôles: *Le nombre d'applications logicielles associées à une frontière mesure donnée. [une frontière mesure est associée à une seule application logicielle]*

Cardinalités: *min 1; max 1;*

NB: Tous les autres liens entre concepts sont décrits en annexe C de ce document.

La section suivante est consacrée à un exemple d'ontologie de tâches associée à une procédure de mesure.

3.3.2.2 ONTOLOGIE DE TÂCHES ASSOCIÉE À LA PROCÉDURE DE MESURE DE MKII-FPA

Le manuel de mesure de MkII-FPA version 1.3.1 [UKSMA 00, ISO/IEC 02], ainsi que l'expérience personnelle de mesure à l'aide de MkII-FPA, ont été nécessaires pour produire cette ontologie. Nous avons procédé à une analyse détaillée de la procédure de mesure associée à MkII-FPA. Toutes les tâches jugées pertinentes et associées à la procédure de mesure ont été identifiées. Les liens entre les tâches ont été explicitement déterminés en vue de produire une hiérarchie de tâches. Finalement, les tâches identifiées et liens entre tâches ont été représentés dans le formalisme choisi (formalisme orienté-objet [diagramme de composants UML] [Booch et al. 99]). Le diagramme d'activités UML a été également utilisé pour la spécification des tâches. Pour la documentation des tâches et liens entre tâches, la méthodologie CommonKADS a été mise à contribution, avec l'approche de description des tâches et liens entre tâches qui y est proposée. L'aide d'experts MkII-FPA est toujours requise pour raffiner et consolider ce travail.

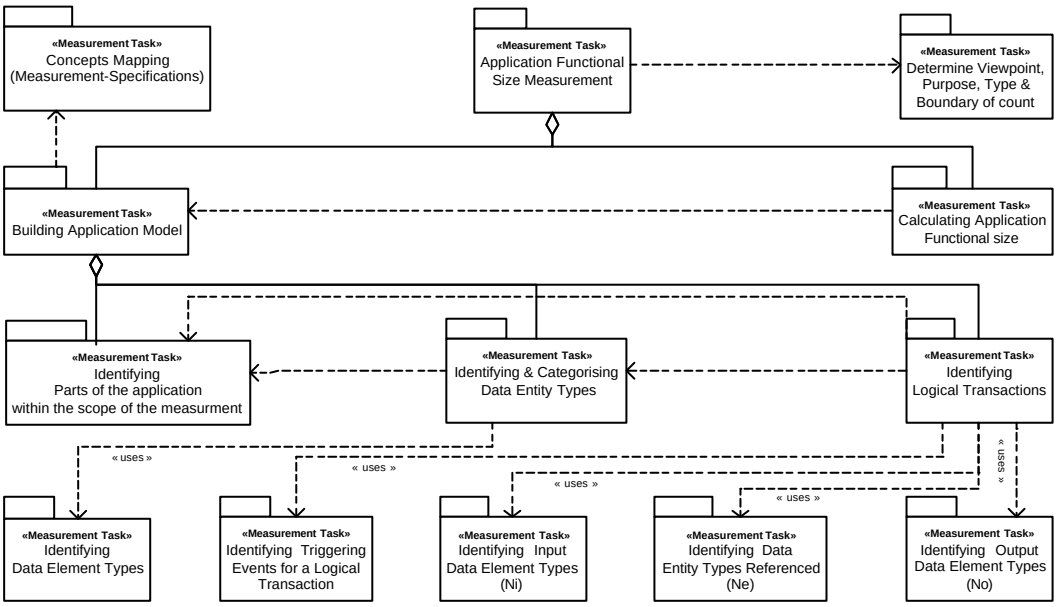


Figure 19 : Hiérarchie de tâches associée à la procédure de mesure de MkII-FPA (*Basée sur [Bevo et al 01]*)

Ci-dessous, nous présentons un exemple de description de tâches apparaissant dans la Figure 19.

J. Les tâches

J.1 Mesure de la taille fonctionnelle d'une application (Application Functional Size Measurement)

Définition :

Finalité : *Etant données un ensemble de spécifications pour une application logicielle, une perspective de mesure, un objectif de mesure, un type de mesure et une frontière, la tâche a pour finalité la production d'un modèle MkII-FPA pour l'application considérée, et la détermination de sa taille fonctionnelle en unités de taille fonctionnelle MkII-FPA. Cette tâche se décompose en deux (2) sous tâches complémentaires: « Construction Modèle Application » (« Building Application Model »), et « Calcul Taille Fonctionnelle Application » (« Calculating Application Functional size »).*

Entrées :*Ensemble_de_specifications_pour_une_application_logicielle, Perspective_de_la_mesure, Objectifs_de_la_mesure, Type_de_mesure, Frontière, Ensemble_de_correspondances_specifications_mesure*

Sorties :*Modele_MkIIFPA_pour_l_application_logicielle, Taille_fonctionnelle_de_l_application_logicielle [précision/certitude_globale_pour_la_mesure]*

Corps :

Type : *composition*

Sous-Tâches : *« Construction Modèle Application » (« Building Application Model »), «Calcul Taille Fonctionnelle Application» (« Calculating Application Functional size »)*

Exécution : *exécuter séquentiellement :*

- *Construction_Modele_Application (+Ensemble_de_specifications_pour_une_application_logicielle, +Perspective_de_la_mesure, +Objectifs_de_la_mesure, +Type_de_mesure, +Frontière, +Ensemble_de_correspondances_specifications_mesure, -Modele_MkIIFPA_pour_l_application_logicielle),*
- *Calcul_Taille_Fonctionnelle_Application (+Modele_MkIIFPA_pour_l_application_logicielle, -Taille_fonctionnelle_de_l_application_logicielle, -précision/certitude_globale_pour_la_mesure)*

J.2 Mapping de concepts (Concepts Mapping)

Définition :

Finalité : *Etablir des correspondances entre concepts de mesure et concepts d'un langage de spécification (par exemple UML).*

Entrées : *Concepts MkII-FPA, Concepts d'un langage de spécifications*

Sorties : *Ensemble_de_correspondances_specifications_mesure*

Corps :

Type : *simple*

Sous-Tâches : *;*

Exécution : *exécuter:*

- *Etablir_Correspondances (*
  +Concepts_MkII-FPA [ ],
  +Concepts_Langage_Spécifications[ ],
  -Ensemble_de_correspondances_specifications_mesure[ ])

NB: Toutes les autres tâches sont décrites en annexe C de ce document.

3.3.3 FORMALISATION ONTOLOGIQUE DE LA PROCÉDURE DE MESURE ASSOCIÉE À LA MÉTHODE DE MESURE FPA

3.3.3.1 ONTOLOGIE DE DOMAINE ASSOCIÉE À LA PROCÉDURE DE MESURE DE FPA

Le manuel de mesure de FPA v4.1 [IFPUG 01, ISO/IEC 03b], ainsi que notre expérience personnelle de mesure à l'aide de FPA, ont été nécessaires pour produire cette ontologie. Nous avons procédé à une analyse détaillée de la procédure de mesure associée à FPA. Tous les concepts jugés pertinents et associés à la procédure de mesure ont été identifiés et leurs définitions respectives extraites du manuel de mesure. Les liens entre les concepts ont été explicitement déterminés. Finalement, les concepts identifiés et liens entre concepts ont été représentés dans le formalisme choisi (formalisme orienté-objet [diagramme de classes UML [Booch et al. 99]). Pour la documentation des concepts et liens entre concepts, la méthodologie CommonKADS a été mise à contribution, avec l'approche de description des concepts et liens entre concepts qui y est proposée. L'aide d'experts FPA est toujours requise pour raffiner et consolider ce travail.

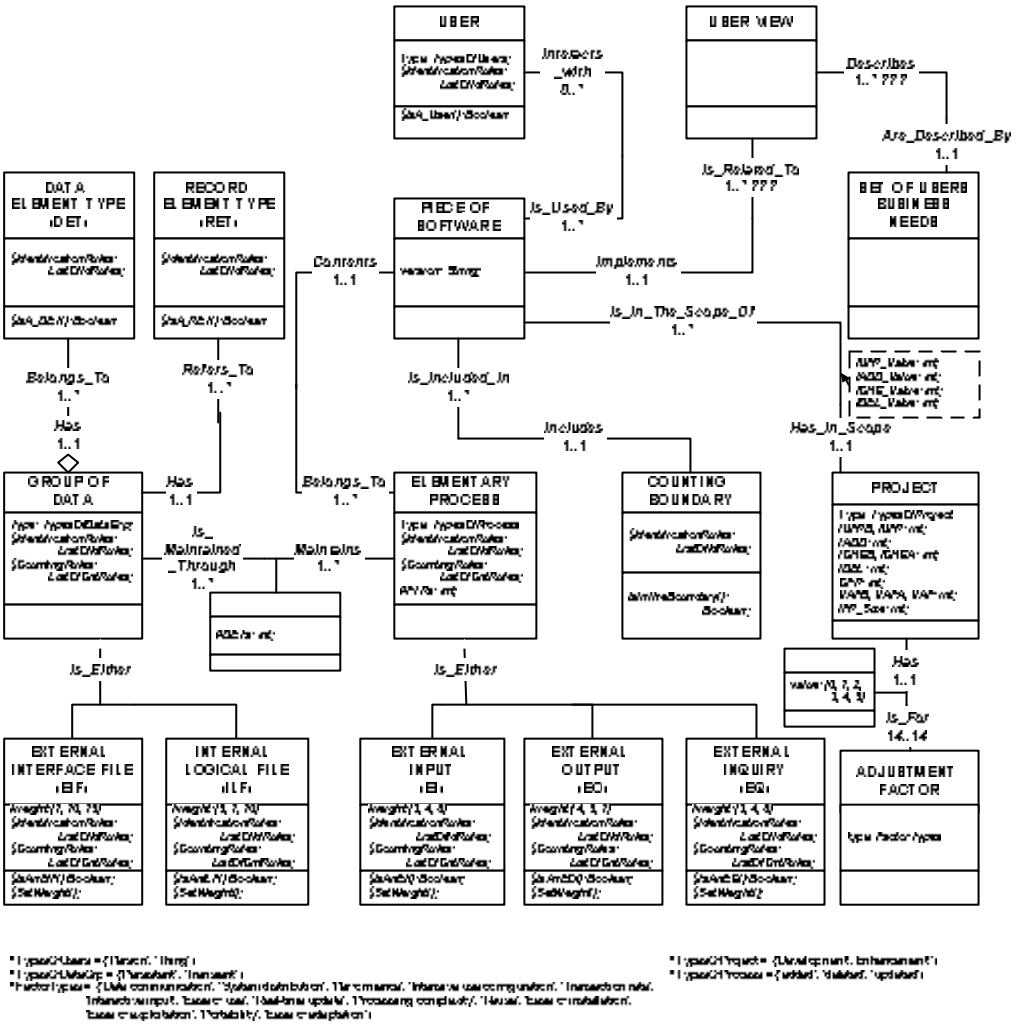


Figure 20 : Ontologie de domaine associée a la procédure de mesure de FPA
(*Adaptée de [Bevo et al 01]*)

Ci-dessous, nous présentons un exemple de description de concepts apparaissant dans la Figure 20.

- **Concepts**

NB: Toutes les définitions sont extraites du manuel de mesure de FPA v4.1 [IFPUG 01, ISO/IEC 03b].

J.3 Fichier Logique Interne (Internal Logical Files [ILF])

Définition:

*Un **Fichier Logique Interne** est un groupe de données ou d'information de contrôle logiquement associés, identifiables par les utilisateurs, et qui sont maintenues à l'intérieur de la frontière de l'application considérée. L'intention première d'un ILF est de contenir des données maintenues par le biais d'un ou plusieurs processus élémentaires de l'application considérée.*

Propriétés:

- *Name: {String};*
- *Description: {String};*
- *Weight: {int};*
- *IdentificationRules: {IdRule}*
- *MeasurementRules: {CntRule}*

Concept: IdRule

Condition: {logical expression}

Action: {procedure}

Concept: CntRule

Condition: {logical expression}

Action: {procedure}

J.4 Fichier Interface Externe (External Interface File [EIF])

Définition:

*Un **Fichier Interface Externe** est un groupe de données ou d'information de contrôle logiquement associés, référencés par l'application considérée, identifiables par les utilisateurs, mais qui sont maintenues à l'intérieur de la frontière d'une application autre que celle considérée. L'intention première d'un EIF est de contenir des données référencés par le biais d'un ou plusieurs processus élémentaires à l'intérieur de la frontière l'application considérée. Ainsi, un EIF pour l'application considérée est comptée comme un ILF pour une autre application*

Propriétés:

- *Name: {String};*
- *Description: {String};*
- *Weight: {int};*

- *IdentificationRules: {IdRule}*
- *MeasurementRules: {CntRule}*

Concept: IdRule

Condition: {logical expression}

Action: {procedure}

Concept: CntRule

Condition: {logical expression}

Action: {procedure}

J.5 Frontière de comptage (Measurement Boundary)

Définition:

*Une **frontière de comptage** fait référence à une interface conceptuelle entre une application et son environnement utilisateur extérieur. La frontière d'une application définit ce qui est extérieur à l'application. Elle agit comme une membrane à travers laquelle passent les données envoyées ou reçues par l'application considérée. Elle englobe les données logiques maintenues par l'application considérée, permet l'identification des données logiques référencées et non maintenues par l'application considérée. Elle dépend de la vue externe utilisateur de l'application (user's external business view of the application).*

Propriétés:

- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*

***Concept:** IdRule*

Condition: {expression logique}

Action: {procédure}

NB: Tous les autres concepts sont décrits en annexe D de ce document.

Un exemple de description de liens entre concepts apparaissant dans la Figure 20, est proposé dans la section suivante.

K. Liens entre concepts

K.1 Morceau de Logiciel – Processus Élémentaire (Piece_Of_Software - Elementary_Process)

Description:

Les processus élémentaires identifiés dans un morceau de logiciel.

Arguments:

Morceau de Logiciel (Piece_Of_Software)

Rôles: *Le nombre morceaux de logiciels dans lesquels l'on retrouve un processus élémentaire donné [un processus élémentaire donné se retrouve dans une et une seule version d'un morceau de logiciel]*

Cardinalités: *min 1; max 1;*

Processus Élémentaire (Elementary_Process)

Rôles: *Le nombre de processus élémentaire identifiés dans un morceau de logiciel donné [Un morceau de logiciel peut contenir plusieurs processus élémentaires]*

Cardinalités: *min 1; max n;*

K.2 Morceau de logiciel – Utilisateur (Piece Of Software - User)

Description:

Les utilisateurs pour un morceau de logiciel.

Arguments:

Morceau de logiciel (Piece Of Software)

Rôles: *Le nombre de morceaux de logiciel utilisés par un utilisateur donné [Un utilisateur pourrait l'être pour plusieurs morceaux de logiciel]*

Cardinalités: *min 1; max n;*

Utilisateur (User)

Rôles: *Le nombre d'utilisateurs associés à un morceau de logiciel donné [Un morceau de logiciel pourrait avoir plusieurs utilisateurs]*

Cardinalités: *min 0; max n;*

K.3 Morceau de logiciel – Frontière de comptage (Piece of Software – Measurement Boundary)

Description:

Frontière de comptage pour un morceau de logiciel.

Arguments:

Frontière de comptage (Measurement Boundary)

***Rôles**: Le nombre frontières de comptage pour un morceau de logiciel donné [un morceau de logiciel donné est associé une et une seule frontière de comptage]*

***Cardinalités:** min 1; max 1;*

Morceau de logiciel (Piece of Software)

***Rôles:** Le nombre de morceaux de logiciel associés à une frontière comptage donnée. [une frontière de comptage pourrait englober plusieurs morceaux de logiciels]*

***Cardinalités:** min 1; max n;*

NB: Tous les autres liens entre concepts sont décrits en annexe D de ce document.

La section suivante est consacrée à un exemple d'ontologie de tâches associée à une procédure de mesure.

3.3.3.2 ONTOLOGIE DE TÂCHES ASSOCIÉE À LA PROCÉDURE DE MESURE DE FPA

Le manuel de mesure de FPA v4.1 [IFPUG 01, ISO/IEC 03b], ainsi que notre expérience personnelle de mesure à l'aide de FPA, ont été nécessaires pour produire cette ontologie. Nous avons procédé à une analyse détaillée de la procédure de mesure associée à FPA. Toutes les tâches jugées pertinentes et associées à la procédure de mesure ont été identifiées. Les liens entre les tâches ont été explicitement déterminés en vue de produire une hiérarchie de tâches. Finalement, les tâches identifiées et liens entre tâches ont été représentés dans le formalisme choisi (formalisme orienté-objet [diagramme de composants UML] [Booch et al. 99]). Le diagramme d'activités UML a été également utilisé pour la spécification des tâches. Pour la documentation des tâches et liens entre tâches, la méthodologie CommonKADS a été mise à contribution, avec l'approche de description des tâches et liens entre tâches qui y est proposée. L'aide d'experts FPA est toujours requise pour raffiner et consolider ce travail.

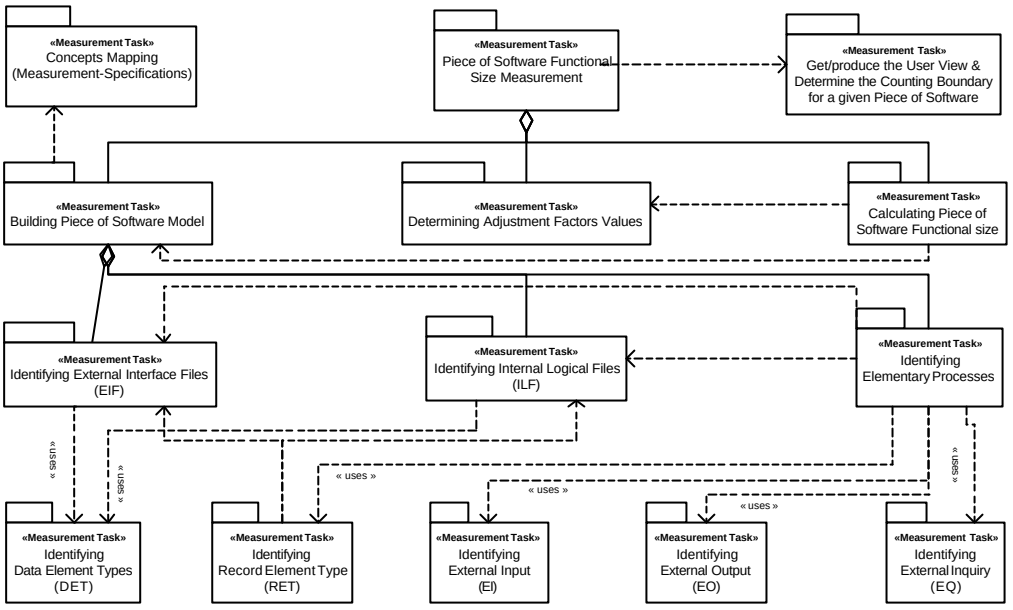


Figure 21 : Hiérarchie de tâches associée à la procédure de mesure de FPA (*Basée sur [Bevo et al 01]*)

Ci-dessous, nous présentons un exemple de description de tâches apparaissant dans la Figure 21.

L. Les tâches

L.1 Mesure de la taille fonctionnelle d'un morceau de logiciel (Piece of software Functional Size Measurement)

Définition :

Finalité : *Etant données une vue utilisateur associée à un morceau de logiciel (description formelle des fonctions d'affaire utilisateur dans un langage accessible aux utilisateurs) et une frontière de comptage, la tâche a pour finalité la production d'un modèle FPA pour le morceau de logiciel, la détermination des valeurs des facteurs d'ajustement associées à la mesure et le calcul de la taille fonctionnelle en unités de taille fonctionnelle FPA. Cette tâche se décompose en trois (3) sous tâches complémentaires: « Construction Modèle Morceau Logiciel » (« Building Piece of Software Model »), « Détermination Valeurs Facteurs Ajustement » (Determining*

Adjustment Factors Values) et « Calcul Taille Fonctionnelle Morceau Logiciel » (« Calculating Piece of Software Functional size »).

Entrées : *Vue_Utilisateur,*
Frontiere_Comptage,
Caracteristiques_Projet,
Valeurs_Facteurs_Ajustement
Ensemble_de_correspondances_specifications_mesure

Sorties : *Modele_FPA_pour_le_morceau_de_logiciel,*
Taille_fonctionnelle_du_morceau_de_logiciel,
[précision/certitude_globale_pour_la_mesure]

Corps :

Type : *composition*

Sous-Tâches : *« Construction Modèle Morceau Logiciel » (« Building Piece of Software Model »), « Détermination Valeurs Facteurs Ajustement » (Determining Adjustment Factors Values) et « Calcul Taille Fonctionnelle Morceau Logiciel » (« Calculating Piece of Software Functional size »)*

Exécution : *exécuter séquentiellement :*

- *Construction_Modele_Morceau_Logiciel*
  (+Vue_Utilisateur,
  +Frontiere_Comptage,
  +Ensemble_de_correspondances_specifications_mesure,
  -Modele_FPA_pour_le_morceau_de_logiciel),
- *Détermination_Valeurs_Facteurs_Ajustement (*
  + Vue_Utilisateur,
  +Caracteristiques_Projet,
  -Valeurs_Facteurs_Ajustement[]),
- *Calcul Taille Fonctionnelle Morceau Logiciel(*
  +Modele_FPA_pour_le_morceau_de_logiciel,
  +Valeurs_Facteurs_Ajustement[]
  -Taille_fonctionnelle_du_morceau_de_logiciel,
  -précision/certitude_globale_pour_la_mesure)

L.2 Construction du modèle FPA d'un morceau de logiciel (Building a FPA Piece of software model)

Définition:

Finalité : *Etant données une vue utilisateur associée à un morceau de logiciel (description formelle des fonctions d'affaire utilisateur dans un langage accessible aux utilisateurs), une frontière de comptage et un ensemble de correspondances entre concepts de mesure et concepts d'un langage de spécification, la tâche a pour finalité la production d'un modèle FPA pour le morceau de logiciel. Cette tâche se décompose en quatre (4) sous tâches complémentaires: « Identification_Fichiers_Interface_Externes» (« Identifying_External_Interface_Files »),*
« Identification_Fichiers_Logiques_Internes»
(« Identifying_Internal_Logical_Files »),
« Identification_Processus_Elementaires»
(« Identifying_Elementary_Processes »),
« Production_Modele_FPA_pour_le_morceau_de_logiciel»
(« Produce_The_FPA_Model_Of_The_Piece_Of_Software»)

Entrées : *Vue_Utilisateur,*
Frontiere_Comptage,
Ensemble_de_correspondances_specifications_mesure

Sorties : *Modele_FPA_pour_le_morceau_de_logiciel*

Corps :

Type : *composition*

Sous-Tâches : *« Identification_Fichiers_Interface_Externes»*
(« Identifying_External_Interface_Files »),
« Identification_Fichiers_Logiques_Internes»
(« Identifying_Internal_Logical_Files »),
« Identification_Processus_Elementaires»
(« Identifying_Elementary_Processes »),
« Production_Modele_FPA_pour_le_morceau_de_logiciel»
(« Produce_The_FPA_Model_Of_The_Piece_Of_Software»)

Exécution : *exécuter séquentiellement :*

- *Identification_Fichiers_Interface_Externes (*
  +Vue_Utilisateur,
  +Ensemble_de_correspondances_specifications_mesure[],

+Frontiere_Comptage
-Fichiers_Interface_Externes[ ])

- *Identification_Fichiers_Logiques_Internes (*
  +Vue_Utilisateur,
  +Ensemble_de_correspondances_specifications_mesure[],
  +Frontiere_Comptage
  -Fichiers_Logiques_Internes[ ])
- *Identification_Processus_Elementaires (*
  +Vue_Utilisateur,
  +Ensemble_de_correspondances_specifications_mesure[],
  +Frontiere_Comptage
  -Processus_Elementaires[ ])
- *Production_Modele_FPA_pour_le_morceau_de_logiciel (*
  +Vue_Utilisateur,
  +Fichiers_Interface_Externes[ ]
  +Fichiers_Logiques_Internes[ ],
  +Processus_Elementaires[ ]
  - Modele_FPA_pour_le_morceau_de_logiciel)

L.3 Calcul de la taille fonctionnelle d'un morceau de logiciel (Calculating Piece of Software Functional Size)

Définition :

Finalité : *Etant donné un modèle FPA d'un morceau de logiciel et les valeurs des facteurs d'ajustement pour ce morceau de logiciel, la tâche a pour finalité la détermination de la taille fonctionnelle du morceau de logiciel (taille non ajustée puis taille ajustée) détermination de la taille fonctionnelle doit tenir compte du paramètre d'incertitude associé à certains éléments du modèle (ILFs, EIFs, groupes de données, etc.), en vue de déterminer la précision associée au résultat de mesure.*

Entrées : *Modele_FPA_pour_le_morceau_de_logiciel, Valeurs_Facteurs_Ajustement[]*

Sorties : *Taille_fonctionnelle_du_morceau_de_logiciel, [précision/certitude_globale_pour_la_mesure]*

Corps :

Type : *simple*

Sous-Tâches : ;

Exécution : *exécuter:*

- *Calcul Taille Fonctionnelle Morceau Logiciel(*
  +Modele_FPA_pour_le_morceau_de_logiciel,
  +Valeurs_Facteurs_Ajustement[]
  -Taille_fonctionnelle_du_morceau_de_logiciel,
  -précision/certitude_globale_pour_la_mesure)

L.4 Mapping de concepts (Concepts Mapping)

Définition :

Finalité : *Etablir des correspondances entre concepts de mesure et concepts d'un langage de spécification (par exemple UML).*

Entrées : *Concepts FPA, Concepts d'un langage de spécifications*

Sorties : *Ensemble_de_correspondances_specifications_mesure*

Corps :

Type : *simple*

Sous-Tâches : *;*

Exécution : *exécuter:*

- *Etablir_Correspondances (*
  +Concepts_FPA[ ],
  +Concepts_Langage_Spécifications[ ],
  - Ensemble_de_correspondances_specifications_mesure[ ])

NB: Toutes les autres tâches sont décrites en annexe D de ce document.

3.4 EVALUATION DES ONTOLOGIES DEVELOPPEES

Pour l'évaluation des ontologies proposées dans le cadre de la formalisation ontologique des procédures de mesure, nous nous inspirons travail de [Madhavan et al. 02] qui proposent un cadre d'évaluation consistant en deux (2) choses : (i) l'évaluation des propriétés des ontologies produites, (ii) l'évaluation des propriétés de la technologie utilisée pour produire les ontologies.

L'évaluation des propriétés des ontologies produites couvre aussi bien les aspects syntaxiques que sémantiques desdites ontologies. Sur le plan syntaxique, il s'agit de vérifier la conformité de l'ontologie relativement au langage de représentation choisi (l'on pourrait alors faire appel au manuel de référence du langage). Sur le plan sémantique, il s'agit de

vérifier la cohérence de l'ontologie (pas de spécifications contradictoires) et la fidélité de l'ontologie par rapport à ce qu'elle est censée spécifier. Pour ce dernier point, la tâche incombe principalement aux experts des domaines/processus pour lesquels les ontologies ont été développées (dans notre cas, il s'agit des experts des différentes méthodes de mesure pour lesquelles les procédures de mesure sont formalisées). D'où la nécessite de choisir un langage de spécification simple, expressif, accessible idéalement à tous les intervenants (ou à une bonne partie).

L'évaluation des propriétés de la technologie utilisée pour produire les ontologies se focalise sur les outils de développement des ontologies utilisées. Elle couvre les aspects suivants: interopérabilité (facilité d'échange des ontologies produites avec d'autres outils existants), stabilité des représentations (elles ne varient en fonction du temps), performance, allocation mémoire, extensibilité, connectivite à d'autres outils. Dans notre cas, cette évaluation est préalable au développement des ontologies et oriente le choix d'un outil de développement d'ontologies.

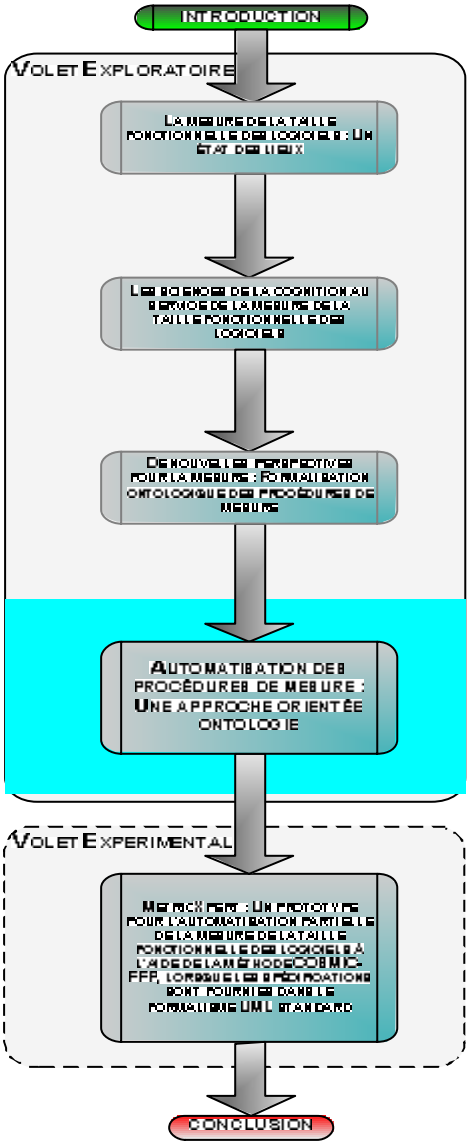
4 SYNTHÈSE

Au total, la formalisation ontologique des procédures de mesure que nous préconisons pour chaque méthode de mesure constitue à notre avis un outil efficace pour unifier l'interprétation des concepts et règles associées aux procédures de mesure, éviter les ambiguïtés terminologiques, proposer une représentation explicite partagée des connaissances relatives aux procédures de mesure, améliorer la communication entre experts et/ou novices, donc permettre une meilleure réutilisation et interopérabilité.

Les ontologies de domaine produites constituent l'un des maillons essentiels dans la mise en œuvre du modèle cognitif décrivant une procédure de mesure que nous présentons plus bas. *Quant aux ontologies de tâches, elles constituent des spécialisations (au sens orienté-objet) du modèle cognitif décrivant une procédure de mesure pour les méthodes de mesure considérées*. Ce modèle est à la base de l'approche orientée ontologie pour le développement d'outils d'automatisation du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir de spécifications présentées dans un formalisme bien défini. L'approche est au cœur de la section qui suit.

CHAPITRE 4.

DÉVELOPPEMENT D'OUTILS D'AUTOMATISATION DES PROCÉDURES DE MESURE : UNE APPROCHE ORIENTÉE ONTOLOGIE



FEUILLE DE ROUTE

1. *Introduction*
2. *Modèle cognitif décrivant une procédure de mesure*
3. *L'approche orientée ontologie pour le développement d'outils d'automatisation des procédure de mesure*
4. *Architecture générale d'un système automatisant une bonne partie des procédures de mesure*
5. *Synthèse*

EN BREF …

Ce chapitre introduit une nouvelle approche que nous préconisons pour le développement d'outils d'automatisation des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels. Il s'agit d'une approche orientée ontologie. Elle s'appuie sur le travail de formalisation ontologique des procédures de mesure introduit plus haut et découle d'un modèle cognitif décrivant une procédure de mesure (dont les détails du modèle sont fournis en section 2 de ce chapitre). Les principaux avantages d'une telle approche sont précisés. Finalement, afin de soutenir l'approche, une architecture générale pour un système automatisant une bonne partie des procédures est présentée. Cette architecture s'arrime bien avec l'approche. Proposée.

1 INTRODUCTION

Tel que souligné plus haut, l'automatisation complète ou partielle des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels, constitue à notre avis une piste prometteuse pour s'attaquer à bon nombre de problèmes auxquels fait face l'application des méthodes de mesure aujourd'hui (assistance d'experts malheureusement pas toujours disponibles, subjectivité due à l'interprétation des règles de mesure, etc.). Un certain nombre de caractéristiques souhaitables pour les outils d'automatisation de la mesure, ont été identifiés par Keith Paton et Alain Abran [Abran et al. 97] :

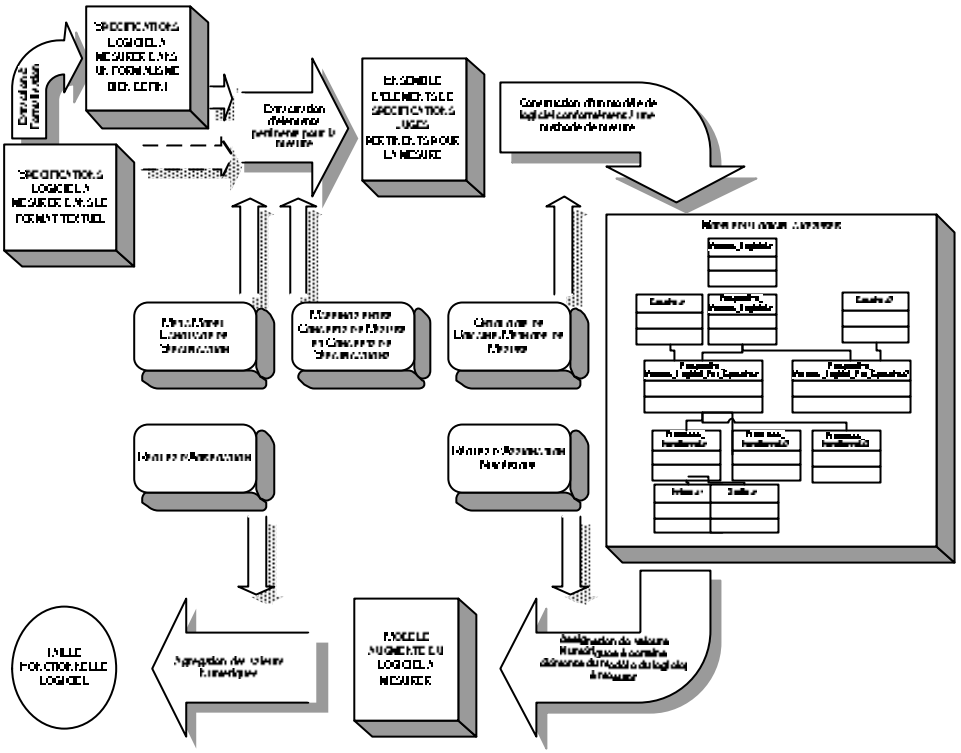
- les outils doivent être associés à des **normes** ou des **standards** reconnus. Aujourd'hui, il existe dans le domaine de la mesure de la taille fonctionnelle des logiciels, un certain nombre de normes [ISO 93, ISO/IEC 97a, ISO/IEC 97b, Abran et al. 99] et de standards reconnus (MkII FPA, FPA, COSMIC-FFP);
- les outils doivent offrir un bon degré de **flexibilité**, notamment :
    - les règles de mesure implantées doivent être modifiables par programme et stockées à l'extérieur de l'outil de mesure. Ceci faciliterait l'adaptation des outils à l'évolution des méthodes de mesure (A titre d'exemple, COSMIC-FFP est passée de la version 1.0 à la version 2.1 en 6 ans, FPA est passée de la version 1.0 à la version 4.1 en 13 ans, MkII FPA en est à la version 1.3.1 aujourd'hui),
    - la modularité doit être de mise dans la conception et la construction des outils, ce qui garantira une meilleure modifiabilité;
- l'**architecture** des outils doit être **ouverte**. Cela se traduit par :
    - une possibilité d'interaction avec les outils CASE d'analyse et de conception de logiciels. Il serait ainsi possible d'importer les spécifications produites à l'aide de tels outils (par exemple, importation des diagrammes conceptuels [diagrammes UML, DFD,…]),
    - une facilité d'exportation des données produites par les outils (résultats de mesure) vers des SGBD externes, ce qui garantit la persistance desdites données,

  - une facilité d'interaction avec des outils statistiques pour l'analyse des résultats de mesure par exemple,
  - une facilité d'interaction avec d'autres outils de mesure;
- les outils doivent produire une représentation explicite de l'application cible de la mesure (**richesse sémantique du modèle**);
- les outils doivent offrir une présentation des résultats de mesure facilitant les analyses (**documentation des mesures**);
- les outils doivent être **performants** en traitements et en stockage;
- Les outils doivent être **multi plates-formes** (Windows, Unix, Apple Machintosh II/A/UX, …);
- les outils doivent pouvoir gérer certaines **connaissances incertaines** (incertitudes liées à l'identification des concepts et dues parfois à la mauvaise qualité des documents de spécification).

L'approche que nous préconisons dans le cadre du développement d'outils d'automatisation des procédures de mesure, permet de relever bon nombre de ces défis. Elle s'appuie sur un modèle cognitif décrivant une procédure de mesure, qui est présenté ci-dessous.

2 MODÈLE COGNITIF DÉCRIVANT UNE PROCÉDURE DE MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS À PARTIR DES SPÉCIFICATIONS

Sur le plan cognitif, le processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir de spécifications s'apparente à un processus cognitif d'interprétation comme l'indique la Figure 22.



présentées dans le format textuel ou alors un formalisme « bien défini » (*par exemple UML standard : Lorsque les spécifications sont disponibles dans le formalisme UML, un certain nombre de diagrammes sont jugés pertinents pour la mesure. Ce sont les diagrammes de cas d'utilisation, de séquences, de classes, etc. Dans le cadre de la mesure, l'on s'intéressera donc dans les spécifications à ces diagrammes*). Lorsque l'on a affaire au format textuel, l'extraction est plus complexe et à notre avis **difficilement (voire pas du tout) automatisable** (dans le contexte actuel, les systèmes de traitements du langage naturel). L'étape d'extraction est suivie d'une **étape de classification** des éléments de spécification extraits : Pour chaque élément de spécification identifié il s'agit de déterminer à quelle catégorie il correspond relativement à la méthode de mesure considérée (les catégories dont il est question ici sont celles décrites dans l'ontologie de domaine associée à la méthode de mesure considérée). *Pour la méthode COSMIC-FFP par exemple, ce sont les couches, les processus fonctionnels, les groupes de données, etc. Pour la méthode MkII-FPA, ce sont les « Logical Transactions », les « Input Data Element Types (Ni) », les « Data Entity Types Referenced (Ne) », etc.*). Ce travail de classification sera d'autant plus aisé s'il existe une correspondance (un « mapping ») entre les concepts de mesure (qui représentent les catégories mentionnées plus haut) et les concepts de spécification, *auquel cas ce travail pourrait même être automatisé* (les concepts de spécification sont ceux que l'on retrouve dans le langage de spécification utilisé dans la documentation du logiciel, et qui sont jugés pertinents relativement à la mesure). *Pour le langage UML standard par exemple, ce sont les concepts d'acteur, de classe persistante, de message, etc.*). Malheureusement, la correspondance entre ces concepts est rarement disponible, ce qui n'est pas pour faciliter le travail du « mesureur ». Nous avons réalisé une telle correspondance pour la méthode COSMIC-FFP et le formalisme UML standard [Bevo et al., 99]. L'étape de classification est suivie d'une **étape de modélisation** au cours de laquelle sera produite un modèle du logiciel à mesurer à partir des instances de catégories identifiées précédemment. Pour la construction du modèle, l'on a besoin d'un moule fourni par l'ontologie de domaine associée à la méthode de mesure considérée.

La construction du modèle est suivie de la phase de « mesure » du processus de mesure. Cette ultime phase du processus compte deux (2) étapes élémentaires. La première étape consiste en l'**application d'un ensemble de règles d'assignation numérique** au modèle construit. Ces règles définissent les conditions d'assignation de valeurs numériques à

certains éléments du modèle construit, en vue de déterminer la taille fonctionnelle du logiciel à mesurer (*Par exemple pour la méthode COSMIC-FFP on a la règle suivante : « A unit of measurement, 1 Cfsu, shall be assigned to each data movement identified, that is to each instance of a data movement (Entry, Exit, Read or Write) identified. » [Abran et al., 03]*). Après l'étape d'assignation numérique suit une **étape d'agrégation** au cours de laquelle les valeurs numériques assignées sont agrégées suivant un ensemble de règles d'agrégation définies par la méthode de mesure considérée (*Par exemple pour la méthode COSMIC-FFP on a les règles suivantes :*

« SizeCfsu (Functional Process$_i$) = S *size(Entries$_i$) +* S *size(Exits$_i$) +* S *size(Reads$_i$) +* S *size(Writes$_i$) », « SizeCfsu (FUR$_i$) =* S *size(Functional Process$_i$) » [Abran et al., 03]. Pour la méthode MKII-FPA : « The Functional Size (Function Point Index) is the weighted sum over all Logical Transactions, of the Input Data Element Types (N_i), the Data Entity Types Referenced (N_e), and the Output Data Element Types (N_o) »*).

Nous présentons dans la section qui suit l'approche que nous préconisons dans le cadre du développement d'outils d'automatisation du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir de spécifications présentées dans un formalisme bien défini.

3 UNE APPROCHE ORIENTEE ONTOLOGIE POUR LE DEVELOPPEMENT D'OUTILS D'AUTOMATISATION DES PROCEDURES DE MESURE

3.1 LES GRANDES LIGNES DE L'APPROCHE

L'approche que nous proposons dans le cadre du développement d'outils d'automatisation du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir de spécifications présentées dans un formalisme bien défini, prend appui sur les ontologies associées au processus (Figure 23).

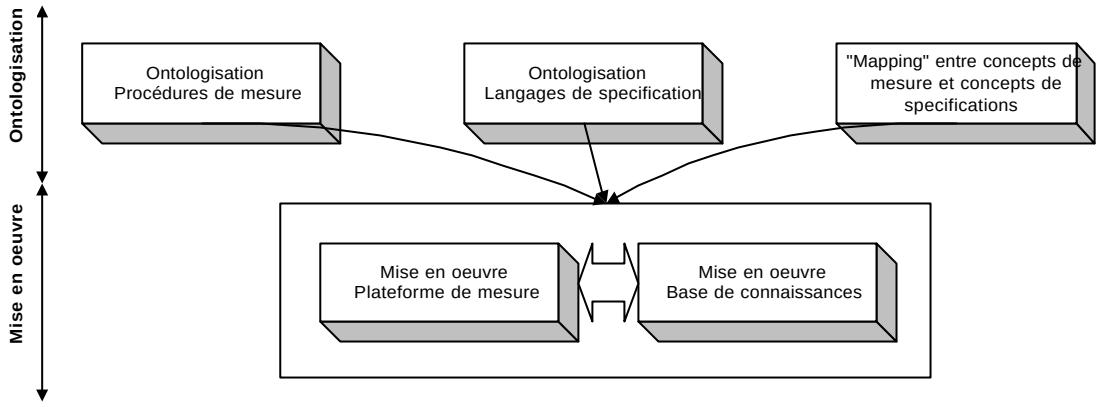


Figure 23 : Approche orientée ontologie pour le développement d'outils d'automatisation du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir de spécifications présentées dans un formalisme bien défini

Elle consiste en deux étapes complémentaires : Une étape d'« ontologisation » et une étape de mise en œuvre (Figure 23). À travers les ontologies bien spécifiques associées au processus, cette approche met l'emphase sur la représentation et la manipulation des connaissances relatives au processus.

3.1.1 L'ÉTAPE D'« ONTOLOGISATION »

Il s'agit ici de rassembler et analyser les ontologies associées aux procédures de mesure. Lorsqu'elles n'existent pas, les ontologies doivent être développées. Les ontologies à développer ou à utiliser (lorsqu'elles existent) sont relatives aux tâches de mesure, aux concepts de mesure et aux concepts de spécification. L'une des tâches clefs de cette étape d'« ontologisation » est le « mapping » (la mise en correspondance) entre les concepts de mesure et les concepts de spécification jugés pertinents pour la mesure. Les ontologies sont, avec les correspondances (« mappings ») entre concepts de mesure et concepts de spécification, les éléments essentiels du processus d'application d'une méthode de mesure. L'idéal serait d'aboutir à un système qui prendrait ces éléments en paramètre. Nous estimons que ce sont des paramètres nécessaires et suffisants pour le processus. Le principal défi dans l'étape suivante de notre approche est de trouver une façon de les **implanter** et les **exploiter** dans un système informatique pour la mesure. Nous levons un pan de voile sur une façon de relever ce défi dans la section suivante.

3.1.2 L'ÉTAPE DE MISE EN ŒUVRE

Il s'agit d'une étape presque essentiellement technique. Il faut exploiter les éléments issus de l'étape d'« ontologisation » dans la construction de l'outil de mesure. Nous optons pour une approche outil à base de connaissances. Par conséquent il faudrait construire la base de connaissances de l'outil et implanter les services à offrir aux utilisateurs.

3.1.2.1 MISE EN OEUVRE DE LA BASE DE CONNAISSANCES

La base de connaissances rassemble les connaissances relatives aux méthodes de mesure, les connaissances relatives aux langages de spécification et les correspondances entre concepts de mesure et/ou concepts de spécification. Les connaissances relatives aux méthodes de mesure sont essentiellement les concepts de mesure et les liens entre concepts de mesure pour les méthodes de mesure considérées (ontologies de domaine associées aux méthodes de mesure). Les connaissances relatives aux langages de spécification sont essentiellement les concepts de spécification et les liens entre concepts de spécification pour les langages de spécification considérés (ontologies de domaine associées aux langages de spécification).

Dans le cas par exemple où il est fait appel à un SGBD pour la mise en œuvre de la base de connaissances, alors le schéma relationnel associé au diagramme de classes de la Figure 12, pourrait servir de méta-modèle pour la base de connaissances. De plus, pour chaque concept inséré dans la base de connaissances (concept de mesure ou de spécification), il sera créé une table relationnelle associée au concept; cette table contiendra les instances du concept (faits associés à la connaissance) générés pendant les séances de mesure. Par exemple, pour le concept de *processus fonctionnel*, il sera créé la table *COSMIC_FUNCTIONAL_PROCESS_T*:

```
CREATE TABLE COSMIC_FUNCTIONAL_PROCESS_T (
id                          VARCHAR(50),
name                        VARCHAR(300) NOT NULL,
typeOfFunctionalProcess     VARCHAR(30) DEFAULT 'ADDED',
description                 VARCHAR(300),
version                     VARCHAR(40) DEFAULT 'Version 1',
dateInsert                  DATE DEFAULT SYSDATE NOT NULL ,
COSMIC_FFP_Size             NUMBER(9)  DEFAULT 0,
```

```
addedSize                NUMBER(9)  DEFAULT 0,
changedSize              NUMBER(9)  DEFAULT 0,
deletedSize              NUMBER(9)  DEFAULT 0,
belief                   NUMBER(4)   DEFAULT 100
                         CHECK (belief>=0 and belief<=100),
CONSTRAINT COSMIC_chk_typeOfFctlProc
                         CHECK(typeOfFunctionalProcess IN
                         ('ADDED', 'DELETED', 'UPDATED')),
CONSTRAINT COSMIC_pk_Fctl_Proc PRIMARY KEY(id));
```

Pour chaque correspondance entre un concept de mesure et un concept de spécification il sera créé une table relationnelle associée à la correspondance ; cette table contiendra les instances de liens entre instances de concepts de mesure et instances de concepts de spécification, générés pendant les séances de mesure. De plus, il sera créé un ou plusieurs déclencheurs (« triggers ») chargés de la maintenance **automatique** du contenu de la table associée à la correspondance et de la table associée au concept de mesure impliqué dans la correspondance, suivant les événements survenant sur la table associée au concept de spécification impliqué dans la correspondance. ***En fait, ces déclencheurs s'occupent de la construction et la maintenance automatique des instances de modèles de logiciels relatifs aux méthodes de mesure*** *(contenus dans les tables associées aux concepts de mesure)****, sur la base des spécifications desdits logiciels*** *(contenues dans les tables associées aux concepts de spécification)****.*** À titre d'exemple, pour la correspondance *scénario – processus fonctionnel,* seront créés la table *MAPPING_SCENARIO_T* et les déclencheurs *MAPPING_trigSCENARIO_INSERT, MAPPING_trigSCENARIO_UPDATE* et *MAPPING_trigSCENARIO_DELETE* :

```
CREATE TABLE MAPPING_SCENARIO_T (
dateInsert          DATE DEFAULT SYSDATE NOT NULL ,
scenarioID          VARCHAR2(50),
functionalProcessID VARCHAR(50),
CONSTRAINT MAPPING_fk_MapScen_scenario FOREIGN KEY(scenarioID)
                    REFERENCES UML_SCENARIO_T (id),
CONSTRAINT MAPPING_fk_MapScen_fctlProc
```

```
                    FOREIGN KEY(functionalProcessID)
                    REFERENCES  COSMIC_FUNCTIONAL_PROCESS_T  (id)
                        ON DELETE CASCADE,
 CONSTRAINT MAPPING_pk_MapScen PRIMARY KEY(scenarioID,
                                    functionalProcessID));

CREATE OR REPLACE TRIGGER MAPPING_trigSCENARIO_INSERT
AFTER INSERT ON UML_SCENARIO_T
FOR EACH ROW
DECLARE
   newID INTEGER;
   pieceOfSoftID VARCHAR2(50);
   fctlProcID VARCHAR2(50);
   layrID VARCHAR2(50);
   projID VARCHAR2(50);
BEGIN
INSERT INTO COSMIC_FUNCTIONAL_PROCESS_T (id, name,  description, belief)
                        VALUES  (:new.id, :new.title, :new.description, 100);
SELECT COSMIC_seqFCTL_PROC.currVal INTO newID FROM DUAL;
SELECT id INTO fctlProcID FROM COSMIC_FUNCTIONAL_PROCESS_T
                    WHERE id LIKE '%\_'||newID ESCAPE '\';
SELECT id INTO layrID FROM COSMIC_LAYER_T
                    WHERE name LIKE 'Default Layer%';
SELECT cosmicPieceOfSoftwareID INTO pieceOfSoftID
 FROM MAPPING_UML_POF_SOFTWARE_T
 WHERE umlPieceOfSoftwareID IN (SELECT applicationID
                    FROM UML_USE_CASE_T
                    WHERE id = :new.useCaseID);
SELECT id INTO projID  FROM COSMIC_PROJECT_T
                    WHERE name LIKE 'Default Project%';
INSERT INTO MAPPING_SCENARIO_T (scenarioID, functionalProcessID)
                    VALUES  (:new.id, fctlProcID);
INSERT INTO COSMIC_PROCESS_LAYER_PIECE_T (functionalProcessID, layerID,
                                    pieceOfSoftwareID, projectID)
                    VALUES (fctlProcID, layrID, pieceOfSoftID,  projID);
```

```
END;
/

CREATE OR REPLACE TRIGGER MAPPING_trigSCENARIO_DELETE
BEFORE DELETE ON UML_SCENARIO_T
FOR EACH ROW
DECLARE
BEGIN
DELETE FROM COSMIC_FUNCTIONAL_PROCESS_T
 WHERE id IN (SELECT functionalProcessID
            FROM MAPPING_SCENARIO_T
            WHERE scenarioID = :old.id);
 DELETE FROM MAPPING_SCENARIO_T
 WHERE scenarioID = :old.id;
END;
/

CREATE OR REPLACE TRIGGER MAPPING_trigSCENARIO_UPDATE
AFTER UPDATE ON UML_SCENARIO_T
FOR EACH ROW
DECLARE
BEGIN
UPDATE COSMIC_FUNCTIONAL_PROCESS_T
SET name = :new.title,
Description = :new.description
 WHERE id IN (SELECT functionalProcessID
            FROM MAPPING_SCENARIO_T
            WHERE scenarioID = :old.id);
END;
/
```

Pour ce qui est des correspondances entre concepts de différentes méthodes de mesure ils seront exploités dans les prochaines itérations (post thèse) pour la convertibilité inter méthodes de mesure des résultats de mesure.

Au total, la première grande tâche dans cette deuxième étape de l'approche que nous proposons est consacrée à la mise en place de la base de connaissance de l'outil à développer en exploitant des éléments issus de l'étape d'« ontologisation ». Il s'agit maintenant d'exploiter à bon escient et de gérer la base de connaissances pour répondre aux besoins des utilisateurs d'outils de mesure. C'est l'objet de la seconde grande tâche dans cette deuxième étape de l'approche que nous proposons.

3.1.2.2 MISE EN ŒUVRE DE LA PLATEFORME DE MESURE

La plateforme de mesure implante tous les services à offrir aux utilisateurs de l'outil de mesure. Ces services sont relatifs à la gestion des connaissances de la base, l'extraction des éléments de spécification jugés pertinents pour la mesure, la gestion des modèles de logiciel produits, l'évaluation de l'intervalle de confiance associé à la mesure, la documentation des résultats de mesure ou encore l'analyse des résultats de mesure.

Pour la gestion des connaissances de la base, l'on pourrait mettre en œuvre trois (3) services bien spécifiques : Un service spécifique à la gestion des connaissances relatives aux méthodes de mesure, un service spécifique à la gestion des connaissances relatives aux langages de spécification et un service spécifique à la gestion des correspondances entre concepts de mesure et concepts des langages de spécification de logiciels. Les deux premiers services permettraient en fait d'importer (complètement ou partiellement) des ontologies produites à l'aide de tels outils, de les stocker dans la base de connaissances du système en vue de leur exploitation dans le cadre de la mesure.

Pour l'extraction des éléments de spécifications pertinents pour la mesure, l'on pourrait mettre en œuvre un ensemble de « parseurs » spécifiques aux outils CASE de spécification.

De plus amples détails sur tous ces services sont fournis dans la section 4 plus bas consacrée à l'architecture générale pour un système d'automatisation. Cette architecture s'adapte bien à l'approche que nous avons détaillée plus haut pour le développement d'outils d'automatisation. Dans la section qui suit, nous soulignons l'intérêt d'une telle approche.

3.2 L'INTÉRÊT DE L'APPROCHE

L'approche que nous préconisons favorise entre autres:

- *Un « design » plus précis des méthodes de mesure :* l'approche est à notre avis susceptible d'encourager un «design » plus précis des méthodes de mesure de la taille fonctionnelle des logiciels. En effet, la formalisation ontologique des procédures de mesure, qui constitue la base de l'approche, permet de rendre explicite les connaissances et les hypothèses relatives au processus d'application d'une méthode de mesure. L'idéal serait que cette formalisation soit réalisée par les concepteurs des méthodes de mesure. Mais dans tous les cas, grâce à l'approche, autant les concepteurs que les utilisateurs des méthodes de mesure pourraient tirer profit du travail d'automatisation, même s'il n'aboutit pas à un outil logiciel. Les concepteurs pourraient avoir du feedback pour la maintenance des procédures de mesure et donc des méthodes de mesure, tandis que les utilisateurs pourraient bénéficier d'une meilleure compréhension des méthodes de mesure et de leurs processus d'application. De plus, il est désormais possible de déterminer formellement, à l'avance (avant de se lancer dans toute construction d'outil), si la procédure de mesure considérée est complètement ou partiellement automatisable, et dans le cas où elle est partiellement automatisable, quelles en sont les tâches automatisables et celles qui ne le sont pas (ou plutôt qui seraient difficilement automatisables dans le contexte).

- *Une catégorisation explicite des connaissances associées au processus de mesure (séparation claire entre connaissances « structurales » et connaissances opérationnelles ou «procédurales ») :* l'approche préconise, relativement aux méthodes de mesure, l'exploitation de deux (2) sortes d'ontologies, l'une spécifique au domaine et l'autre spécifique aux tâches, ce qui permet de distinguer les connaissances «structurales » (concepts de mesure et liens entre concepts), des connaissances opérationnelles (tâches/règles de mesure). Cette séparation offre une certaine flexibilité dans l'automatisation : Ces deux types de connaissances peuvent être mis à jour séparément. De plus, l'on pourrait simuler l'automatisation dans un système à base d'agents dits « intelligents », les connaissances de domaine étant utilisées dans les communications entre agents et chaque agent étant doté d'un ensemble de connaissances opérationnelles (tâches et/ou règles de mesure). Un système à base d'agents constituerait un plus à la modularité d'un système d'automatisation de la mesure.

- *Une réutilisation des connaissances:* les ontologies à la base de l'approche sont développées une seule fois et font l'objet d'un consensus. Si une ontologie existe déjà pour une procédure de mesure donnée, alors tous les concepteurs d'outils d'aide à la mesure (documentation, audit, analyse, diagnostic…) pour la méthode de mesure considérée peuvent réutiliser telle quelle l'ontologie existante. De plus, dans le cas où l'on voudrait bâtir un outil général de mesure de la taille fonctionnelle des logiciels prenant en compte toutes les principales méthodes de mesure existantes, ou encore un outil de conversion des mesures inter-méthodes, l'on pourrait simplement intégrer les ontologies existantes associées à chacune des méthodes.

- *Une exportabilité des résultats de mesure :* L'approche garantit grâce aux ontologies de domaine, que les résultats produits par l'outil construit sont dans un format exportable, et donc exploitables par d'autres outils logiciels de mesure (outils d'analyse/documentation/validation de résultats de mesure, outils d'estimation et de benchmarking...).

- *Une évolutivité des outils de mesure construits :* En tant qu'approche mettant l'emphase sur la manipulation des connaissances en jeu dans les procédures de mesure, les systèmes implantant l'approche pourront s'adapter aisément à l'évolution des procédures de mesure, par ajout ou mise à jour des connaissances.

- *La construction d'outils « boite blanche » :* L'approche permet, grâce aux ontologies de tâches, d'aboutir à des outils dont on comprend le principe de fonctionnement interne. De tels outils pourraient offrir une traçabilité des résultats qu'ils produisent.

Il est à noter que nous ne prétendons pas à travers cette approche pouvoir automatiser au complet les procédures de mesure. De l'analyse du modèle cognitif décrivant une procédure de mesure présentée plus haut, il ressort que certaines parties de ces procédures sont difficilement automatisables.

3.3 LES LIMITES DE L'AUTOMATISATION DES PROCEDURES DE MESURE

L'**étape d'extraction** au cours de laquelle les documents de spécifications sont parcourus à la recherche d'éléments de spécifications jugés pertinents relativement à la mesure, est à notre avis difficilement automatisable, en l'état actuel des recherches en

sciences de la cognition, notamment lorsque les spécifications sont disponibles dans le format textuel. Les travaux de recherche sur l'analyse textuelle pourraient servir, mais pour l'instant nous ne pensons pas qu'ils soient suffisamment à point pour être exploités dans notre contexte. Toujours est-il que c'est une piste qui reste ouverte pour le futur.

La mise en correspondance (« mapping ») entre concepts de mesure et concepts de spécification, qui est nécessaire pour l'**étape de classification,** est également à notre avis difficilement automatisable, en l'état actuel des recherches sur le «mapping ». En effet, comme le fait remarquer Jayant Madhavan [Madhavan et al. 02], le processus de « mapping » peut difficilement être automatisé au complet. A défaut, il s'agit de pouvoir au moins représenter de façon exploitable les correspondances établies entre concepts.

Ainsi, **l'automatisation des procédures de mesure à la date d'aujourd'hui ne saurait être complète**. Cependant des espoirs sont toujours permis pour le futur avec les travaux de recherche sur l'analyse textuelle et sur le « mapping ». En attendant, une automatisation partielle est un pas décisif vers la facilitation de l'application d'une méthode de mesure. Nous présentons dans la section qui suit une architecture générale pour un système automatisant une bonne partie du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir de spécifications présentées dans un formalisme bien défini, dans le cadre de l'approche que nous proposons.

4 ARCHITECTURE GENERALE D'UN SYSTÈME D'AUTOMATISATION D'UNE PROCÉDURE DE MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS A PARTIR DE SPÉCIFICATIONS PRÉSENTÉES DANS UN FORMALISME BIEN DÉFINI

Un système d'automatisation de la procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels à partir de spécifications présentées dans un formalisme bien défini, comporte trois (3) principales composantes (Figure 24) : Une composante pour la gestion des connaissances associées à la procédure (*Gestionnaire Base de connaissances*), une composante pour la gestion la mesure (*Gestionnaire Mesure*) et une composante pour l'analyse des résultats de mesure (*Gestionnaire Analyse des résultats de mesure*).

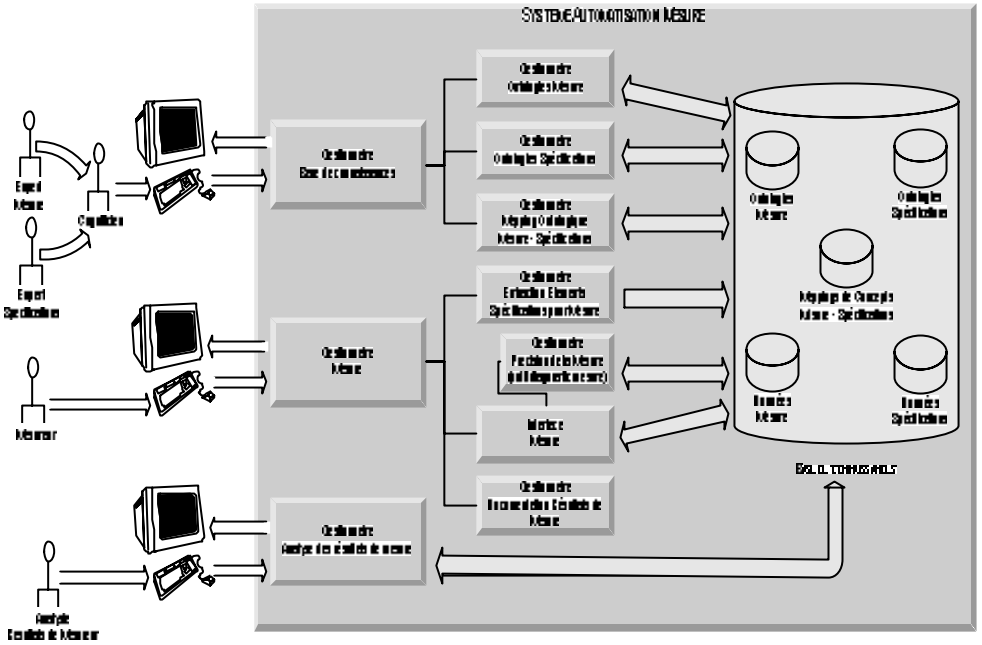


Figure 24 : Architecture générale d'un système d'automatisation de la procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels à partir de spécifications présentées dans un formalisme bien défini

4.1 LES COMPOSANTES

4.1.1 COMPOSANTE *GESTIONNAIRE BASE DE CONNAISSANCES*

La composante *Gestionnaire Base de connaissances* compte trois (3) principaux modules :

- Un module dédié à la gestion des ontologies de domaine et de tâches associées à une méthode de mesure (*Gestionnaire Ontologies Mesure*). En fait, ce module ne se substitue pas aux outils existants de gestion des ontologies. Il permet en fait d'importer (complètement ou partiellement) des ontologies produites à l'aide de tels outils, de les stocker dans la base de connaissances du système en vue de leur exploitation dans le cadre de la mesure;
- Un module dédié à la gestion des ontologies de domaine définies par les langages de spécification de logiciels supportés par l'outil (*Gestionnaire Ontologies Spécifications*). Tout comme le module précédent, ce module se limite à l'importation (complète ou

partielle) des ontologies produites à l'aide d'outils appropriés, et leur stockage dans la base de connaissances du système en vue d'une exploitation dans le cadre de la mesure;

- Un module dédié à la gestion des correspondances entre concepts de mesure et concepts des langages de spécification de logiciels supportés par l'outil (*Gestionnaire Mapping Ontologique Mesure - Spécifications*). Les correspondances entre concepts de différentes méthodes de mesure permettent une convertibilité inter méthodes de mesure des résultats de mesure. Toutes les correspondances doivent être stockées de façon adéquate dans la base de connaissances. Certains mécanismes de base de données (contraintes d'intégrité référentielle, triggers, procédures stockées, etc.) peuvent être mis à contribution lorsque la base de connaissances est implantée dans un SGBD (Système de Gestion de Bases de Données).

4.1.2 COMPOSANTE *GESTIONNAIRE MESURE*

Tout comme la composante précédente, la composante *Gestionnaire Mesure* compte trois (3) principaux modules :

- Un module en charge de l'extraction des éléments de spécifications pertinents pour la mesure (*Gestionnaire Extraction Eléments de spécifications pour Mesure*). L'approche « parseurs » est adoptée pour ce module : Il est en fait constitué d'un ensemble de « parseurs », un pour chaque outil CASE de spécification. Chaque «parseurs » a en charge l'extraction dans une spécification produite à l'aide d'un outil CASE spécifique suivie du stockage de façon appropriée, d'instances de concepts de spécification nécessaires pour la mesure de la taille fonctionnelle. Dans le cas de plusieurs outils CASE qui permettent de produire des spécifications dans un format standard (par exemple XMI), l'on pourrait penser à un seul parseur pour tous ces outils. Cette idée a été explorée et développée dans le cadre d'un mémoire de maîtrise déposé en 2003 [Vilus 03];

- Un module *interface de mesure* pour le choix des applications à mesurer, la visualisation des résultats de mesure, la gestion de la précision des résultats de mesure (*Gestionnaire Précision de la mesure*). Le sous module *Gestionnaire Précision de la Mesure* pourrait interfacer un outil de diagnostic pour la mesure tel que celui proposé par Desharnais dans sa thèse [Desharnais 03]. Un tel outil se chargerait du calcul de la

précision/certitude associée à chaque élément de mesure, tandis que le module *Gestionnaire Précision de la Mesure* s'occuperait de l'agrégation de ces précisions/certitudes individuelles (suivant la théorie de Stanford par exemple) pour déterminer la précision/certitude associée au résultat final de la mesure;

- Un module pour la gestion de la documentation des résultats de mesure (*Gestionnaire Documentation Résultats de Mesure*). Pour la méthode de mesure COSMIC-FFP par exemple, la documentation des résultats de mesure inclut les informations suivantes [Abran et al. 03, ISO/IEC 03a] :
  - identification de chaque morceau de logiciel mesuré dans le cadre d'un projet de mesure donné (nom, identification de la version ou de la configuration);
  - description de la raison d'être et l'étendue de la mesure (perspective de la mesure);
  - description des liens entre morceaux de logicie l mesurés, utilisateurs et positions des frontières;
  - taille fonctionnelle de chaque morceau de logiciel mesuré;
  - pour chaque morceau de logiciel mesuré :
    - la liste des processus fonctionnels identifiés avec pour chacun les mouvements de données qui le composent;
    - la liste des groupes de données identifiées ;
    - le nombre total de points de processus fonctionnels identifiés ;
    - le nombre total de groupes de données identifiées ;
    - le nombre total d'entrées (Entries) ;
    - le nombre total de sorties (Exits) ;
    - le nombre total de lectures (Reads) ;
    - le nombre total d'écritures (Writes).

4.1.3 COMPOSANTE *GESTIONNAIRE ANALYSE DES RÉSULTATS DE MESURE*

La composante *Gestionnaire Analyse des résultats de mesure* est dédiée à l'analyse des résultats de mesure. Elle est en fait une composante d'aide à l'analyse des résultats de mesure, en vue d'en faire ressortir des caractéristiques éventuelles pouvant aider à la prise de décision. Elle pourrait être couplée avec un outil de Data Mining.

Chacune des trois (3) composantes principales présentées ci-haut correspond à un intervenant spécifique dans une procédure de mesure.

4.2 LES ACTEURS

Les intervenants dans une procédure de mesure sont les suivants :

- le *cogniticien* qui identifie et modélise les connaissances en jeu dans une procédure de mesure (celle de COSMIC-FFP par exemple) en vue de les stocker dans la base de connaissances du système afin de l'adapter à l'évolution éventuelle de la procédure (un tel utilisateur a besoin pour cela de connaissances en mesure et en modélisation de connaissances, sinon il doit faire appel à un *expert en mesure* et à un *expert en modélisation de connaissances*);
- le *mesureur* qui a pour tâche de mesurer la taille fonctionnelle des applications (utilisateurs des procédures de mesure). Ils ont également accès en consultation aux connaissances en jeu dans une procédure de mesure (celle de COSMIC-FFP par exemple), pour une meilleure compréhension des procédures;
- l'*analyste des résultats de mesure,* qui s'intéresse aux extrants de la procédure de mesure.

5 SYNTHÈSE

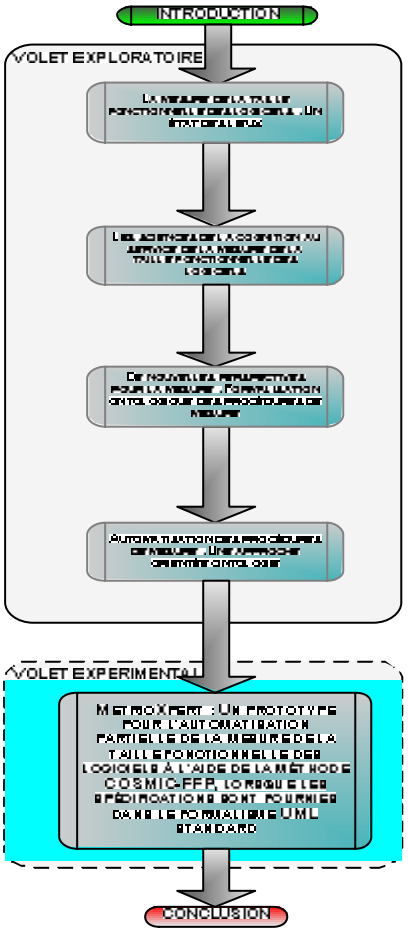
En définitive, en prenant appui sur un modèle cognitif décrivant une procédure de mesure, l'approche orientée ontologie que nous préconisons dans le cadre du développement d'outils d'automatisation des procédures de mesure, met l'accent sur le stockage et l'exploitation des connaissances associées au processus d'application d'une méthode de mesure. Le travail d'automatisation est basé sur l'implantation des ontologies associées aux

procédures de mesure (ontologies de tâches et de domaine). Cette approche permet d'aboutir à des outils présentant bon nombre de caractéristiques souhaitables pour les outils d'automatisation de la mesure, tels qu'identifiées par Keith Paton et Alain Abran [Abran et al. 97] (architecture ouverte, flexibilité, respect des normes ou des standards reconnus, etc.). Parmi les points forts de l'approche, nous relevons qu'elle favorise : un design plus précis des méthodes de mesure, une catégorisation explicite des connaissances associées au processus de mesure (séparation claire entre connaissances « structurales » et connaissances opérationnelles ou « procédurales »), la réutilisation des connaissances, l'exportabilité des résultats de mesure, l'évolutivité des outils de mesure construits et la construction d'outils « boite blanche ». De plus, il est possible de déterminer formellement (avant toute construction d'outil) si une procédure de mesure est complètement automatisable, et le cas échéant, quelles en sont les tâches automatisables et celles qui ne le sont pas.

Afin d'illustrer cette approche, nous présentons dans le chapitre suivant un prototype développé suivant l'approche, pour l'automatisation partielle de la mesure à l'aide de la méthode COSMIC-FFP, lorsque les spécifications sont produites en UML avec certains outils CASE (Together, Objecteering et Rational Rose).

CHAPITRE 5.

METRICXPERT : UN PROTOTYPE POUR L'AUTOMATISATION PARTIELLE DE LA MESURE DE LA TAILLE FONCTIONNELLE DES LOGICIELS À L'AIDE DE LA MÉTHODE COSMIC-FFP



FEUILLE DE ROUTE

1. *Introduction*
2. *Le prototype*
3. *Evaluation du prototype & Analyse des résultats*
4. *Synthèse*

EN BREF …

Ce chapitre présente le prototype qui a été construit pour l'automatisation partielle de la mesure de la taille fonctionnelle des logiciels à l'aide de méthode COSMIC-FFP, lorsque les spécifications sont présentées dans le formalisme UML standard. Ce prototype a été construit suivant l'approche introduite plus haut. Pour des raisons de temps (la thèse s'inscrivant dans un espace limité dans le temps), il n'implante pas toutes les parties de l'architecture générale présentée plus haut, mais uniquement celles jugées essentielles pour illustrer l'approche. Les spécifications fonctionnelles et les contraintes associées au prototype sont indiquées. Les choix technologiques effectués pour la construction sont également précisés. La stratégie ainsi que les résultats de l'évaluation du prototype sont présentés. Une analyse des résultats d'évaluation est proposée

1 INTRODUCTION

Tel que mentionné plus haut, COSMIC-FFP est une méthode de mesure de la taille fonctionnelle des logic iels maintenue par le groupe COSMIC, consortium regroupant un ensemble d'experts du domaine des métriques à travers le monde. Elle est devenue (avec MkII-FPA et FPA) depuis mars 2003, un standard ISO pour la mesure de la taille fonctionnelle des logiciels. Le choix de cette méthode pour l'illustration de l'approche que nous avons introduite plus haut (chapitre quatrième) fait suite à un certain nombre de considérations :

- ***Intérêt économique* :** Parmi les méthodes les plus utilisées aujourd'hui (ou qui font l'objet de nombreux travaux de recherche) dans le domaine de la mesure de la taille fonctionnelle des logiciels (et donc qui ont plus de chance d'être implantées dans l'industrie), figure sans ambages la méthode COSMIC-FFP. En effet, selon ses auteurs, la méthode COSMIC-FFP suscite de plus en plus d'intérêt au niveau de l'industrie, à travers le monde, du moins si nous en jugeons par l'accueil réservé à la version 2.0 de la méthode [Oligny et al. 00, Abran et al. 00] : Des présentations portant sur la méthode ont été faites dans plusieurs pays à travers le monde, et dans certains cas des tests jugés concluants ont été réalisés dans des entreprises ou organisations (Australie, Allemagne, Canada, Japon, Canada, Grange Bretagne, France, Belgique, États-Unis). Le groupe COSMIC la présente comme étant de la nouvelle génération des méthodes de mesure de la taille des logiciels.

- ***Intérêt technico-économique* :** Des méthodes de mesure de la taille fonctionnelle des logiciels qui existent à ce jour, COSMIC-FFP est celle qui permettrait de mesurer la taille fonctionnelle des logiciels aussi bien de gestion (MIS) que temps réels ou embarqués [Abran et al. 99a, Abran et al. 00]. Ceci lui confère plus d'espoir pour l'avenir et un champ d'application plus vaste, par rapport aux autres méthodes qui ne s'avèrent efficaces que pour une seule catégorie de logiciels (MIS en général).

- ***Intérêt par rapport à la recherche :*** L'équipe de recherche du Laboratoire de Recherche en Gestion des Logiciels (laboratoire au sein duquel le projet a été initiée), de concert avec des partenaires de l'industrie [Abran et al. 99a], travaille actuellement à la

vulgarisation de la méthode, son renforcement, la facilitation (simplification) de son d'application (question d'augmenter ses chances d'implantation dans l'industrie)[35]. L'étude sur l'automatisation de son processus d'application pourrait contribuer à la facilitation de son application et au renforcement de la méthode (notamment en soulignant les éventuelles difficultés rencontrées pour l'automatisation). Pour le moment, **deux études seulement ont été complétées sur l'automatisation du processus d'application de la méthode à partir des spécifications [Diab et al. 01, Azzouz et al. 04]**. Contrairement aux autres, notre étude se veut plus générale (non spécifique à un outil CASE en particulier).

Pour ce qui est du choix du langage UML (Unified Modeling Language) [Rumbaugh et al. 04, OMG 03, Booch et al. 99] comme langage de spécification de logiciels pour l'illustration de l'approche que nous avons introduite plus haut (chapitre quatrième). Il tient à l'expressivité, la notoriété et la popularité du formalisme. UML fournit un vocabulaire et des règles assez claires, pour représenter les différents modèles permettant de comprendre (de visualiser) un système. Cette notation a été adoptée comme standard par l'O.M.G. (Object Management Group) depuis novembre 1997, et a été proposée à l'I.S.O. par l'O.M.G. pour devenir un standard en technologies de l'information. UML *s'imposant peu à peu* comme une notation standard pour les méthodes d'analyse et de conception orientées-objet, *de plus en plus de documents* de spécifications de logiciels seront basés sur cette notation. Plusieurs outils CASE pour la spécification des logiciels basés sur cette notation existent à ce jour : Entreprise Architech (Cie Sparse), Objecteering (Cie Softeam), Rational Rose (Cie Rational), Together, etc. L'on pourrait donc associer à de tels outils une composante pour la mesure de la taille fonctionnelle des logiciels spécifiés. Ainsi, un système automatisant la mesure de la taille fonctionnelle des logiciels à l'aide de la méthode COSMIC-FFP, ou de toute autre méthode lorsque les spécifications sont fournies dans le formalisme UML standard, revêt un intérêt certain. L'utilisation de la notation standardisée UML pour la modélisation permet d'uniformiser les modèles de logiciels qui servent de paramètres d'entrée pour chacune des méthodes de mesure. Ainsi, les concepteurs de ces méthodes n'auraient plus qu'à assurer la qualité du « Mapping » requis entre le modèle du

[35] http://www.lrgl.uqam.ca

logiciel en UML et la méthode de mesure, sans inférence aucune dans la processus de modélisation.

La section suivante (section 2), est consacrée à la présentation dans son ensemble du prototype que nous avons construit. Nous l'avons baptisée **MetricXpert**. Nous survolons l'ensemble des concepts ayant permis la réalisation du prototype ainsi que l'ensemble des exigences qu'il est appelé à satisfaire. Dans la section, nous nous limitons juste à l'essentiel, les autres détails étant fournis dans un document de spécifications et exigences logicielles qui sera produit ultérieurement dans le cadre de la construction d'un outil commercial. Les résultats d'évaluation du prototype ainsi qu'une analyse de ces résultats, sont au cœur de la section 3.

2 LE PROTOTYPE: DESCRIPTION GÉNÉRALE

2.1 PORTÉE

L'objectif principal que nous visons à travers la construction du prototype MetricXpert est l'illustration de l'approche introduite dans le chapitre troisième de cette thèse pour l'automatisation de la mesure de la taille fonctionnelle des logiciels. Ainsi, il a une portée beaucoup plus exploratoire, illustrative, que commerciale. Mais il pourrait servir de base à la construction d'un outil commercial. Par conséquent certaines fonctionnalités n'ont pas été implantées dans le cadre de la thèse. Seules les fonctionnalités jugées pertinentes pour l'illustration ont été implantées et sont détaillées ici.

2.2 ENVIRONNEMENT OPERATIONNEL

2.2.1 LES INTERFACES DU PROTOTYPE

Le prototype doit idéalement pouvoir interfacer :

- un outil diagnostic de détermination de la pertinence associée à chaque concept de mesure identifié dans des spécifications
- un outil d'analyse de résultats de mesure

- un outil CASE de spécification UML de logiciels (par exemple Rational Rose, Tau, Together, etc.)
- un outil de développement d'ontologies

2.2.2 LES INTERFACES UTILISATEURS

Les caractéristiques des interfaces utilisateurs sont les suivantes :

- écrans classiques ressemblant aux écrans Windows de Microsoft.
- Regroupement des différentes fonctions dans des menus et des sous-menus.
- Utilisation des touches de raccourci pour les fonctions qui sont fréquemment utilisées.
- Utilisation des messages d'erreurs et d'avertissement à la fois courts et précis.

2.2.3 LES COMPOSANTS MATERIELS

Le matériel requis pour la mise en place du prototype est le suivant : Un poste serveur de base de données sur lequel se trouvera la base de connaissances et éventuellement un ou plusieurs postes clients sur lesquels seront installés le prototype (une version application Web du prototype est envisagée).

2.2.4 LES COMPOSANTS LOGICIELS

Les logiciels suivants sont requis pour la mise en place du prototype :

- Le logiciel de gestion de base de données Oracle 8i et version postérieure, version opérant sur les systèmes d'exploitation Windows NT/2000/XP, Linux, etc.
- Un système d'exploitation (Windows, Linux, Unix, etc.)
- JVM (Java Virtual Machine) pour l'exécution du prototype.

2.3 VUE D'ENSEMBLE DES FONCTIONS DU PROTOTYPE

MetricXpert permettra dans sa version finale :

- l'importation des spécifications UML d'un logiciel (spécifications stockées dans le format XMI), d'en extraire les éléments jugés pertinents pour la mesure et les stocker dans la base de données du système;
- l'ajout, la modification, la suppression, la recherche et la présentation des éléments de spécifications UML d'un logiciel, jugés pertinents pour la mesure ;
- l'importation (pour des besoins d'analyse par exemple) d'un modèle COSMIC-FFP d'un logiciel mesuré à l'aide d'un autre outil de mesure et stocké dans le format XMI (suivant l'ontologie de domaine associée à la procédure de mesure COSMIC-FFP) *(fonctionnalité non implantée);*
- l'ajout, la modification, la suppression, la recherche et la présentation des éléments d'un modèle COSMIC-FFP d'un logiciel mesuré;
- la documentation des résultats de mesure de la taille fonctionnelle d'un logiciel, de la façon préconisée dans le manuel ISO de COSMIC-FFP *(fonctionnalité non implantée);*
- la visualisation, l'analyse des résultats de mesure (statistiques, graphiques, camemberts, etc.) *(fonctionnalité non implantée);*
- la mise à jour de l'ontologie de domaine associée à la méthode de mesure COSMIC-FFP (ajout, mise à jour ou suppression de concepts) *(fonctionnalité non implantée);*
- la mise à jour de l'ontologie de domaine associée au langage de spécifications UML (ajout, mise à jour ou suppression de concepts) *(fonctionnalité non implantée);*
- la mise en correspondance (« mapping ») entre concepts de spécifications (UML) et concepts de mesure (COSMIC-FFP);
- la gestion de l'incertitude/imprécision associée aux résultats de mesure (du fait de l'incomplétude ou de l'imprécision des spécifications par exemple) *(fonctionnalité non implantée).*

2.4 DESCRIPTION DES UTILISATEURS

Nous identifions quatre (4) types d'utilisateurs possibles pour le système MetricXpert :

- les «*cogniticien* », capables d'identifier et de modéliser les connaissances en jeu dans une procédure de mesure (celle de COSMIC-FFP par exemple) en vue de les stocker dans la base de connaissances du système afin de l'adapter à l'évolution éventuelle de la procédure (un tel utilisateur a besoin pour cela de connaissances en mesure et en modélisation de connaissances, sinon il doit faire appel à un *expert en mesure* et à un *expert en modélisation de connaissances*);
- les *mesureurs*, qui ont pour tâche de mesurer la taille fonctionnelle des applications (utilisateurs des procédures de mesure). Ils ont également accès en consultation aux connaissances en jeu dans une procédure de mesure (celle de COSMIC-FFP par exemple), pour une meilleure compréhension des procédures;
- les *analystes des résultats de mesure*, qui s'intéressent aux extrants de la procédure de mesure;
- L'*administrateur du système*, qui gère et alloue les droits d'accès au système, s'assure de son bon fonctionnement. Il doit connaître les procédures des opérations de tous les utilisateurs du système, doit bien maîtriser le logiciel de MetricXpert et doit avoir les connaissances techniques suffisantes pour en assurer la maintenance.

2.5 CONTRAINTES D'ORDRE GÉNÉRAL

Dans le cadre de la thèse, nous allons nous contenter d'un prototype partiel, qui prend uniquement en entrée les diagrammes UML de cas d'utilisations, les diagrammes de séquences correspondants ainsi que les diagrammes de classes du domaine (obtenus dès la phase d'analyse et contenant les données qui seront manipulées et stockées par les systèmes considérés) [Lethbridge et al. 01]. Nous nous limiterons également à la version standard du formalisme UML pour les spécifications.

Le prototype automatisera la construction du modèle COSMIC-FFP d'un logiciel à mesurer, à partir des spécifications dudit logiciel présentées dans le formalisme UML standard, et des résultats de la mise en correspondance (« mapping ») entre concepts UML et concepts COSMIC-FFP (*Annexe E*). L'application des règles d'assignation numérique et l'agrégation des résultats partiels en vue d'obtenir la taille fonctionnelle du logiciel, seront

également automatisées. La gestion du « mapping » ainsi que les autres fonctionnalités mentionnées plus haut seront examinées dans des travaux de recherche subséquents.

2.6 HYPOTHÈSES

- La taille fonctionnelle fournie par le prototype est indicative de la taille fonctionnelle réelle d'un morceau de logiciel mesuré. Ultérieurement (avec les travaux subséquents à la thèse) elle sera fournie avec un intervalle de confiance.
- Le modèle COSMIC-FFP d'un morceau de logiciel mesuré peut être ajusté manuellement par le mesureur afin d'affiner le résultat de la mesure

D'autres détails techniques relatifs au prototype sont fournis en annexe F de la thèse (spécifications fonctionnelles, modèle structural, etc.). La section qui suit est consacrée à l'évaluation du prototype.

3 ÉVALUATION DU PROTOTYPE ET ANALYSE DES RESULTATS

3.1 STRATÉGIE D'ÉVALUATION

L'évaluation du prototype construit s'est orientée vers deux (2) aspects du prototype :

- L'aspect **fonctionnel** : le prototype est-il conforme aux spécifications établies au début de ce volet du projet ? Pour cela un jeu de tests basé sur les cas d'utilisation recensés permet de tester chaque fonctionnalité implantée du système. Dans le cadre de la thèse nous nous sommes limités aux fonctionnalités jugées pertinentes par rapport aux objectifs visés par la construction du prototype (celles qui ont été présentées plus haut dans la section 2.2 consacrée aux spécifications fonctionnelles du prototype).
- L'aspect **précision des résultats** : le prototype fournit-il la « bonne » taille fonctionnelle (taille qui correspond à celle obtenue par les experts) ? Nous avons choisi deux sous-ensembles d'applications dont on a la taille fonctionnelle obtenue par des experts et dont on dispose des spécifications présentées dans le formalisme UML (Dans certains cas, les spécifications UML ont été produites par des étudiants de maîtrise que nous avions à notre disposition). Le premier sous-ensemble sera constitué de «petites » applications

(applications de taille inférieure à 100 CFSUs) tandis que le second contiendra des applications de taille relativement grande (applications de taille supérieure à 100 CFSUs). Dans le contexte académique qui est le nôtre, nous avons limité la taille de chaque sous-ensemble à trois (3) applications pour le premier sous-ensemble et deux (2) applications pour le second sous-ensemble. Il est entendu que dans le cadre de la thèse, **le prototype vise simplement à illustrer l'approche proposée dans le volet exploratoire du projet** pour l'automatisation d'une bonne partie de la procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels à partir de spécifications présentées dans un formalisme bien défini. Cependant, notre échantillon de test est représentatif des types de logiciels que l'on peut mesurer avec COSMIC-FFP. Ainsi, nous y avons inclus des applications de type temps-réel et des applications de type MIS (application de gestion). Pour chaque application l'écart entre la taille fonctionnelle réelle (celle obtenue par l'expert) et la taille fonctionnelle obtenue avec le prototype a été relevée. De plus, la différence entre le modèle COSMIC-FFP de l'application selon l'expert et celui obtenu avec l'outil a été examiné. Les quatre premières applications sont des études de cas proposées par le groupe COSMIC et disponibles sur le site web du LRGL. Quant à la dernière application, il s'agit d'une étude de cas disponible dans le livre de Pierre-Alain Muller [Muller 00], dont la mesure COSMIC-FFP a été faite par Malcolm Jenner et vérifiée par Charles Symons (tous les deux spécialistes COSMIC-FFP). Les spécifications UML pour les quatre (4) premières applications que nous avons choisies ont été produites par des étudiants de maîtrise en génie logiciel que nous avions à notre disposition (deux étudiants de l'Université Libre de Bruxelles et un étudiant de l'UQAM). Ces étudiants avaient une certaine expérience dans la spécification UML des logiciels.

Nous présentons ci-dessous le détail des résultats de l'évaluation du prototype.

3.2 LES RESULTATS DE L'ÉVALUATION DU PROTOTYPE

Morceau de logiciel	**Caractéristiques**	**Taille fonctionnelle expert**	Taille fonctionnelle MetricXpert
Rice cooker	• Application conçue pour le contrôle d'une « cuiseuse de riz » (Rice cooker) • Application de type temps-réel	12 CFSUs	16 CFSUs
Valve control system	• Système de contrôle d'une valve (ouverture et fermeture de la valve) • Application de type temps-réel	20 CFSUs	23 CFSUs
ISDN Loop Back Tester (LBT)	• Système de gestion d'un dispositif pouvant aider des officiers à tester à distance l'intégrité d'un circuit ISDN à 4 voies. • Application de type temps-réel	146 CFSUs	129 CFSUs
Gendarme	• Système d'automatisation du traitement d'événements reçus de divers systèmes de surveillance (feu, intrusion) et de contrôle (chaudières, ventilation) • Application de type hybride (une partie temps-réel et une partie « MIS »)	830 CFSUs	686 CFSUs
ContrôleAccès	• **Système de contrôle des accès à un bâtiment** • **Application de type « MIS » et temps réel**	**119 CFSUs**	88 CFSUs

Tableau 1: Synthèse des résultats d'évaluation du prototype

Une analyse de ces résultats d'évaluation est proposée dans la section qui suit.

3.3 ANALYSE DES RÉSULTATS

Le Tableau 2 ci-dessous présente les écarts absolus et relatifs entre la taille fonctionnelle obtenue par des experts et celle obtenue à l'aide de MetricXpert.

Morceau de logiciel	**Taille fonctionnelle expert**	**Taille fonctionnelle MetricXpert**	**Ecart absolu**	Ecart relatif
Rice cooker	12 CFSUs	16 CFSUs	+4 CFSUs	33%

Valve control system	20 CFSUs	23 CFSUs	+3 CFSUs	15%
ISDN Loop Back Tester (LBT)	146 CFSUs	129 CFSUs	-17 CFSUs	11%
Gendarme	830 CFSUs	686 CFSUs	144 CFSUs	17%
ContrôleAccès	**119**	**88 CFSUs**	**31 CFSUs**	26%

Tableau 2 : Tableau analytique des résultats de l'évaluation

D'après le Tableau 2 (ci-dessus), l'écart relatif moyen entre la taille fonctionnelle obtenue par des experts et celle obtenue à l'aide de MetricXpert est de l'ordre de 20%. Un tel écart n'est pas loin de l'écart moyen entre deux mesureurs auquel a abouti Abran (environ 15% pour des mesureurs formés) [Abran 94] ou encore Nishiyama (environ 5% pour des experts) [Nishiyama 99]. De plus, les tailles obtenues via l'outil et inscrites dans les tableaux ci-haut, sont des **tailles brutes**, **non ajustées**. Un ajustement direct de la part du mesureur réduirait davantage l'écart observé (le mesureur pourrait par exemple compléter les spécifications en ajoutant des éléments manquants (séquences omises, données échangées dans certaines séquences, etc.).

De façon plus générale, force est de constater que **le niveau de détail des spécifications disponibles est déterminant pour la mesure** : Les mouvements de données identifiés (leur nombre et leur nature/type) dépendent du niveau de détail des diagrammes de séquences. L'incertitude demeure encore pour ce qui est de la détermination de la nature d'un mouvement de données (« Entrée », «Sortie », «Lecture » ou «Ecriture »). En effet, pour déterminer la nature des mouvements de données, il faut que soit précisés explicitement dans les diagrammes de séquences et ce pour chaque message échangé, la nature de l'expéditeur du message (« sender ») et celle du récepteur (« receiver »). Par exemple, si l'expéditeur est un acteur et le récepteur n'est ni une classe, ni un acteur, alors le mouvement de données correspondant est une « Entrée ». A défaut de retrouver ces renseignements dans les diagrammes de séquences, nous pensons que l'on pourrait les avoir plutôt dans les diagrammes de collaborations qui détaillent les messages échangés entre objets du système. Nous examinerons dans nos prochaines itérations la contribution des diagrammes d'activités pour la mesure. Il est vrai que la nature des mouvements de données

ne change rien à la taille fonctionnelle obtenue, mais elle pourrait servir dans les analyses des résultats de mesure.

Dans tous les cas, nous pensons que la question de l'impact du niveau de détail des spécifications dans la mesure doit être examinée plus minutieusement et une stratégie doit être mise au point dans les outils de mesure pour en tenir compte. L'on pourrait penser par exemple à une pondération du résultat de la mesure en fonction du niveau de détail des spécifications disponibles. L'on pourrait penser également à une mesure qui se limiterait à la détermination des processus fonctionnels, sachant que l'on pourrait associer, sur une base statistique, une taille fonctionnelle approximative à chaque processus fonctionnel. Cette valeur serait fonction du type de logiciel mesuré (temps-réel, « MIS ». etc.), de l'environnement dans lequel le logiciel a été développé.

Par ailleurs, le problème de détermination de la nature des mouvements de données nous a conduits à un autre constat : **l'influence que pourrait avoir la mesure sur les spécifications**. En effet, plus les spécifications sont détaillées et précises, plus il est aisé de déterminer les mouvements de données d'une part, et la nature de chaque mouvement de données d'autre part. Ainsi le modèle COSMIC-FFP (par exemple) d'un logiciel pourrait fournir des indices sur le niveau de détail, la precision des spécifications et indirectement leur qualité.

4 SYNTHÈSE

En définitive, le prototype que nous avons présenté dans ce chapitre illustre bien l'approche que nous avons introduite dans le chapitre troisième de cette thèse pour l'automatisation de la mesure de la taille fonctionnelle des logiciels. Ses éléments architecturaux, structuraux et fonctionnels illustrent bien ceux préconisés dans l'approche. Tous les éléments préconisés n'ont pas été implantés dans le cadre de cette thèse. Seuls les éléments jugés pertinents pour l'illustration ont été implantés et sont détaillées dans ce chapitre. Les résultats d'évaluation du prototype indiquent clairement que l'écart entre les résultats de mesure fournis par le prototype et ceux obtenus par des mesureurs experts n'est pas énorme (de l'ordre de 20% en moyenne). Cependant, le niveau de détail des spécifications pourrait affecter la précision de la mesure (ce qui se vérifie même chez les mesureurs humains). Comment en tenir compte dans le résultat d'une mesure ? Dans le

cadre de nos travaux futurs, nous comptons focaliser notre attention sur cet aspect des choses.

CONCLUSION

Le travail de recherche que nous avons mené comportait deux volets : un volet exploratoire et un volet expérimental. Le volet exploratoire était consacré aux questions de *formalisation et d'automatisation des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels,* à la lumière des possibilités offertes par les sciences de la cognition, notamment en matière de *représentation et exploitation des connaissances*.

Du volet exploratoire, il en ressort que :

- face aux difficultés auxquelles les utilisateurs des méthodes de mesure sont confrontés, lesquelles difficultés justifient ce travail de thèse, la formalisation et l'automatisation des procédures de mesure associées aux méthodes de mesure, sont des pistes prometteuses pour surmonter ou s'attaquer à une bonne partie de ces difficultés;
- les questions de représentation des connaissances, de catégorisation/classification, de « mapping » ou encore de simulation, que l'on retrouve au coeur du débat en sciences cognitives, sont transposables dans le domaine de la mesure de la taille fonctionnelle des logiciels. Ainsi la mesure de la taille fonctionnelle des logiciels pourrait tirer profit des travaux de recherche en sciences cognitives qui traitent ces questions;
- la formalisation et l'automatisation des procédures de mesure associées aux méthodes de mesure pourraient tirer profit des travaux sur la représentation des connaissances. L'illustration est faite dans la thèse avec la formalisation ontologique des procédures de mesure.
- les résultats de la formalisation ontologique des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels, peuvent être exploités dans le cadre de l'automatisation des procédures de mesure. D'où l'idée d'une approche orientée ontologie pour l'automatisation des procédures de mesure. Une telle approche proposée trouve son fondement dans un modèle cognitif décrivant une procédure de mesure. Ce modèle est en fait une généralisation des

ontologies de tâches que nous préconisons dans le cadre de la formalisation ontologique des procédures de mesure.

Le volet exploratoire de la thèse trouve sa concrétisation dans le volet experimental. En effet, ce dernier volet montre entre autres à travers le prototype qui y est présenté, comment les formalisations proposées dans le volet exploratoire pourraient être exploitées dans le cadre du développement d'outils d'aide à la mesure (notamment les outils d'automatisation de la mesure). Le prototype est construit suivant l'approche orientée ontologie proposée dans le volet exploratoire pour l'automatisation. L'évaluation du prototype ainsi que l'analyse des résultats de l'évaluation viennent confirmer la pertinence des hypothèses émises dans ce volet.

Au terme de cette thèse, nous pensons avoir contribué effectivement à l'ouverture de nouvelles perspectives dans le sens d'aider les utilisateurs des méthodes de mesure dans cette tâche non triviale qu'est l'application des méthodes de mesure. Nous présentons dans la section qui suit une synthèse de nos contributions.

1 SYNTHÈSE DES CONTRIBUTIONS

Des principales contributions de cette thèse nous relevons les suivantes :

- Précision du lien entre catégorisation/classification et « design »/application d'une méthode de mesure de la taille fonctionnelle des logiciels : D'une part, le problème auquel est confronté le concepteur d'une méthode de mesure lorsqu'il a à produire un « meta-modèle » associé à la méthode de mesure, peut être perçu en partie comme un problème de catégorisation (catégorisation des éléments de spécfication de logiciels). D'autre part, la phase d'"identification/classification et modélisation" d'une procédure de mesure associée à une méthode de mesure de la taille fonctionnelle des logiciels, consiste en partie en une classification de certains éléments de spécification du logiciel à mesurer (ceux tirés de la documentation du logiciel et jugés pertinents relativement à la mesure) : Pour chaque élément de spécification identifié, il s'agit de déterminer à quelle catégorie il correspond relativement à la méthode de mesure (les catégories dont il est question ici sont celles décrites dans le « meta-modèle » associé à la méthode de mesure). Ainsi, le «design » et l'application d'une méthode de mesure tireraient par exemple profit des travaux sur la catégorisation et la classification, notamment dans la

perspective d'une systématisation des procédures de mesure. Il est à noter que les travaux sur la simulation pourraient être exploités dans une telle systématisation;

- Jet des bases de la formalisation ontologique du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels : (1) formalisation ontologique de la procédure de mesure associée à la méthode COSMIC-FFP [Bévo et al. 03a], (2) formalisation ontologique de la procédure de mesure associée à la méthode Mk II FPA [Bevo et al. 03b], (3) formalisation ontologique de la procédure de mesure associée à la méthode FPA [Bevo et al. 03c]. La formalisation ontologique des procédures de mesure que nous préconisons pour chaque méthode de mesure constitue à notre avis un outil efficace pour unifier l'interprétation des concepts et règles associées aux procédures de mesure, éviter les ambiguïtés terminologiques, proposer une représentation explicite partagée des connaissances relatives aux procédures de mesure, améliorer la communication entre experts et/ou novices, donc permettre une meilleure réutilisation et interopérabilité;

- Jet des bases d'une approche orientée ontologie pour l'automatisation d'une bonne partie des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels lorsque les spécifications sont fournies dans un formalisme bien défini. L'approche est basée sur un modèle cognitif décrivant une procédure de mesure. Elle permet d'aboutir à des outils présentant bon nombre de caractéristiques souhaitables pour les outils d'automatisation de la mesure, tels qu'identifiées par Keith Paton et Alain Abran [Abran et al. 97] (architecture ouverte, flexibilité, respect des normes ou des standards reconnus, etc.). Il est à noter que notre objectif à terme n'était pas de proposer une solution globale à la question d'automatisation du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications, mais d'abord et avant tout d'examiner en profondeur la question, d'identifier les difficultés à surmonter et les principaux défis à relever pour arriver à une automatisation effective;

- Construction d'un prototype permettant d'automatiser une bonne partie du processus d'application de la méthode de mesure COSMIC-FFP à partir de spécifications présentées dans la notation UML standard. Ce prototype illustre bien l'approche que nous avons introduite dans le chapitre troisième de cette thèse pour l'automatisation de

la mesure de la taille fonctionnelle des logiciels. Ses éléments architecturaux, structuraux et fonctionnels illustrent bien ceux préconisés dans l'approche.

Ci-dessous nous présentons quelques perspectives de recherche et des travaux futurs, à la suite de ce projet de thèse.

2 PERSPECTIVES DE RECHERCHE ET TRAVAUX FUTURS

La fin de la thèse ne marque pas la fin de la réflexion qui y a été menée. Au contraire, nous comptons consolider les acquis au moins sur deux axes :

- Validation, consolidation et stabilisation des formalisations ontologiques des procédures de mesure qui ont été produites. Tel que mentionné plus haut (chapitre troisième), l'aide des experts des différentes méthodes de mesure est toujours requise afin de valider, raffiner et consolider les ontologies produites. Le consensus étant l'une des principales finalités des ontologies, la consultation du plus grand nombre d'experts serait à notre avis une bonne façon de bâtir un tel consensus. La thèse s'étant attelée à jeter les bases de la formalisation ontologique des procédures de mesure, à montrer sa pertinence et son utilité pour la mesure, il s'agit maintenant de consolider les acquis, et d'aboutir à un véritable corpus de connaissances généralement acceptées pour chaque méthode de mesure;

- Ajustement et complétion du prototype proposé dans la perspective d'une demande de financement pour le développement d'un outil commercial. Les résultats de l'évaluation du prototype étant plutôt encourageants, nous pensons que sa complétion avec l'implantation de certains modules jugés non prioritaires pour la thèse (module de documentation des mesures, module de visualisation des résultats de mesure, module de gestion des correspondances entre concepts, etc.) augmenterait les chances d'obtention d'un financement pour le développement d'un outil commercial;

Par ailleurs, un certain nombre d'avenues de recherche abordées dans cette thèse et non examinées en profondeur feront l'objet de nos travaux futurs. Ce sont notamment :

- La gestion de la précision associée aux mesures et l'approximation des mesures. La valeur fournie par un outil de mesure indiquant la taille fonctionnelle d'un logiciel donné en entrée, devrait être accompagnée d'une indication précisant l'intervalle de

confiance de la mesure effectuée. En effet, la qualité de la documentation est un paramètre non négligeable dans le processus d'application d'une méthode de mesure. Elle affecte l'identification des éléments pertinents pour la mesure dans les spécifications, et donc la mesure. A chaque élément identifié l'on pourrait par exemple associer un paramètre indiquant le degré de certitude du mesureur lors de l'identification. Ainsi un outil de mesure pourrait interfacer un outil de diagnostic pour la mesure tel que celui proposé par Desharnais dans sa thèse [Desharnais 03]. Un tel outil diagnostic se chargerait du calcul de la précision/certitude associée à chaque élément de mesure, tandis que le module *Gestionnaire Précision de la Mesure* s'occuperait de l'agrégation de ces précisions/certitudes individuelles (suivant la théorie de Stanford par exemple) pour déterminer la précision/certitude associée au résultat final de la mesure ;

- La question de la convertibilité inter méthodes de mesure. Une nouvelle approche pour s'attaquer à la question de convertibilité inter-méthodes est en chantier. Cette approche est orientée ontologie: les liens entre ontologies de domaine associées aux procédures de mesure pour les méthodes considérées, doivent être formellement établis. Les travaux actuels sur les liens entre ontologies seront mis à contribution [Madhavan et al. 02]. ;

RÉFÉRENCES

[Aamodt et al. 94] Aamodt, A.; and Plaza, E., "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches", In *Artificial Intelligence Communications, IOS Press*, Vol. 7:1, pp. 39 – 59, 1994.

[Abran et al. 03] Abran, A.; Desharnais, J.-M.; Oligny, S.; St-Pierre, D.; and Symons, C., COSMIC FFP – Measurement Manuel, version 2.2, Montreal, March, 2003.

[Abran et al. 01] Abran, A.; Desharnais, J.-M.; Oligny, S.; St-Pierre, D.; Symons, C., *COSMIC FFP - Manuel de mesures version 2.1 - Essais sur le terrain*, Montréal, Mai, 2001.

[Abran et al. 00] Abran, A.; Symons, C.; Desharnais, J.-M.; Fagg, P.; Morris, P.; Oligny, S.; Onvlee, J.; Meli, R.; Nevalainen, R.; Rule, G.; St-Pierre, D., "COSMIC FFP Field Trials Aims, Progress and Interim Findings", in *the 11th European Software Control and Metric Conference (ESCOM SCOPE 2000)*, Munich, Germany, April 18-20, 2000, 31 p.

[Abran et al. 99a] Abran, A., "FFP Release 2.0 : An implementation of COSMIC Functional Size Measurement Concepts", in *FESMA99*, Amsterdam, October 4-7,1999.

[Abran et al. 99b] Abran, A.; Jacquet, J.-P., "A Structured Analysis of the New ISO Standard on: 'Functional Size Measurement - Definition of Concepts' (ISO/IEC 14143-1)", in *4th IEEE International Software Engineering Standards Symposium, ISESS'99*, Curitiba, Brazil, May 17-22, 1999.

[Abran et al. 97] Abran, A.; Paton, K.,"Automation of Function Points Counting: Feasibility and Accuracy?", 1997, pp. 11.

[Abran 94] Abran, A., *Analyse du processus de mesure des points de fonction*, in Département de génie électrique et de génie informatique, Montréal, École polytechnique de Montréal, 1994, pp. 405 (p.21).

[Arpírez et al., 01] Arpírez, J., C.; Corcho, O.; Fernández-López, M.; Gómez-Pérez, A., "WebODE: a Scalable Workbench for Ontological Engineering", In *First International Conference on Knowledge Capture (KCAP01)*, Victoria. Canada. October, 2001.

[Ashby et al. 01] Ashby, F. G., Ell, S. W., *The neurobiology of human category learning*, TRENDS in Cognitive Sciences, Vol. 5, No. 5, 2001.

[Azzouz et al. 04] Azzouz, S.; Abran, A., "A proposed measurement role in the Rational Unified Process (RUP) and its implementation with ISO 19761: COSMIC FFP", in *SMEF2004*, Italy, January 28-30 2004.

[Basili et al. 86] Basili, V., R.; Selby, R., W.; and Hutchens, D., H., "Experimentation in software engineering", *IEEE Transactions on Software Engineering*, SE-9, p. 733-743 (1986).

[Bareiss 89] Bareiss, R., *Exemplar-based knowledge acquisition: A unified approach to concept representation, classification, and learning* Academic Press, Boston, 1989.

[Bechhofer et al. 01] Bechhofer, S.; Horrocks, I.; Goble, C.; and Stevens, R., "OILEd: a Reason-able Ontology Editor for the Semantic Web", In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence*, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.

[Beckett 04] Beckett Dave, "RDF/XML Syntax Specification (Revised)", W3C Recommendation, 10 February 2004, **http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/**. (Latest version available at http://www.w3.org/TR/rdf-syntax-grammar).

[Bévo et al. 03a] Bévo, V., Lévesque, G., and Meunier, J.-G., "Toward an ontological formalisation for a software functional size measurement method's application process: The COSMIC-FFP case", in *IWSM'03*, Montréal, Canada, September 23-25, 2003.

[Bévo et al. 03b] Bévo, V., Lévesque, G., and Meunier, J.-G., "Toward an ontological formalisation for a software functional size measurement method's measurement procedure: The MkII-FPA case", in *ICSSEA'03*, CENAM - Paris, France, December 2-4, 2003.

[Bévo et al. 03c] Bévo, V., Lévesque, G., and Meunier, J.-G., "Toward an ontological formalisation for software functional size measurement method's application process: The FPA case", in *RIVF'04*, Hanoi, Vietnam, February 2-5, 2004.

[Bévo et al. 01] Bévo, V.; Lévesque, G.; Abran, A.; and Meunier, J.-G., "Vers une approche multi-agent pour la mesure de la taille fonctionnelle des logiciels", in *IWSM'01*, Montréal, Canada, 2001.

[Bévo et al. 99] Bévo, V.; Lévesque, G.; and Abran, A., "Application de la méthode FFP à partir d'une spécification selon la notation UML: compte rendu des premiers essais d'application et questions", in *IWSM'99*, Lac Supérieur, Canada, September 8-10, 1999.

[Black et al. 99] Black, S.; and Wigg, D., "X-Ray : A Multi-Language, Industrial Strengh Tool", in *IWSM'99*, Lac Supérieur, Canada, p.39, Sept.ember 8-10, 1999.

[Booch et al. 99] Booch, G.; Rumbaugh, J.; and Jacobson, I., *The Unified Modeling Language User Guide*, 1999.

[Barrows et al. 95] Barrows, H.; and Kelson, A., C., *Problem-based learning in secondary education and the problem-based learning institute*, (Monograph 1), Springfield IL: Problem-Based Learning Institute, 1995.

[Bourque et al. 91] Bourque, Pierre; and Coté, Vianney, "An Experiment in Software Sizing with Structured Analysis Metrics", *Journal of Systems and Software*, Vol. 15, n° 2, p. 159 - 172, (1991).

[Bradford 97] Bradford, K. Clark, *The Effects of Software Process Maturity on Software Development Effort*, in Faculty of the Graduate School, University of Southern California, 1997, pp.129 (p.6).

[Brickley et al. 04] Brickley, Dan, and Guha, R., V., "RDF Vocabulary Description Language 1.0: RDF Schema", W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. (Latest version available at http://www.w3.org/TR/rdf-schema/).

[Brickley, 03] Brickley, D.; and Guha, R., V., *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Working Draft, 2003, http://www.w3.org/TR/rdf-schema/.

[Brill 93] Brill Dave, *Loom Reference Manual*, version 2.0, December 1993, http://www.isi.edu/isd/LOOM/documentation/manual/quickguide.html.

[Britt 97] Britt, David, W., *Conceptual Introduction to Modeling : Quantitative and Qualitative Perspectives*, p. 2, IEA Lawrence Erlbaum Associates, Mahwah, New Jersey, 1997.

[Broekstra et al. 01] Broekstra, J.; Klein, M.; Decker, S.; Fensel, D.; van Harmelen, F.; and Horrocks, I., "Enabling knowledge representation on the Web by Extending RDF Schema", in *Proceedings of the Tenth International World Wide Web Conference*, 2001.

[Ceusters et al. 01] Ceusters, W.; Martens, P.; Dhaen, C.; and Terzic, B., "LinkFactory: an Advanced Formal Ontology Management System", In *Proceedings of Interactive Tools for Knowledge Capture, KCAP-2001*, Victoria, October 20, 2001,

[Collins et al. 88] Collins, A.; and Smith, E., E., *Readings in cognitive science : A perspective from Psychology and Artificial Intelligence*, Morgan Kaufmann Publishers, INC, San Mateo,California, 1988.

[Denny 02] Denny, M., "Ontology Building: A Survey of Editing Tools", at (*http://www.xml.com/pub/a/2002/11/06/ontologies.html?page=2*), November 06, 2002.

[Desharnais et al. 02] Desharnais, J.-M.; Abran, A.; Mayer, A.; Buglione, L.; and Bévo, V., "Knowledge Modeling for the Design of a KBS in the Functional Size Measurement Domain", in *KES'02*, Italy, 2002.

[Desharnais 03] Desharnais, J.-M, *application de la mesure fonctionnelle cosmic-ffp : une approche cognitive* in Département d'informatique, Montréal Université Du Québec A Montréal, 2003, pp 201

[Desharnais 04] Desharnais, J.-M., *A application de la mesure fonctionnelle cosmic-ffp : une approche cognitive*, in Département d'informatique, Université du Québec à Montréal, Montréal, 2004.

[Diab et al. 01] Diab, H.; Frappier, M.; and St-Denis, R., "A Formal Definition of COSMIC-FFP for Automated Measurement of Room Specifications", in *FESMA*, 2001.

[Domingue et al. 99] Domingue, J.; Motta, E.; and Corcho Garcia, O., *Knowledge Modelling in WebOnto and OCML: A User Guide*, 1999. http://kmi.open.ac.uk/projects/ocml/.

[Domingue 98] Domingue, J., "Tadzebao and Webonto: Discussing, Browsing and Editing Ontologies on the Web", In *Proceedings of the Eleventh Knowledge Acquisition Workshop, KAW98*, Banff, 1998.

[Dorolle 49] Dorolle, M., *Le raisonnement par analogie*, Presses universitaires de France, Paris, 1949.

[Edge 00] Edge, N.J., *Automatic Calculation of an Ex-Post Unadjusted Function Point Approximation Count from COBOL Source Code*, Ph.D. Thesis, Bond University, queensland, Australia, 2000.

[Edge et al. 97] Edge, N.; Finnie, G.; and Wittig, G., "Automating Function Point Approximation Counts For COBOL Legacy", in *ACOSM 97*, Australian Software Metrics Association, Canberra National Convention Centre, ACS PCP, 1997, pp. 80-87.

[Ermine 96] Ermine, J.-L., *Les systèmes de connaissances*, Hermes, Paris, 1996.

[Farquhar et al. 97] Farquhar, A.; Fikes, R.; Pratt, W.; and Rice, J., "Tools for Assembling Modular Ontologies in Ontolingua", Technical Report KSL-97-03, Knowledge Systems Laboratory, Stanford University, 1997.

[Farquhar et al. 96] Farquhar; Fikes, R.; and Rice, J., "The Ontolingua Server: A Tool for Collaborative Ontology Construction", In *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada, 44.1-44.19, 1996, http://www.ksl.stanford.edu/software/ontolingua/].

[Farquhar et al. 95] Farquhar, A.; Fikes, R.; Pratt, W.; and Rice, J., "Collaborative Ontology Construction for Information Integration", Technical Report KSL 95-63, Knowledge Systems Laboratory, Stanford University, 1995.

[Fernandez et al. 02] Fernandez, José L.; and Sanchez Antonio Monzon, "Une extension d'UML pour les architectures à base de composants temps réel", magazine *Génie Logiciel (Le magazine de l'ingénierie du logiciel et des systèmes), no 60*, mars 2002.

[Fetcke 99] Fetcke, T., "A Generalized Structure for Function Point Analysis", in *IWSM'99*, Lac Supérieur, Canada, September 8-10, 1999.

[Fodor 75] Fodor, J., *The Language of Thought*, Harvard University Press, 1975.

[French 02] Franch, R., M., "The computational Modeling of Analogy-making", *Trend in Cognitive Sciences*, 6(5), 200-205, 2002.

[Fridman et al. 01] Fridman Noy, N.; and McGuinness, D., "Ontology Development 101: A Guide to Creating Your First Ontology", at (*http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html*), March, 2001.

[Frohn et al. 99] Frohn, J.; Himmer der, R.; Kandzia, P.; and Schlepphorst, C., "How to Write F-logic Programs in FLORID - A Tutorial for the Database Language F-logic", 1999, http://www.informatik.uni-freiburg.de/~dbis/florid/.

[Gangemi et al. 96] Gangemi, A.; Steve, G.; and Giacomelli, F., "ONIONS: An Ontological Methodology for Taxonomic Knowledge Integration", In *ECAI-96 Workshop on Ontological Engineering*, Budapest, August 13th, 1996.

[Gentner et al. 97] Gentner, D., and Markman, A., B., "Structure mapping in analogy and similarity", In *American Psychologist*, 52, 45-56, 1997. (To be reprinted in Mind readings: Introductory selections on cognitive scie nce, by P. Thagard, Ed., MIT Press)

[Gire 97] Gire, Alain, Méthodologie ouverte de la modélisation : Quelques réflexions épistémologiques, p. 55, 1997.

[Glaser 92] Glaser, Barney, G., Basics of Grounded Theory Analysis : Emergence vs. Forcing, (1992).

[Glaser et al. 67] Glaser, Barney, G.; and Strauss, Anselm, L. , *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine, p. 271, MEMO : The starting point of Grounded Theory Method. (1967).

[Gómez-Pérez et al. 02] Gómez-Pérez Asunción, Fernández-López Mariano, Corcho Oscar, *OntoWeb: Ontology-based information exchange for knowledge management and electronic commerce*, Deliverable 1.3: A survey on ontology tools, IST-2000-29243, 31 st May, 2002

[Gomez-Perez et al. 96] Gomez-Perez, A.; Fernandez, M.; and De Vicente, A., J., "Towards a Method to Conceptualize Domain Ontologies", In *ECAI-96 Workshop on Ontological Engineering*, Budapest, 1996.

[Gordon et al. 97] Gordon, M., B.; and Paugam-Moisy, H., *Sciences cognitives : Diversité des approches*, Hermes, Paris, 1997.

[Gramantieri et al. 97] Gramantieri, F.; Lamma, E.; Riguzzi, F.; and Mello, P., "A system for measuring function points from specifications", 1997.

[Gruber 93] Gruber T. R., "Toward Principles for the Design of Ontologies Used for Knowledge Sharing", Presented at the *International Workshop on Formal Ontology*, Padova, Italy. To appear in a collection edited by Nicola Guarino. Available as Technical Report KSL 93-04, Knowledge Systems Laboratory, Stanford University, March 1993. {robert.stevens carole seanb}@cs.man.ac.uk.

[Gruninger et al. 95] Gruninger, M.; and Fox, M., S., "Methodology for the Design and Evaluation of Ontologies", In *IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, August 19-20th 1995.

[Gruninger et al. 94a] Gruninger, M.; and Fox, M., S., "The Design and Evaluation of Ontologies for Enterprise Engineering", In *Workshop on Implemented Ontologies, European Conference on Artificial Intelligence*, Amsterdam, NL., 1994.

[Gruninger et al. 94b] Gruninger, M.; and Fox, M., S., "The Role of Competency Questions in Enterprise Engineering", In *IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, Trondheim, Norway, 1994.

[Guarino 98] Guarino, N., *Formal ontology in information systems*, IOS press, Amsterdam (NL), 1998

[Harnad, 03] Harnad, S., "The Symbol Grounding Problem", in *Encyclopedia of Cognitive Science*, Nature Publishing Group, Macmillan, 2003.

[Harold 01] Elliotte Rusty Harold, *The XML Bible - Gold Edition*, John Wiley & Sons 2001.

[Heflin et al., 1999] Heflin, J.; Hendler, J.; and Luke, S., "SHOE: A Knowledge Representation Language for Internet Applications", Technical Report, CS-TR-4078 (UMIACS TR-99-71), Dept. of Computer Science, University of Maryland at College Park, 1999.

[Hmelo 95] Hmelo, C. E., "Problem-based learning: Development of knowledge and reasoning strategies". In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, Hillsdale NJ: Erlbaum, 1995.

[Ho et al. 99] Ho, T., V.; and Abran, A., "A Framework for Automatic Function point Counting From Source Code", in *IWSM'99*, Lac Supérieur, Canada, p.248, Sept.ember 8-10, 1999.

[Holyoak et al. 95] Holyoak, K., J.; and Thagard, P., *Mental leaps: Analogy in creative thought*. Cambridge, MA: MIT Press/Bradford Books, 1995.

[Horrocks 02] Horrocks, I., "DAML+OIL: A reason-able web ontology language", in *Proceedings EDBT-02*, volume 2287 of LNCS, pages 2—13, Springer, 2002. , http://www.daml.org/.

[Humphreys et al. 93] Humphreys, B., L.; and Lindberg, D., A., B., "The UMLS project: making the conceptual connection between users and the information they need", *Bulletin of the Medical Library Association* 81(2): 170, 1993.

[IFPUG 01] IFPUG, *Function Point Analysis Counting Practices Manual, release 4.1* , Mequon, Wisconsin, International Function Point Users Group (IFPUG), 2001.

[Isakowitz, 02] Isakowitz, S. J., "NASA Cost Estimation Handbook", NASA HQ, Washington DC, spring 2002.

[ISO 01] ISO, ISO TC JTC1/SC SC7/WG 12, *Software Engineering — COSMIC-FFP Functional size measurement method*, Document type: International Standard, 2001.

[ISO/IEC 03a] ISO/IEC 19761:2003, Software engineering -- COSMIC-FFP -- A functional s*ize measurement method*, 2003.

[ISO/IEC 03b] ISO/IEC 20926:2003, Software engineering -- IFPUG 4.1 Unadjusted functional size measurement method Counting practices manual , 2003.

[ISO/IEC 02] ISO/IEC 20968:2002, Software engineering -- Mk II Function Poin*t Analysis -- Counting Practices Manual* , 2002.

[ISO/IEC 98] ISO/IEC, ISO/IEC 14143-1:1998 – Software Engineering - Software measurement - Functional size measurement– Part1 - Definition of concepts, 1998.

[ISO/IEC 97a] ISO/IEC, ISO/IEC 14143-1:1997 - I*nformation technology - Software measurement - Functional size measurement - Part 1 - Definition of concepts* , 1997.

[ISO/IEC 97b] ISO/IEC, ISO/IEC 14143-3:1997 – Software engineering – Software measurement – Functional size measurement – Part 3: Verifica*tion of functional size measurement methods*, 1997.

[ISO 93] ISO, Vocabulaire International des Termes Fondamentaux et Généraux de Métrologie, Deuxième édition, 1993, ISBN 92-67-01075-l.

[Jacquet 97] Jacquet, J.-P.; and Abran, A., "From Metrics to Software Measurement Methods : A Process Model" , *International Symposium on Software Engineering Standards (ISESS'97)*, Walnut Creek, CA, p. 128 – 135, (1997).

[Jones 96a] Jones , Carpers, *Applied Software Measurement: Assuring Productivity and Quality*, McGraw-Hill, New York, pp xvii,1,2,17,119 1996.

[Jones 96b] Jones, C. T., *Estimating Software Costs*, McGraw-Hill, New York, pp 9, 269, 276, 1996.

[Joshua et al. 89] Joshua, S.; and Dupin, J.-J., Représentations et modélisations : Le « débat scientifique » dans la classe et l'apprentissage de la physique, p. 30, Peter Lang, Berne, 1989.

[Karp et al. 99] Karp, P., D.; Chaudhri, V., K.; and Thomere, J., "Xol: An xml-based ontology exchange language", Technical report version 0.3, 1999.

[KBSI 94] KBSI, "The IDEF5 Ontology Description Capture Method Overview", KBSI Report, Texas, 1994.

[Kibler et al. 87] Kibler, D. and Aha, D., "Learning representative exemplars of concepts; An initial study", in *Proceedings of the fourth international workshop on Machine Learning*, UC-Irvine, pp 24-29, June 1987.

[Kolodner et al. 96] Kolodner, J., L.; Hmelo, C.,E., and Narayanan, N., H., "Problem-Based Learning Meets Case-Based Reasoning", In *Proceedings of the Second International Conference on the Learning Sciences*. Charlottesville, Va.: AACE Press, 188-95.1996.

[Kulik et al. 02a] Kulik, Peter J.; and Catherine Weber, "Les meilleures pratiques de la mesure du logiciel: Un bilan à fin 2001", *magazine Génie Logiciel (Le magazine de l'ingénierie du logiciel et des systèmes), no 61*, juin 2002.

[Kulik et al. 02b] Kulik, Peter J.; and Catherine Weber, " Software Metrics best practices - 2001", in *ASM 2002*, 14 Feb 2002.

[Kulik 00] Kulik, Peter J., "A Practical Approach to Software Metrics", *IEEE IT Professional*, Jan/Feb 2000.

[Lasila et al. 99] Lassila, Ora; and Swick, Ralph, R. , "Resource Description Framework (RDF) Model and Syntax Specification", World Wide Web Consortium Recommendation, 1999, **http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/** . (Latest version available at http://www.w3.org/TR/REC-rdf-syntax/).

[Lemoigne 90] Lemoigne, J.-L., *La modélisation des systèmes complexes*, p. 15,5, Dunod, 1990.

[Lethbridge et al. 01] Lethbridge, C., Timothy; and Laganière, Robert, *Practical software development using UML and Java*, 458p, The McGraw-Hill Companies, London, 2001.

[Lévesque 98] Lévesque Ghislain, Analyse de système orientée-objet et génie logiciel: Concept, méthodes et applications, 458p, Chenelière/McGraw-Hill , Montréal, 1998.

[Low 90] Low, G., C.; and Jefferey D., R., "Function Points in the Estimation and Evaluation of the Software Process", *IEEE Transaction on Software Engineering, Vol. 16, no 1,* Jan. 1990, 64-71.

[Luger 02] Luger,G. F., *Artificial Intelligence, Structures and Strategies for Complex Problem Solving*, Fourth Edition, Addison-Wesley, Fourth edition published 2002.

[Madhavan et al. 02] Madhavan, J.; Bernstein, P.; Domingos, P.; and Halevy, A., "Representing and Reasoning about Mappings Between Domain Models", In *Proceedings of the AAAI Eighteenth National Conference on Artificial Intelligence*, 2002.

[Mahesh 96] Mahesh, K., "Ontology Development for Machine Translation: Ideology and Methodology", Technical Report MCCS 96-292, Computing Research Laboratory, New Mexico State University, Las Cruces, NM, 1996.

[Mahesh et al. 95] Mahesh, K.; and Nirenburg, S. "A Situated Ontology for Practical NLP", In *IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, August 19-20th, 1995.

[McGuinness et al. 04] McGuinness, Deborah, L.; and Van Harmelen, Frank, "OWL Web Ontology Language Overview", W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/. (Latest version available at http://www.w3.org/TR/owl-features/.)

[Mendes 96] Mendes, O.; Abran, A.; Bourque, P., "An FP Tool Classification Framework and Market Survey", in *IFPUG Fall Conference*, Dallas, September 30 - October 4, 1996.

[Meunier 92] Meunier, J., G., "Le problème de la catégorisation dans la représentation des connaissances", *INTELLICA*, 1-2 p 353, 1992.

[Missikoff et al. 02] Missikoff, M.; Velardi, P.; and Navigli, R., "The Usable Ontology: An Environment for Building and Assessing a Domain Ontology", In *Proceedings of the International Semantic Web Conference 2002 (ISWC2002)*, Sardinia, Italy, June 2002.

[Mizoguchi 02] Mizoguchi, R., "Ontological Engineering: Ontology design and knowledge systematisation", in *CIRTA*, Montreal, Canada, August 2002.

[Mizoguchi et al. 00] Mizoguchi, R.; and Bourdeau, J., "Using Ontological Engineering to Overcome Common AI-ED problems", in *International Journal of Artificial Intelligence in Education*, 2000, 11, 107-121.

[Morris 98] Morris, P.; and Desharnais, J.-M., "Measuring ALL the Software not just what the Business Uses", *IFPUG*, 1998.

[Muller 00] Muller, P.-A., *Modélisation objet avec UML*, Éditions Eyrolles, 2000.

[Muysken, 03] Muysken, P., "Modelling language differences: The case of sign language", in *Ecole d'été en science cognitive*, Montreal, 2003.

[Naur et al. 69] Naur, P.; and Randel, B., "Software Engineering : A Report on a Conference sponsored by the NATO Science Committee", *NATO*, 1969.

[Newell et al. 76] Newell; and Simon, *ACM Turing Award Lecture*, 1976.

[Nishiyama 99] Nishiyama, Shigeru, "On Precision on Function Point Analysis", *Research and Development Center*, Nippon Telegraph and Telephone, Japan, 1999.

[Nishiyama et al. 94] Nishiyama, S.; and Furuyama, T., "The validity and applicability of function point analysis", *EOQ-SC'94*, Basel, Switzerland, 1994.

[Noy et al. 01] Noy, N., F.; Sintek, M.; Decker, S.; Crubezy, M.; Fergerson, R., W.; and Musen, M., A., "Creating Semantic Web Contents with Protege-2000", In *IEEE Intelligent Systems* 16(2):60-71, 2001.

[Noy et al. 01] Noy, Natalya, F.; and McGuinness, Deborah, L., "Ontology Development 101: A Guide to Creating Your First Ontology", *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05* and *Stanford Medical Informatics Technical Report SMI-2001-0880*, March 2001.

[Noy et al. 00] Noy, N., F.; Fergerson, R., W.; and Musen, M. A., "The knowledge model of Protege-2000: Combining interoperability and flexibility", In *12th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000)*, Juan-les-Pins, France, 2000.

[Oligny et al. 00] Oligny, S.; Abran, A., "COSMIC-FFP - Current Status and Outlook" in *Unisia-JECS Co. Ltd*, Japan, January, 2000, 19 p.

[OMG 03] O.M.G., *OMG Unified Modeling Language Specification*, The Object Management Group Inc., March 2003.

[Patel et al. 03] Patel; Supekar; and Lee, "OntoGenie: Extracting Ontology Instances From WWW", in *Proceedings of ISWC2003*, 2003.

[Paton 99] Paton, K., "Automatic Function Point Counting Using Static and Dynamic Code Analysis", in *International Workshop on Software Measurement (IWSM)*, Lac Supérieur, Québec, 1999, pp. 6.

[Poirier et al. 01] Poirier, P.; and Faucher, L., *Philosophie de l'esprit et des sciences cognitives, cahier de textes*, notes de cours, PHI-4315, Automne 2001.

[Porter et al. 86] Porter, B.; and Bareiss, R., "PROTOS: An experiment in knowledge acquisition for heuristic classification tasks", In *Proceedings of the First International Meeting on Advances in Learning (IMAL)*, Les Arcs, France, pp. 159-174, 1986.

[Price et al. 00] Price, C.; and Spackman, K.,A., "Snomed Clinical Terms", *British Journal of Healthcare Computing and Information Management* 17(2):27-31, 2000.

[Putnam et al. 92] Putnam, L., H.; and Myers, W., *Measures for excellence: Reliable software on time, within budget*, Yourdon Press, Prentice Hall Building, Englewood Cliffs, New Jersey 07632, p.61,64, 1992.

[Rask 91] Rask, R., *Algorithms for Counting Unadjusted Function points from Dataflow Diagrams*, ressearch report, University of Joensuu, Finland, 1991.

[Rector et al. 97] Rector, A., L.; Bechhofer, S., K.; Goble, C., A.; Horrocks, I.; Nowlan, W., A.; and Solomon, W., D., *The GRAIL Concept Modelling Language for Medical Terminology*, Artificial Intelligence in Medicine, Volume 9, 1997.

[Reifer 90] Reifer, D., J., *Assert-R : A Function Point sizing tool for scientific and real-time systems*, Journal of Systems Software, vol 11, pp. 159-171, 1990.

[Rialle et al. 94] Rialle, Vincent; and Payette, Daniel, *Modèles de la cognition : Vers une science de l'esprit*, p. 208, Lekton, Vol. IV – n° 2, Automne 1994.

[Rich et al. 91] Rich, E., Knight, K., *Artificial Intelligence*, 2nd edition, Mc Graw Hill, 1991.

[Rogers et al. 01] Rogers, J., E.; Roberts, A.; Solomon, W., D.; van der Haring, E; Wroe, C., J.; Zanstra, P., E.; Rector, A., L., "GALEN Ten Years On: Tasks and Supporting tools", in *Proceedings of MEDINFO2001*, 2001.

[Rudolph 89] Rudolph, E., E., "Precision of Function Point Counts", *IFPUG Spring Conference*, San Diego, CA, April 1989.

[Rumbaugh et al. 04] Rumbaugh, J.; Booch, G.; and Jacobson, I, *The Unified Modeling Language Reference Manual*, Addison-Wesley Object Technology Series, 2004

[Schreiber et al. 99] Schreiber, G.; Akkermans, H.; Anjewierden, A.; De Hoog, R.; Shadbolt, N.; Van de Velde, W.; and Wielinga, B., *Knowledge engineering and management: The CommonKADS methodology*, A Bradford Book, The MIT Press, Cambridge, Massachusetts, 1999.

[Schreiber et al. 95] Schreiber, A., TH.; Wielinga, B., J.; and Jansweijer, W., H., "The KACTUS View on the 'O' Word", In *IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, August 19-20th, 1995.

[Schank 82] Schank, R., Dynamic memory; a theory of reminding and learning in computers and people, Cambridge University Press, 1982.

[Sowa, 03] Sowa, J. F., "categorization in cognitive computer science", in *Ecole d'été en science cognitive*, Montreal, 2003.

[Sowa, 00] Sowa, J. F., "Ontology, metadata, and semiotic", presented at *ICCS'2000*, Darmstadt, Germany, August 14, 2000.

[Spackman et al. 00] Spackman, K., A.; and Price, C., "Converging towards a standard clinical terminology for health", in *Proceedings INFOCUS 2000*, Vancouver B.C., June 2000.

[Staab et al. 00] Staab, S.; and Maedche, A.,"Ontology Engineering beyond the Modeling of Concepts and Relations", 2000.

[Steve et al. 96] Steve, G.; and Gangemi, A., "ONIONS Methodology and the Ontological Commitment of Medical Ontology ON8.5", In *Proceedings of the 10 th Knowledge Acquisition Workshop - KAW'96*, Banff, Canada, November 9-14, 1996.

[Stevens et al. 00] Stevens, R.; Goble, C., A., and Bechhofer S., *Ontology-based Knowledge Representation for Bioinformatics*, in Department of Computer Science and School of Biological Sciences, University of Manchester, Oxford Road, Manchester, M13 9PL, September, 2000.

[Sure et al. 02] Sure, Y.; Erdmann, M.; Angele, J.; Staab, S.; Studer, R.; and Wenke, D., "OntoEdit: Collaborative Ontology Engineering for the Semantic Web", In *Proceedings of the International Semantic Web Conference 2002 (ISWC 2002)*, Sardinia, Italia, June 9-12 2002.

[Swartout et al. 97] Swartout, B.; Ramesh, P.; Knight, K.; and Russ, T., "Toward Distributed Use of Large-Scale Ontologies", In *Symposium on Ontological Engineering of AAAI*, Stanford, California, March, 1997.

[Symons et al. 99] Symons, C.; Abran, A.; Desharnais, J.-M.; Fagg, P.; Goodman, P.; Morris, P.; Oligny, S.; Onvlee, J.; Nevalainen, R.; Rule, G.; and St-Pierre, D., "COSMIC FFP - Buts, approche conceptuelle et progrès", in *ASSEMI*, Paris, France, September 30, 1999, 29 p.

[Symons 88] Symons, C.R., "Function Points Analysis : Difficulties and Improvements", *IEEE trans. On Soft. Eng., Vol. SE-14*, no. 1, Jan.,2-11 1988.

[Uemura et al. 01] Uemura, T.; Kusumoto, S.; and Inoue, K., "Function-point analysis using design specifications based on the Unified modelling Language", in *Journal of software maintenance and evolution : Research and practice*, June 2001.

[UKSMA 00] UKSMA Metrics Practices Committee, *MK II Function Point Analysis Counting Practices Manual., v.1.3.1*, UK, Septemb er 2000.

[Uschold et al. 96] Uschold, M.; and Gruninger, M., "Ontologies: Principles, Methods and Applications", *Knowledge Engineering Review*, 11(2), 93-137, 1996.

[Uschold et al. 95] Uschold, M.; and King, M., "Towards A Methodology for Building Ontologies", In *IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, Canada, 1995.

[Van den Berg 98] Van den Berg, Klaas “Software Quality in the Objectory Process”, Position Paper presented at the *Workshop on Automating the Object-Oriented Development Process*, ECOOP’98, Brussels, July 20, 1998.

[Van Heijst 97] Van Heijst, G.; Schreiber A., Th.; Wielinga, B., J., “Using explicit ontologies in KBS development”, in *Internatinal Journal of Human-Comupter Studies*, Vol 47, pp.183-292, 1997.

[Vignaux 92] Vignaux, Georges , *Les sciences cognitives : une introduction*, éditions La découverte, Paris, 1992.

[Vilus 03] Vilus, L., Extraction, dans une spécification produite á l’aide d’un outil case, d’instances de concepts uml pertinents pour la mesure de la taille fonctionnelle des logiciels suivant la méthode cosmic-ffp, in Département d’informatique, UQAM, Montréal, 2003

[Vincenti 90] Vincenti, Walter, G., *What Engineers Know and How They Know It : Analytical Studies from Aeronautical History* , p. 4, 12, 13, The Johns Hopkins University Press, Baltimore, (1990).

[Visser et al. 98] Visser, Pepijn; Jones, Dean; and Bench-Capon, Trevor, “Methodologies for ontologies development”, in *13th European Conference on Artificial Intelligence ECAI'98*, Brighton, England August 23-28, 1998.

[Whitmire 95] Whitmire, S., A., *An introduction to 3D Function Points*, pp. 43-53, Software Development, 1995.

[Wielinga et al. 94] Wielinga, B., J.; Schreiber, A., TH.; Jansweijer, W., H.; Anjewierden, A.; and Vam Harmelen, F., “Framework and Formalism for Expressing Ontologies”, KACTUS Project Deliverable DO1b.1, University of Amsterdam, 1994.

[Wielinga 92] Wielinga, “B.J.KADS: A modeling approach to knowledge engineering”, in *Journal of Knowledge Acquisition*, Vol.4, No.1, pp.5-53, 1992.

[Wilson et al. 99] Wilson, Robert, A.; and Keil, Frank, C. ,*The MIT Encyclopedia of the Cognitive Sciences, Psychology* , (1999).

[Wooldridge et al. 00] Wooldridge, M.; and Ciancarini, P., “Agent-Oriented Software Engineering : The state of the Art”, in *Agent Oriented software engineering : first international workshop; revised papers/AOSE 2000*, Limerrick, Ireland, June 10, 2000.

[Zeigler 76] Zeigler, B., P., *Theory of modeling and simulation*, J. Wiley, New York, 1976.

[Zuse 03] Zuse, Horst, “Ten Theses: What can practitioners learn from Measurement Theory?”, in *IWSM’03*, Montréal, Canada, September 23-25, 2003.

REVUES

[Communications of the ACM 99] Communications of the ACM, Vol. 42, No 10, October 1999

ANNEXES

ANNEXE A: TERMINOLOGIE

Pour une meilleure compréhension des problèmes théoriques inhérents à notre projet de recherche, nous nous proposons de passer en revue les principaux concepts que l'on retrouve dans le projet, ainsi que leur lien avec le projet.

MESURE[36]

Dans la littérature [Jacquet et al. 97], le terme «mesure » peut être utilisé pour faire référence à :

- Une **méthode** permettant d'assigner une valeur numérique ou symbolique à un objet dont on veut caractériser un attribut.
- l'**application d'une méthode**
- Le **résultat** de l'application d'une méthode
- Le **processus** qui va du design à l'exploitation d'une méthode de mesure

Dans notre projet de recherche nous abordons principalement la question d'***application d'une méthode de mesure*** et en partie la question de *design d'une méthode de mesure*; la mesure ici porte sur la notion de *taille* dans le contexte du logiciel.

TAILLE

Dans le contexte du projet, le concept de **taille** renvoie à la notion de *dimension*, de *grandeur* ou encore d'*ampleur* de quelque chose : Il en est un attribut. Il décrit une partie de notre connaissance empirique et objective de la chose considérée (Dans le cas d'espèce, la chose dont il est question c'est le *logiciel*). Nous nous intéressons plus spécifiquement à la *taille fonctionnelle* du logiciel, entendue comme la grandeur dudit logiciel en termes de *fonctionnalités* (du point de vue de l'utilisateur du logiciel).

FONCTIONNALITÉ

[36] « The sizing problem is difficult, but it is not impossible. » [38 Putnam et al. 92, p. 61]

Le sens du concept de **fonctionnalité** qui nous intéresse est celui de *possibilité offerte* par un système informatique (ce que le système permet aux utilisateurs de faire). Le logiciel, perçu comme système informatique abstrait, offre un certain nombre de « possibilités d'utilisation »/services aux utilisateurs (*Par exemple, un logiciel de traitement de texte permet à l'utilisateur d'editer du texte, d'imprimer du texte, de corriger l'orthographe des mots, etc.*). Ces « possibilités d'utilisation » (ou fonctionnalités) sont décrites de façon implicite ou explicite dans les cahiers de spécification à travers des FURs (Fonctionnalités Utilisateur Requises). C'est à ces «possibilités d'utilisation » que nous nous intéressons dans le projet.

CONNAISSANCE

Si l'on admet que la cognition c'est «l'acte de connaître », alors la **connaissance** apparaît comme « le résultat de cette action ». À l'origine, il y a l'environnement, qui regorge de données (ou faits), lesquelles données sont potentiellement porteuses d'informations et susceptibles d'être captées, enregistrées, transmises ou modifiées par un agent cognitif (naturel ou artificiel) de traitement de l'information. L'agent peut faire appel à des procédures pour affecter du sens aux données, des lois pour exprimer des relations entre les informations manipulées, des approches (ou des techniques) pour transformer les données ou les informations en d'autres données ou informations. Ces procédures d'affectation de sens à des données, ces lois exprimant des relations entre informations et ces modes de transformation de données (ou informations ou connaissances), constituent ce que l'on pourrait qualifier de « formes de connaissances ».

A propos de la « connaissance » en ingénierie, Walter G. Vincenti [Vincenti 90] estime que « *la connaissance créative, constructive d'un ingénieur est le savoir dont il a besoin pour mettre en œuvre son art* ». Il rejoint ainsi le philosophe Gilbert Ryle [Vincenti 90] qui lui, identifie deux sortes de «connaissances » : la première semblable à celle évoquée par Vincenti, qu'il appelle le «*savoir comment* » (« *Knowing how* ») et qui fait référence à la connaissance associée au savoir comment réaliser des tâches; la seconde c'est ce qu'il appelle le «*savoir que* » (« *Knowing that* ») et qui fait référence à la connaissance de faits.

Vu sous cet angle (le précédent), nous déduisons que chaque domaine d'activités, y compris le domaine de la mesure, est doté d'un ensemble de « connaissances » qui lui sont propres et qui y sont manipulées, aussi bien pour réaliser des tâches que pour générer de

nouvelles connaissances. Cela suppose un certain consensus autour de la façon de représenter les « connaissances » dans chaque domaine, ce qui n'est pas toujours sans poser de problèmes. Dans notre projet de recherche il est beaucoup question de *représentation* des connaissances, de *modélisation*, dans le domaine de la mesure en général, et la mesure de la taille fonctionnelle du logiciel en particulier.

REPRÉSENTATION

Winston (1984) définit une **représentation** comme un ensemble de conventions syntaxiques et sémantiques permettant de décrire des choses.

Pour Samuel Joshua et Jean-Jacques Dupin [Joshua et al. 89], une **représentation** désigne le *contenu structuré de la pensée d'un sujet* concernant un phénomène ou une classe de phénomènes.

Il existe plusieurs modes de représentation. Relativement à la théorie de Pierce, nous pouvons identifier :

- La représentation *indiciaire* (utilisant des signaux par relation causale. *Exemple : une trace d'animal*). Proche de ce mode, nous retrouvons entre autres la représentation *distribuée (utilisant des réseaux de neurones).*
- La représentation *iconique* (fondée sur la similarité. *Exemple : un dessin, une photo*).
- La représentation *symbolique (utilisant des symboles)*. Proche de ce mode, nous retrouvons entre autres la représentation *propositionnelle (utilisant des propositions logiques).*
- Des modes de représentation mixtes combinant certains modes sus cités

Dans le projet il sera question entre autres du mode de représentation symbolique et d'un mode de représentation alliant graphique et symboles. La question de *modélisation* occupera également une place importante.

MODÉLISATION[37]

37 « L'abeille confond par la structure de ses cellules de cire l'habileté de plus d'un architecte. Mais ce qui distingue dès l'abord le plus mauvais architecte de l'abeille la plus experte, c'est qu'il a construit la cellule dans sa tête avant de la construire dans la ruche. » [Karl Marx, Le capital, 1, p. 428]

De manière très simple, l'on pourrait définir la **modélisation** comme *l'élaboration, la conception des modèles*. Il apparaît dès lors que la définition de la modélisation est étroitement liée à la définition d'un modèle.

Pour Jean-Louis Lemoigne [Lemoigne 90], la **modélisation** désigne l'action d'*élaboration et de construction intentionnelle*, par composition de symboles, de modèles susceptibles de rendre intelligible un phénomène perçu complexe, et d'amplifier le raisonnement de l'acteur projetant une intervention délibérée au sein du phénomène, raisonnement visant notamment à anticiper les conséquences de ces projets d'actions possibles.

Dans l'un et l'autre cas la définition est étroitement liée au concept de *modèle*.

MODÈLE

D'après Jean-Louis Lemoigne [Lemoigne 90] un **modèle** est une *représentation artificielle* que « l'on construit dans sa tête » et que l'on « dessine » sur un support physique : le sable de la plage, la feuille de papier, l'écran d'un ordinateur,… Autrement dit, un système de symboles, un système artificiel (créé par l'homme) qui agence des symboles : des signes que l'on tient indissociables de leur capacité d'être à la fois signifiés (ils ont du sens pour qui les émet) et signifiants (ils ont du sens pour qui les reçoit).

Pour Alain Gire [Gire 97], un **modèle** se définit comme un objet transitionnel entre le réel et les systèmes de représentation, dont la nature hermétique impose une multi-valence permanente, même si tel ou tel axe est privilégié.

Quant à Richardson et Pugh (1981) [Britt 97], ils estiment que le terme « **modèle** » fait référence à une représentation d'un morceau de réalité, le but étant de gagner en compréhension, en exposant les hypothèses du modèle par rapport à un problème donné, à la critique, l'expérimentation et la re-formulation.

Dans le projet, il est entre autre question de *modèle statique* d'une méthode de mesure, de modèle d'une procédure de mesure.

MODÈLE STATIQUE D'UNE MÉTHODE DE MESURE

Nous entendons ici par **modèle statique** *[Bévo et al. 01]* d'une méthode de mesure, un modèle proposé par la méthode de mesure considérée et qui est utilisé par le « mesureur » pour bâtir le modèle du logiciel à mesurer, relativement à la méthode de mesure. Ce modèle

regroupe l'ensemble des concepts qui sont manipulés (processus fonctionnels, mouvements de données, groupes de données, …); et qui permettent de synthétiser les informations *pertinentes* (du point de vue de la mesure) relatives à un logiciel dont on veut déterminer la taille fonctionnelle. Ainsi, l'« instanciation » du modèle statique à partir de la documentation d'un logiciel, résulte en un modèle dudit logiciel (sa représentation, sa description) relativement à la méthode de mesure.

N.B. : *Le modèle statique d'une méthode correspond dans la littérature [Abran et al 99a, Abran et al. 01, Abran et al. 03, ISO/IEC 03a], à une version détaillée de ce qui est désigné sous le vocable « méta-modèle »/«modèle générique d'un logiciel»/«modèle contextuel» associé à la méthode.*

Une spécification assez claire du modèle statique d'une méthode de mesure, permettrait à notre avis d'avoir une meilleure compréhension de la méthode de mesure, en l'exposant à la critique, l'expérimentation et la re-formulation. Il constitue pour nous un excellent outil de travail par rapport à la démarche consensuelle pour laquelle nous avons opté.

En fait le modèle statique d'une méthode de mesure correspond à ce que d'aucun appelerait *l'ontologie de domaine associée au processus d'application de la méthode de mesure*.

ONTOLOGIE

Étymologiquement, le terme **ontologie** est utilisé pour désigner l'étude philosophique de ce qui existe. Aujourd'hui, il n'est plus uniquement utilisé dans les débats philosophiques, certains auteurs lui ayant donné au cours de l'histoire des connotations se rapprochant de leurs domaines d'étude respectifs.

Grüber (1993) [Grüber 93] par exemple, présente une **ontologie** comme une *spécification explicite d'une conceptualisation*.

Pour Wielinga & Schreiber (1993), il s'agit d'une *théorie des entités* qui peuvent exister dans l'esprit d'un agent intelligible.

Albert (1993) quant à lui la considère comme un *corpus de connaissances* relatives à une tâche particulière ou un domaine particulier (taxonomie de concepts pour la tâche ou le domaine, définissant l'interprétation sémantique de la connaissance). Ainsi, elle pourrait servir de plate-forme pour l'évolution vers un consensus dans un domaine considéré.

En IA de manière générale, le terme ontologie fait référence à un *ensemble de catégories, avec les liens entre ces catégories*. Il est bien souvent utilisé comme synonyme de terminologie dans un domaine spécifié (terminologie + interprétation sémantique des termes). Une ontologie peut être spécifique à des tâches ou à des domaines[38].

Dans la thèse nous combinons ces quatre perceptions, en mettant l'accent sur les deux dernières définitions, en vue de jeter les bases d'un corpus de connaissances relatives à l'application d'une méthode de mesure de la taille fonctionnelle des logiciels (Il est a noter que la définition de Gruber constitue pour nous l'épine dorsale de la combina ison). La façon de représenter ces connaissances constitue l'un des défis majeurs, et un pont avec la sémiotique.

SÉMIOTIQUE

Jean-Louis Ermine [Ermine 96] définit la **sémiotique** comme une sorte de « *généralisation de la linguistique,* où le langage est considéré comme une code particulier parmi d'autres. Pour elle (la sémiotique), tout est signe et susceptible d'une interprétation sémantique ». La linguistique selon lui, est «l'une des premières disciplines qui se pose profondément le problème du sens, de sa constitution et de sa structuration. Elle part de l'information brute qu'est le langage parlé ou écrit, et interprète son code pour en tirer un sens.

Dans le projet nous faisons appel à un ensemble de signes pour représenter les connaissances de la perspective quantitative du logiciel, et nous associons une interprétation sémantique à chacun des signes. Par exemple nous aurons besoin d'une façon adéquate de représenter les correspondances (« mappings ») entre concepts.

LE « MAPPING » - LA TRADUCTION

Le dictionnaire *Petit Larousse* définit la **traduction** comme l'action de traduire, de *transposer dans une autre langue*, de *faire passer (un texte par exemple) d'une langue dans*

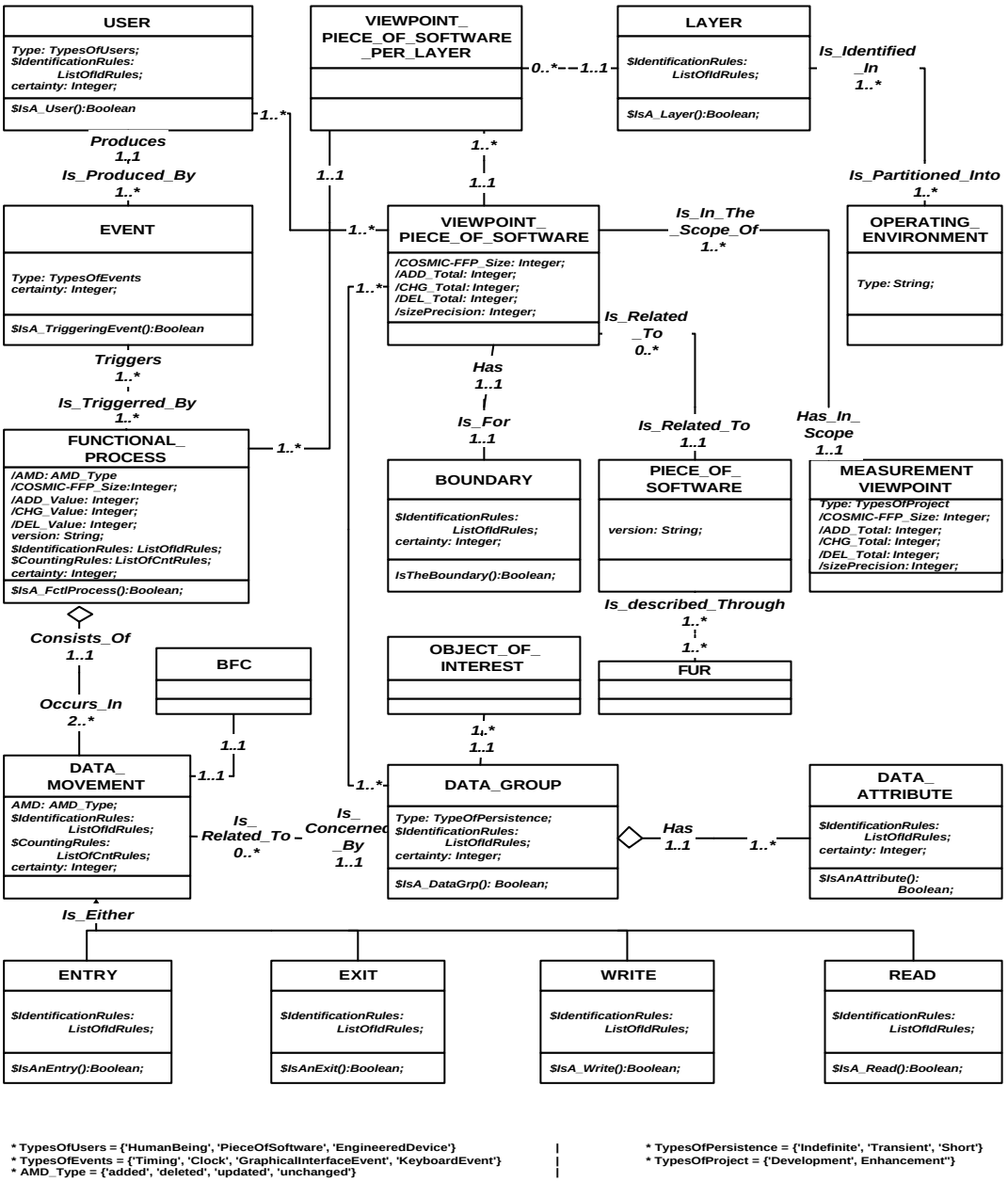
38 Terminological ontologies : Specifie les termes utilisés pour représenter la connaissance dans un domaine (exemple : UMLS = Unified Medical Language System; Lindberg et al., 1993, Page: 174
).
Information ontologies : Structures des enregistrements de bases de données (exemple : schéma d'une BD)
Knowledgemodeling ontologies : specify conceptualizations of the knowledge.

une autre. La traduction est une tâche on ne peut plus cognitive : Elle incombe au cogniticien, qui doit entre autres établir des correspondances (« **mappings** ») entre les concepts de deux ou plusieurs langages donnés. Cela suppose au préalable que les concepts impliqués dans les correspondances sont clairement définis.

Dans notre projet de recherche, il est question d'établir une correspondance entre une taxonomie de catégories représentant les concepts d'un langage de spécifications de logiciels (par exemple UML) et une taxonomie de catégories représentant les concepts d'une méthode de mesure de taille fonctionnelle des logiciels (COSMIC-FFP par exemple). *En d'autres termes, il s'agit d'une mise en correspondance entre deux ontologies de domaines.*

ANNEXE B : DESCRIPTION DES CONCEPTS ET TACHES COSMIC-FFP[39]

LES CONCEPTS



[39] A partir du Guide COSMIC d'implantation de la norme ISO 19761 (Cosmic Implementation Guide For ISO/IEC 19761: 2003)

OPERATING_ENVIRONMENT

Definition :

The software operating environment is the set of software operating concurrently on a specified computer system.

Properties :

- *Type : {String};*
- *Description : {String};*

LAYER

Definition :

A layer is the result of the functional partitioning of the software environment such that all included functional processes perform at the same level of abstraction.

Properties :

- *Name : {String};*
- *Description : {String};*
- *IdentificationRules : {IdRule}*

Concept : *IdRule*

Condition : {logical expression}

Action : {procedure}

PEACE_OF_SOFTWARE

Definition :

It's a set of computer instructions, data, procedures and maybe documentation, operating as a whole, to fulfil a specific set of purposes, all of which can be described from a functional perspective through a finite set of Functional User Requirements, Technical and Quality Requirements.

Properties :

- *Name : {String};*
- *Description : {String};*
- *Version : {String};*

FUNCTIONAL_PROCESS

Definition :

A functional process is an elementary component of a set of Functional User Requirements comprising a unique cohesive and independently executable set of data movements. It is triggered by one or more Triggering events either directly, or indirectly via an 'actor'. It is complete when it has executed all that is required to be done in response to the Triggering event (-type).

Properties :

- *AMD (AddedModifiedDeleted) : {AMD_Type}; /* AMD_Type = {'added', 'deleted', 'updated'} */*
- *Title : {String};*
- *Description : {String};*
- *Version : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *CountingRule[ ] : {CntRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a functional process. It will be used when calculating the precision associated with the functional size */*
- *ADD_Value : {Integer}; /* This value indicates the number of Functional Size Units related to adding of functionalities for this functional process (enhancement or development project) */*
- *CHG_Value : {Integer}; /* This value indicates the number of Functional Size Units related to modifications of functionalities for this functional process (enhancement project) */*
- *DEL_Value : {Integer}; /* This value indicates the number of Functional Size Units related to deletions of functionalities for this functional process (enhancement project) */*
- *COSMIC-FFP_Size : {Integer}; /* This value indicates the total number of Functional Size Units for this functional process */*

***Concept :** IdRule*

Condition : {logical expression}

Action : {procedure}

***Concept :** CntRule*

Condition : {logical expression}

Action : {procedure}

MEASUREMENT VIEWPOINT

Definition :

A measurement viewpoint is a viewpoint of the Functional User Requirements of software defined for the purposes of Functional Size Measurement..

Properties :

- *TypeOfProject : {TypeOfProject}; /* TypeOfProject = {'development', 'enhancement'} */*
- *Details : {String};*
- *sizePrecision : {Decimal}; /* This value indicates the precision associated with the total functional size for all pieces of software/parts of pieces of software within the scope of the measurement viewpoint */*
- *ADD_Value : {Integer}; /* This value indicates the number of Functional Size Units related to adding of functionalities for this measurement viewpoint (enhancement or development project) */*
- *CHG_Value : {Integer}; /* This value indicates the number of Functional Size Units related to modifications of functionalities for this measurement viewpoint (enhancement project) */*
- *DEL_Value : {Integer}; /* This value indicates the number of Functional Size Units related to deletions of functionalities for this measurement viewpoint (enhancement project) */*
- *COSMIC-FFP_Size : {Integer}; /* This value indicates the total number of Functional Size Units for all pieces of software/parts of pieces of software within the scope of the measurement viewpoint */*

BOUNDARY

Definition :

The boundary of a piece of software is the conceptual frontier between this piece and the environment in which it operates, as it is perceived externally from the perspective of its users. The boundary allows the measurer to distinguish, without ambiguity, what is included inside the measured software from what is part of the measured software's operating environment.

Properties :

- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*

- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a boundary. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

USER

Definition :

A User is any person that specifies Functional User Requirements and/or any person or thing that communicates or interacts with the software at any time.

NOTE: As mentioned in the COSMIC-FFP Measurement Manual, the term User is restricted to the second of the two meanings given in this definition, i.e. 'any person or thing that communicates or interacts with the software at any time'.

Properties :

- *Name : {String};*
- *Description : {String};*
- *Type : {TypeOfUser}; /* TypeOfUser = {'Human Being', 'Piece Of Software', 'Engineered Device'} */*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a user. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

EVENT (TRIGGERING EVENT)

Definition :

A triggering event is an event that occurs outside the boundary of the measured software and initiates one or more functional processes. In a set of Functional User Requirements, each event-type, which triggers a functional process, is indivisible for that set of FUR.

Properties :

- *Description : {String};*
- *Type : {TypeOfEvent}; /* TypeOfEvent = {'Timing', 'Clock', 'Graphical Interface Event', 'Keyboard Event'} */*
- *IdentificationRule[ ] : {IdRule}*

- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a triggering event. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

DATA MOVEMENT

Definition :

A data movement is a Base Functional Component, which moves one or more data attributes belonging to a single data group.

Properties :

- *AMD (AddedModifiedDeleted) : {AMD_Type}; /* AMD_Type = {'added', 'deleted', 'updated'} */*
- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *MeasurementRule[ ] : {CntRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a data movement. It will be used when calculating the precision associated with the functional size */*
- *COSMIC_Functional_Size_Unit: {Cfsu}; /* This value indicates the total number of Functional Size Units associated with this data movement */*
- *COSMIC-FFP_Size : {0 Cfsu, 1 Cfsu }; /* This value indicates the total number of Functional Size Units associated with this data movement */*

Concept : *IdRule /* see concept "Functional Process" */*

Concept : *CntRule /* see concept "Functional Process" */*

ENTRY (E)

Definition :

An Entry (E) is a data movement type that moves a data group from a user across the boundary into the functional process where it is required

Properties :

- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as an Entry. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

EXIT(X)

Definition :

An Exit (X) is a data movement type that moves a data group from a functional process across the boundary to the user that requires it.

Properties :

- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as an Exit. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

WRITE (W)

Definition :

A Write (W) is a data movement type that moves a data group lying inside a functional process to persistent storage.

Properties :

- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a Write. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

READ (R)

Definition :

A Read (R) is a data movement type that moves a data group from persistent storage within reach of the functional which requires it.

Properties :

- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a Read. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

DATA GROUP

Definition :

A data group is a distinct, non empty, non ordered and non redundant set of data attributes where each included data attribute describes a complementary aspect of the same object of interest (see definition).

Properties :

- *type : {TypeOfPersistence}; /* TypeOfPersistence = {'Indefinite', 'Transient', 'Short'} */*
- *Name : {String};*
- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a data group. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

DATA ATTRIBUTE

Definition :

A data attribute is the smallest parcel of information, within an identified data group, carrying a meaning from the perspective of the software's Functional User Requirements.

Properties :

- *Name : {String};*
- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a data group. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

BASE FUNCTIONAL COMPONENT (BFC)

Definition :

A Base Functional Component is an elementary unit of the Functional User Requirements defined by a FSM Method for measurement purposes (FSM: Functional Size Measurement).

Properties :

- *Name : {String};*
- *Description : {String};*

FUNCTIONAL USER REQUIREMENT (FUR)

Definition :

The term Functional User Requirements stands for a sub-set of the user requirements. The FUR represents the user practices and procedures that the software must perform to fulfil the user's needs. FUR excludes Quality Requirements and any Technical Requirements.

Properties :

- *Details : {String};*

OBJECT OF INTEREST

Definition :

An object of interest is identified from the point of view of the Functional User Requirements and may be any physical thing, as well as any conceptual objects or parts of conceptual objects in the world of the user about which the software is required to process and/or store data.

Properties :

- *Name : {String};*
- Description : {String};

LES ASSOCIATIONS ENTRE CONCEPTS

OPERATING_ ENVIRONMENT - LAYER

Description :

The partition of an operating environment into layers.

Arguments :

Operating_Environment

Roles : *Number of operating environments related to a given layer. [A given layer may be identified in many operating environments]*

Cardinality : *min 1; max n*;

User

Roles : *Number of layers into which a given operating environment can be partitioned. [A given operating environment can be partitioned into many layers]*

Cardinality : *min 1; max n*;

PIECE_OF_SOFTWARE - FUR

Description :

The FURs (Functional User Requirements) describing a piece of software.

Arguments :

Piece_Of_Software

Roles : *Number of pieces of software related to a given FUR. [A given FUR may be related to many versions of a piece of software]*

Cardinality : *min 1; max n*;

FUR

Roles : *Number of FURs describing a given piece of software. [A given version of a piece of software can be described through many FURs]*

Cardinality : *min 1; max n*;

DATA_GROUP - OBJECT_OF_INTEREST

Description :

The FURs (Functional User Requirements) describing a piece of software.

Arguments :

Data_Group

Roles : *Number of data groups related to a given object of interest. [A given object of interest is related to one data group]*

Cardinality : *min 1; max 1*;

Object_Of_Interest

Roles : *Number of objects of interest related to a given data group. [A given data group can be related to many objects of interest]*

Cardinality : *min 1; max n*;

DATA_MOVEMENT - BFC

Description :

In COSMIC-FFP method, the BFC (Base Functional Component is a data movement).

Arguments :

Data_Movement

Roles : *Number of data movements per BFC.*

Cardinality : *min 1; max 1*;

Object_Of_Interest

Roles : *Number of BFCs per data movement.*

Cardinality : *min 1; max 1*;

VIEWPOINT_PEACE_OF_SOFTWARE

Description :

Part of a piece of software within the scope of a measurement viewpoint.

Arguments :

Measurement_Viewpoint

Roles : *Number of measurement viewpoints related to one piece of software. [A piece of software can be partly or entirely in the scope of one or many measurement viewpoints]*

Cardinality : *min 0; max n*;

Piece_Of_Software

Roles : *Number of pieces of software within the scope of one measurement viewpoint. [A measurement viewpoint includes (has in scope) parts of some pieces of software]*

Cardinality : *min 1; max n*;

VIEWPOINT_PEACE_OF_SOFTWARE_PER_LAYER

Description :

Part of a viewpoint piece of software within a layer.

Arguments :

Viewpoint_Piece_Of_Software

Roles : *Number of measurement viewpoint pieces of software in which a given layer is identified. [A given layer may be identified in many viewpoint pieces of software]*

Cardinality : *min 0; max n*;

Layer

Roles : *Number of layers identified in a given measurement viewpoint pieces of software. [In a measurement viewpoint piece of software can be identified many layers]*

Cardinality : *min 1; max n*;

VIEWPOINT_PEACE_OF_SOFTWARE - BOUNDARY

Description :

The boundary of a viewpoint piece of software.

Arguments :

Viewpoint_Piece_Of_Software

Roles : *Number of viewpoint pieces of software related to a boundary. [A boundary is related to one viewpoint piece of software]*

Cardinality : *min 1; max 1*;

Boundary

Roles : *Number of boundaries for a given viewpoint piece of software. [A viewpoint piece of software has one boundary]*

Cardinality : *min 1; max 1*;

VIEWPOINT_PEACE_OF_SOFTWARE – DATA_GROUP

Description :

The data groups found in a viewpoint piece of software.

Arguments :

Viewpoint_Piece_Of_Software

Roles : *Number of viewpoint pieces of software related to a given data group. [A data group may be found in many viewpoint pieces of software]*

Cardinality : *min 1; max n*;

Data_Group

Roles : *Number of data groups found in a given viewpoint piece of software. [A viewpoint piece of software includes at least one data group]*

Cardinality : *min 1; max n*;

VIEWPOINT_PEACE_OF_SOFTWARE – USER

Description :

The users for a viewpoint piece of software.

Arguments :

Viewpoint_Piece_Of_Software

Roles : *Number of viewpoint pieces of software related to a given user. [A user may use many viewpoint pieces of software]*

Cardinality : *min 1; max n*;

User

Roles : *Number of users found in a given viewpoint piece of software. [A viewpoint piece of software is used at least by one user]*

Cardinality : *min 1; max n*;

VIEWPOINT_PEACE_OF_SOFTWARE_PER_LAYER – FUNCTIONAL_PROCESS

Description :

The functional processes identified in a viewpoint piece of software per layer.

Arguments :

Viewpoint_Piece_Of_Software_Per_Layer

Roles : *Number of viewpoint pieces of software per layer related to a given functional process. [A functional process is identified in one and only one viewpoint piece of software per layer]*

Cardinality : *min 1; max 1*;

User

Roles : *Number of functional processes identified in a given viewpoint piece of software per layer. [A viewpoint piece of software per layer may contain many functional processes]*

Cardinality : *min 1; max n*;

FUNCTIONAL_PROCESS - EVENT

Description :

The triggering events for a functional processes.

Arguments :

Functional_Process

Roles : *Number of functional processes related to a given event. [A given event may be related to many functional processes]*

Cardinality : *min 1; max n*;

Event

Roles : *Number of events triggering a given functional process. [A functional process is triggered by at least one event]*

Cardinality : *min 1; max n*;

FUNCTIONAL_PROCESS – DATA_MOVEMENT

Description :

The data movements for a functional processes.

Arguments :

Functional_Process

Roles : *Number of functional processes related to a given data movement. [A given data movement is related to one functional process]*

Cardinality : *min 1; max 1*;

Data_Movement

Roles : *Number of data movements for a given functional process. [A functional process has at least two (2) data movement]*

Cardinality : *min 2; max n*;

DATA_MOVEMENT – ENTRY – EXIT – READ - WRITE

Description :

The categories of data movements in COSMIC-FFP.

Arguments :

Entry, Exit, Read, Write

Roles : *A data movement in COSMIC-FFP is either an entry, an exit, a read or a write.*

Cardinality : *min 1; max 1*;

Data_Movement

Roles : *An entry, an exit, a read or a write is a data movement in COSMIC-FFP.*

Cardinality : *min 1; max 1*;

DATA_MOVEMENT – DATA_GROUP

Description :

The data groups related to a data movement.

Arguments :

Data_Group

Roles : *Number of data groups related to a given data movement. A data movement in COSMIC-FFP is relative to one data group.*

Cardinality : *min 1; max 1*;

Data_Movement

Roles : *Number of data movements related to a given data group. [A given data group may be subject to many data movements in COSMIC-FFP]*

Cardinality : *min 0; max n*;

DATA_GROUP – DATA_ATTRIBUTE

Description :

The data attributes of a data group.

Arguments :

Data_Group

Roles : *Number of data groups related to a given data attribute. [A given data attribute is related to one data group]*

Cardinality : *min 1; max 1*;

Data_Attribute

Roles : *Number of data attributes for a given data group. [A given data group has at least one data attribute]*

Cardinality : *min 0; max n*;

USER – EVENT

Description :

The events produced by a user.

Arguments :

Event

Roles : *Number of events related to a given user. [A given user may be related to many events]*

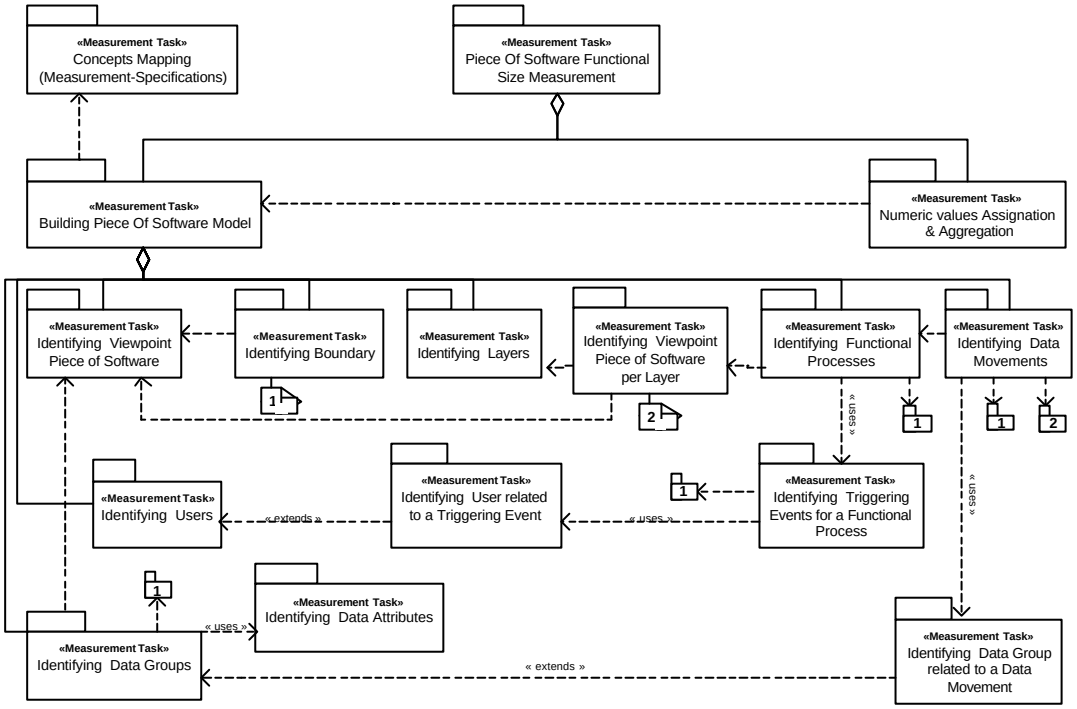
Cardinality : *min 1; max n*;

User

Roles : *Number of users related to a given event. [A given event is related to one user]*

Cardinality : *min 0; max 1*;

LES TACHES



PIECE OF SOFTWARE FUNCTIONAL SIZE MEASUREMENT

Definition :

Goal : *Given a set of specifications for a piece of software and a measurement viewpoint, the task should produce a COSMIC-FFP model for the piece of software and calculate its functional size in Cfsu (Cosmic Functional Size Unit). It's a composition of two (2) complementary tasks : « Building Piece Of Software Model » and « Numeric Values Assignment/Association & Aggregation ».*

Inputs : *Set of specifications for a piece of software, Measurement Viewpoint, Operating Environment, Set of mapping results*

Outputs : *COSMIC-FFP model for the piece of software, functional size of the piece of software, [global precision/certainty of the measurement]*

Body :

Type : *composition*

Sub-Tasks : *« Building Piece Of Software Model », « Numeric Values Assignment/Association & Aggregation »*

Control loop : *execute sequentially :*

- *Building_Piece_Of_Software_Model (+Set_of_specifications_for_a_piece_of_software, +Measurement_Viewpoint, +[Operating_Environment], +Set_Of_Mapping_Results, -COSMIC-FFP_model_for_the_piece_of_software),*
- *Numeric_Values_Assignment/Association_&_Aggregation (+COSMIC-FFP_model_for_the_piece_of_ software, -functional_size_of_the_piece_of_software, -global_precision/certainty_of_the_measurement)*

BUILDING PIECE OF SOFTWARE MODEL

Definition :

Goal : *Given a set of specifications for a piece of software, a measurement viewpoint and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should produce a COSMIC-FFP model for the piece of software. It's a composition of eight*

(8) complementary tasks : « Identifying Layers », « Identifying Viewpoint Piece Of Software », « Identifying boundary », « Identifying Viewpoint Piece Of Software Per Layer », « Identifying users », « Identifying Data Groups », « Identifying Functional Processes » and « Identifying data movements », It uses the results of the task « Concepts Mapping ».

Inputs : *Set of specifications for a piece of software, Measurement Viewpoint, Operating Environment, Set of mapping results*

Outputs : *COSMIC-FFP model for the piece of software*

Body :

Type : *composition*

Sub-Tasks : *« Identifying Layers », « Identifying Viewpoint Piece Of Software », « Identifying boundary », « Identifying Viewpoint Piece Of Software Per Layer », « Identifying users », « Identifying Data Groups », « Identifying Functional Processes » and « Identifying data movements »*

Control loop : *execute sequentially :*

- *Identifying_Layers (+Operating_Environment, -Layer_In_Operating_Environment[ ])*
- *Identifying_Viewpoint_Piece_Of_Software*[40] *(+Set_Of_Specifications_For_a_Piece_Of_Software, +Measurement_Viewpoint, -Viewpoint_Piece_Of_Software)*
- *Identifying_Boundary (+Viewpoint_Piece_Of_Software, -Boundary)*
- *Identifying_Viewpoint_Piece_Of_Software_Per_Layer (+Viewpoint_Piece_Of_Software, +Layer_In_Operating_Environment[ ], -Viewpoint_Piece_Of_Software_Per_Layer[ ])*
- *Identifying_Users (+View_Point_Piece_Of_Software, +Set_Of_Mapping_Results, -Users[ ])*
- *Identifying_Data_Groups (+View_Point_Piece_Of_Software, +Set_Of_Mapping_Results, -Data_Group[ ])*
- *Identifying_Functional_Processes (+View_Point_Piece_Of_Software_Per_Layer[ ], +Set_Of_Mapping_Results, -Functional_Processes[ ])*

[40] */* See the definition of the concept « View_Point_Piece_Of_Software » */*

- *Identifying_Data_Movements (+View_Point_Piece_Of_Software_Per_Layer, +Boundary, +Functional_Processes[ ], +Set_Of_Mapping_Results, -Data_Movement[ ])*

NUMERIC VALUES ASSIGNMENT/ASSOCIATION & AGGREGATION

Definition :

Goal : *Given a COSMIC-FFP model for a given piece of software, the task should assign numeric values to some elements of the model (data movements to be precise), then aggregate these values to derive the functional size in Cfsu (Cosmic Functional Size Unit) of the considered piece of software according to the given viewpoint. The aggregation should take into account the uncertainty parameter associated with some elements of the model (functional processes, data groups, data movements, ...), to derive the global precision of the measurement. We are currently working on this issue.*

Inputs : *COSMIC-FFP model for the piece of software*

Outputs : *Functional size of the piece of software, [global precision/certainty of the measurement]*

Body :

Type : *composition*

Sub-Tasks : *« Numeric values Assignment/Association », « Aggregation »*

Control loop : *execute sequentially :*

- *Numeric_Values_Assignment/Association (+COSMIC-FFP_model_for_the_piece_of_software, -COSMIC-FFP_modelPlus_for_the_piece_of_software)*
- *Aggregation (+COSMIC-FFP_modelPlus_for_the_piece_of_software, -Functional_size_of_the_piece_of_software, -Global_precision/certainty_of_the_measurement)*
-

CONCEPTS MAPPING

Definition :

Goal : *Map measurement concepts (COSMIC-FFP concepts) with some concepts of the specification language (for example UML).*

Inputs : *COSMIC-FFP concepts, Specifications language concepts*

Outputs : *mapping results*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Concepts_Mapping (+COSMIC-FFP_Concept[ ], Specifications_Language_Concept[ ], -Mapping_Result[ ])*

IDENTIFYING LAYERS

Definition :

Goal : *Given a software operating environment, the task should partition functionally the operating environment into layers.*

Inputs : *A software operating environment*

Outputs : *List of layers related to the operating environment*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Layers (+Operating_Environment, -Layer_In_Operating_Environment[ ])*

IDENTIFYING VIEWPOINT PIECE OF SOFTWARE

Definition :

Goal : *Given a set of specifications for a piece of software and a measurement viewpoint, the task should determine the subset of specifications related to parts of the piece of software which are in the scope of the measurement viewpoint.*

Inputs : *Set of specifications for a piece of software, Measurement Viewpoint*

Outputs : *a viewpoint piece of software*[41]

[41] */* See the definition of the concept « View_Point_Piece_Of_Software » */*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Viewpoint_Piece_Of_Software (+Set_Of_Specifications_For_a_Piece_Of_Software, +Measurement_Viewpoint, -Viewpoint_Piece_Of_Software)*

IDENTIFYING BOUNDARY

Definition :

Goal : *Given a viewpoint piece of software (parts of a piece of software which are in the scope of a measurement viewpoint), the task should determine the associated boundary.*

Inputs : *a viewpoint piece of software*[42]

Outputs : *a boundary related to the viewpoint piece of software*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Boundary (+Viewpoint_Piece_Of_Software, -Boundary)*

IDENTIFYING VIEWPOINT PIECE OF SOFTWARE PER LAYER

Definition :

Goal : *Given a viewpoint piece of software (parts of a piece of software which are in the scope of a measurement viewpoint) and a list of layers related to a software environment, the task should determine for each layer, the corresponding part of the viewpoint piece of software. It may happen that a layer is not at all related to any part of a viewpoint piece of software.*

Inputs : *a viewpoint piece of software*[42]*, a list of layers related to a software environment*

Outputs : *parts of a viewpoint piece of software (zero or one per layer)*

Body :

[42] */* See the definition of the concept « View_Point_Piece_Of_Software » */*

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Viewpoint_Piece_Of_Software_Per_Layer (+Viewpoint_Piece_Of_Software, +Layer_In_Operating_Environment[ ], -Viewpoint_Piece_Of_Software_Per_Layer[ ])*

IDENTIFYING USERS

Definition :

Goal : *Given a viewpoint piece of software (parts of a piece of software which are in the scope of a measurement viewpoint) and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine the users of those parts of the piece of software ('any person or thing that communicates or interacts with those parts of the piece of software at any time').*

Inputs : *A viewpoint piece of software, a set of mapping results*

Outputs : *users for the viewpoint piece of software*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Users (+View_Point_Piece_Of_Software, +Set_Of_Mapping_Results, -Users[ ])*

IDENTIFYING DATA GROUPS

Definition :

Goal : *Given a viewpoint piece of software (parts of a piece of software which are in the scope of a measurement viewpoint) and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine the data groups found in those parts of the piece of software. The data attributes of each data group should be pointed out.*

Inputs : *A viewpoint piece of software, a set of mapping results*

Outputs : *data groups found in the viewpoint piece of software*

Body :

Type : *simple*

Used Tasks : *«Identifying Data Attributes»*

Control loop : *execute :*

- *Identifying_Data_Groups (+View_Point_Piece_Of_Software, +Set_Of_Mapping_Results, -Data_Group[ ])*

IDENTIFYING FUNCTIONAL PROCESSES

Definition :

Goal : *Given a list of viewpoint piece of software per layer (part of a viewpoint piece of software related to a layer) and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine the functional processes found in each viewpoint piece of software per layer. The triggering event of each functional process should be pointed out.*

Inputs : *A list of viewpoint piece of software per layer, a set of mapping results*

Outputs : *functional processes found in each viewpoint piece of software per layer*

Body :

Type : *simple*

Used Tasks : *«Identifying triggering event for a Functional Process»*

Control loop : *execute :*

- *Identifying_Functional_Processes (+View_Point_Piece_Of_Software_Per_Layer[ ], +Set_Of_Mapping_Results, -Functional_Processes[ ])*

IDENTIFYING DATA MOVEMENTS

Definition :

Goal : *Given a viewpoint piece of software per layer (part of a viewpoint piece of related to a layer), the boundary related to the viewpoint piece of software, a list of functional processes and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine the data movements found in each*

functional process. The data group related to each data movement should be pointed out.

Inputs : *A viewpoint piece of software per layer, boundary related to the viewpoint piece of software, list of functional processes, a set of mapping results*

Outputs : *data movements found in each functional process*

Body :

Type : *simple*

Used Tasks : *«Identifying data group related to a data movement»*

Control loop : *execute :*

- *Identifying_Data_Movements (+View_Point_Piece_Of_Software_Per_Layer, +Boundary, +Functional_Processes[ ], +Set_Of_Mapping_Results, -Data_Movement[ ])*

IDENTIFYING DATA ATTRIBUTES

Definition :

Goal : *Identify all data attributes related to a data group.*

Inputs : *A viewpoint piece of software a set of mapping results, a data group*

Outputs : *List of data attributes related to the given data group*

Body :

Type : *simple*

Control loop : *execute :*

- *Identifying_Data_Atributes (+View_Point_Piece_Of_Software, +Set_Of_Mapping_Results, +Data_Group, -Data_Attribute[ ])*

IDENTIFYING TRIGGERING EVENTS

Definition :

Goal : *Identify all the events that trigger a given functional process.*

Inputs : *A viewpoint piece of software per layer, boundary related to the viewpoint piece of software, a functional process, a set of mapping results*

Outputs : *list of events triggering the given functional process*

Body :

Type : *simple*

Used Tasks : *«Identifying user related to a triggering event»*;

Control loop : *execute :*

- *Identifying_Triggering_Events (+View_Point_Piece_Of_Software_Per_Layer, +Set_Of_Mapping_Results, +Boundary, +Functional_Process, -Triggering_Event[ ])*

IDENTIFYING DATA GROUP RELATED TO A DATA MOVEMENT

Definition :

Goal : *Identify the data group related to a given data movement.*

Inputs : *A viewpoint piece of software per layer, boundary related to the viewpoint piece of software, a data movement, a set of mapping results*

Outputs : *a data group related to the given data movement*

Body :

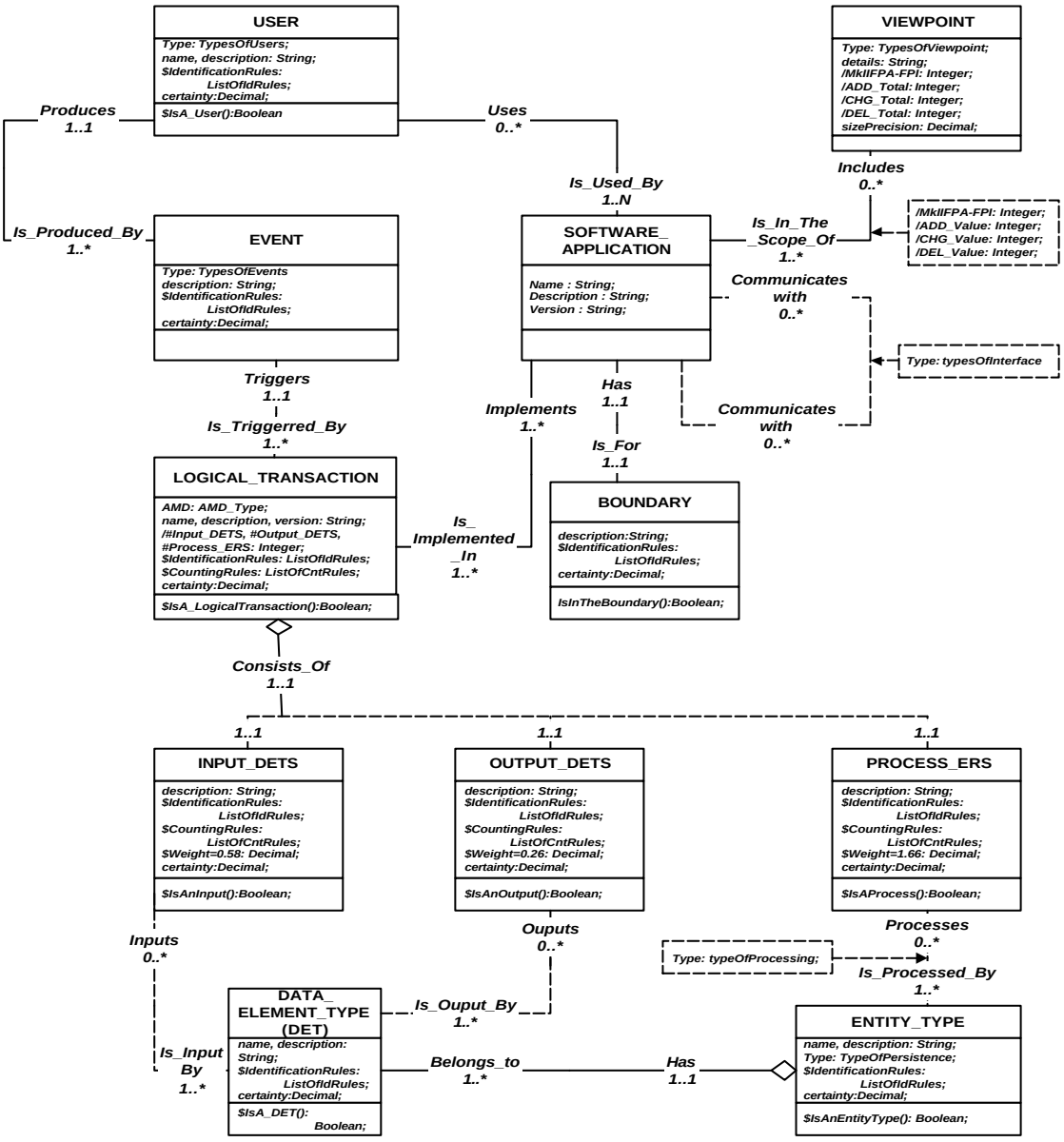
Type : *simple*

Control loop : *execute :*

- *Identifying_Data_Group_Related_To_A_Data_Movement (+View_Point_Piece_Of_Software_Per_Layer, +Boundary, +Set_Of_Mapping_Results, +Data_Movement, -Data_Group)*

ANNEXE C : DESCRIPTION DE CONCEP T ET TACHES MKII-FPA

LES CONCEPTS



*** TypesOfUsers = {'HumanBeing', 'PieceOfSoftware', 'EngineeredDevice'}**
*** TypesOfEvents = {'Timing', 'Clock', 'GraphicalInterfaceEvent', 'KeyboardEvent'}**
*** TypesOfViewpoint = {'Project Viewpoint', 'Application Manager Viewpoint', 'Business Enterprise Viewpoint'}**
*** AMD_Type = {'Added', 'Deleted', 'Updated'}**
*** TypesOfProcessing = {'Create', 'Delete', 'Update', 'Read'}**

MEASUREMENT VIEWPOINT

Definition :

A measurement viewpoint is a viewpoint of the Functional User Requirements of software defined for the purposes of Functional Size Measurement.

Properties :

- *Type : {TypeOfViewpoint}; /* TypeOfViewpoint = {'Project Viewpoint', 'Application Manager Viewpoint', 'Business Enterprise Viewpoint'} */*
- *Details : {String};*
- *sizePrecision : {Decimal}; /* This value indicates the precision associated with the total functional size for all pieces of software/parts of pieces of software within the scope of the measurement viewpoint */*
- *ADD_Total : {Integer}; /* This value indicates the number of Functional Size Units related to adding of functionalities for this measurement viewpoint (enhancement or development project) */*
- *CHG_Total : {Integer}; /* This value indicates the number of Functional Size Units related to modifications of functionalities for this measurement viewpoint (enhancement project) */*
- *DEL_Total : {Integer}; /* This value indicates the number of Functional Size Units related to deletions of functionalities for this measurement viewpoint (enhancement project) */*
- *MkII-FPA_Size : {Integer}; /* This value indicates the total number of Functional Size Units for all pieces of software/parts of pieces of software within the scope of the measurement viewpoint */*

SOFTWARE_APPLICATION

Definition :

It's a coherent collection of automated procedures and data supporting a business objective. Frequently a synonym for 'system', the word 'application' is preferred in MkII-FPA manual since it expresses more precisely the nature of the subject matter of Functional Size Measurement.

Properties :

- *Name : {String};*
- *Description : {String};*
- *Version : {String};*

BOUNDARY

Definition:

A Boundary refers to the conceptual interface between a software application under study and its users. The boundary determines what logical transactions are included in the function point count, and what are excluded. The application or part of the application enclosed by the boundary must be a coherent body of functionality, comprising one or more complete Logical Transaction Types.

Properties:

- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty: {Decimal}; /* This value indicates how certain the measurer is in identifying this as a boundary. It will be used when calculating the precision associated with the functional size */*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

LOGICAL TRANSACTION

Definition:

A Logical Transaction is the basic functional component of Mk II FPA. It is the smallest complete unit of information processing that is meaningful to the end user in the business. It is triggered by an event in the real world of interest to the user, or by a request for information. It comprises an input, process and output component. It must be self-contained and leave the application being counted in a consistent state.

Properties:

- *AMD (AddedModifiedDeleted) : {AMD_Type}; /* AMD_Type = {'added', 'deleted', 'updated'} */*
- *name : {String};*
- *Description: {String};*
- *Version : {String};*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a logical transaction.. It will be used when calculating the precision associated with the functional size */*

- *#Input_DETS : {Integer}; /* This value indicates the total number of Data Element Types input for this logical transaction */*
- *#Output_DETS : {Integer}; /* This value indicates the total number of Data Element Types output for this logical transaction */*
- *#Process_ERS : {Integer}; /* This value indicates the total number of Data Entities Types Referenced/Processed in this logical transaction */*
- *IdentificationRule[]: {IdRule}*
- *MeasurementRule[]: {CntRule}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

INPUT_DETS (INPUT OF DATA ELEMENT TYPES)

Definition :

It's an input of one ore more Data Element Types across an application boundary.

Properties :

- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *MeasurementRule[ ] : {CntRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as an Input_DETS. It will be used when calculating the precision associated with the functional size */*
- *Weight=0.58:{Decimal}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

OUTPUT_DETS (OUTPUT OF DATA ELEMENT TYPES)

Definition :

It's an output of one ore more Data Element Types across an application boundary.

Properties :

- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *MeasurementRule[ ] : {CntRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as an Output_DETS. It will be used when calculating the precision associated with the functional size */*
- *Weight=0.28:{Decimal}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

PROCESS_ERS (PROCESSING OF DATA ENTITY TYPES)

Definition :

It's a processing involving one ore more Data Entity Types within an application boundary.

Properties :

- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *MeasurementRule[ ] : {CntRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as an Output_DETS. It will be used when calculating the precision associated with the functional size */*
- *Weight=1.88:{Decimal}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

USER

Definition :

(ISO Definition) It's any person that specifies Functional User Requirements and/or any person or thing that communicates or interacts with the software at any time.

Properties :

- *Name : {String};*
- *Description : {String};*
- *Type : {TypeOfUser}; /* TypeOfUser = {'Human Being', 'Piece Of Software', 'Engineered Device'} */*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a user. It will be used when calculating the precision associated with the functional size */*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

EVENT (TRIGGERING EVENT)

Definition :

A triggering event is an event that occurs outside the boundary of the measured software and initiates one or more logical transactions.

Properties :

- *Description : {String};*
- *Type : {TypeOfEvent}; /* TypeOfEvent = {'Timing', 'Clock', 'Graphical Interface Event', 'Keyboard Event'} */*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a triggering event. It will be used when calculating the precision associated with the functional size */*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

DATA ENTITY TYPE OR ENTITY

Definition :

A Data Entity Type is a fundamental thing of relevance to the user, about which information is kept. An association between entities that has attributes is itself an entity.

Properties :

- *type : {TypeOfPersistence}; /* TypeOfPersistence = {'Indefinite', 'Transient', 'Short'} */*
- *Name : {String};*
- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a Data Entity Type. It will be used when calculating the precision associated with the functional size */*

Concept : *IdRule /* see concept "Functional Process" */*

DATA ELEMENT TYPE

Definition :

A Data Element Type is a uniquely processed item of information that is indivisible for the purposes of the Logical Transaction being sized. It is part of an input or output data flow across an application boundary. It's a unique user recognisable, non-recursive item of information. The number of DET's is used to determine the input and output size of each Logical Transaction.

Properties :

- *Name : {String};*
- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a data group. It will be used when calculating the precision associated with the functional size */*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

LES ASSOCIATIONS ENTRE CONCEPTS

VIEWPOINT – SOFTWARE_APPLICATION

Description :

Software applications within the scope of a measurement viewpoint.

Arguments :

Viewpoint

Roles : *Number of viewpoints related to a given software application. [A given software application can be in the scope of many measurement viewpoints]*

Cardinality : *min 0; max n*;

Software_Application

Roles : *Number of software applications in the scope of a given measurement viewpoint. [A given measurement viewpoint may include many software applications]*

Cardinality : *min 1; max n*;

SOFTWARE_APPLICATION - BOUNDARY

Description :

The boundary of count for a given software application.

Arguments :

Software_Application

Roles : *Number of software applications related to a boundary of count. [A boundary of count is related to one software application]*

Cardinality : *min 1; max 1*;

Boundary

Roles : *Number of boundaries for a given software application. [A software applications has one boundary]*

Cardinality : *min 1; max 1*;

SOFTWARE_APPLICATION - USER

Description :

The users for a software application.

Arguments :

Software_Application

Roles : *Number of software applications used by a given user. [A user may use many software applications]*

Cardinality : *min 1; max n*;

User

Roles : *Number of users using a given software application. [A software application is used at least by one user]*

Cardinality : *min 1; max n*;

SOFTWARE_APPLICATION – LOGICAL_TRANSACTION

Description :

Logical transactions implemented in a given software application.

Arguments :

Software_Application

Roles : *Number of software applications implementing a given logical transaction. [A logical transaction is implemented in at least one version of a software application]*

Cardinality : *min 1; max n*;

User

Roles : *Number of logical transactions implemented in a given software application. [A software application may implement many logical transactions]*

Cardinality : *min 1; max n*;

USER – EVENT

Description :

The events produced by a user.

Arguments :

Event

Roles : *Number of events related to a given user. [A given user may be related to many events]*

Cardinality : *min 1; max n*;

User

Roles : *Number of users related to a given event. [A given event is related to one user]*

Cardinality : *min 0; max 1*;

LOGICAL_TRANSACTION - EVENT

Description :

The triggering events for a logical transaction.

Arguments :

Logical_Transaction

Roles : *Number of logical transactions triggered by a given event. [A given event may trigger many logical transactions]*

Cardinality : *min 1; max n*;

Event

Roles : *Number of events triggering a given logical transaction. [A logical transaction is triggered by one event]*

Cardinality : *min 1; max 1*;

LOGICAL_TRANSACTION – INPUT_DETS – OUTPUT_DETS – PROCESS_ERS

Description :

The components of a logical transaction.

Arguments :

Logical_transaction

Roles : *Number of logical transactions related to a given combination of Input_DETS (a set of Data Element Types input), output_DETS (a set of Data Element Types output), and Process_ERS (a set of Entity Types processed). [A given combination of one Input_DETS, one output_DETS, and one Process_ERS, is related to one logical transaction]*

Cardinality : *min 1; max 1*;

Input_DETS

Roles : *Number of Input_DETS (set of Data Element Types input) for a given logical transaction. [A logical transaction inputs one (1) set of Data Element Types]*

Cardinality : *min 1; max 1*;

Output_DETS

Roles : *Number of Output_DETS (set of Data Element Types output) for a given logical transaction. [A logical transaction outputs one (1) set of Data Element Types]*

Cardinality : *min 1; max 1*;

Process_ERS

Roles : *Number of Process_ERS (set of Entity Types processed) for a given logical transaction. [A logical transaction processes one (1) set of Entity Types]*

Cardinality : *min 1; max 1*;

INPUT_DETS – DATA_ELEMENT_TYPE (DET)

Description :

The Data Element Types related to a given Input_DETS.

Arguments :

Input_DETS

Roles : *Number of Input_DETS related to a given Data Element Type. [A Data Element Type may be related to many Input_DETS]*

Cardinality : *min 0; max n*;

Data_Element_Type

Roles : *Number of Data element types related to a given Input_DETS. [An Input_DETS includes at least one Data Element Type]*

Cardinality : *min 1; max n*;

OUTPUT_DETS – DATA_ELEMENT_TYPE (DET)

Description :

The Data Element Types related to a given Output_DETS.

Arguments :

Output_DETS

Roles : *Number of Output_DETS related to a given Data Element Type. [A Data Element Type may be related to many Output_DETS]*

Cardinality : *min 0; max n*;

Data_Element_Type

Roles : *Number of Data element types related to a given Output_DETS. [An Output_DETS includes at least one Data Element Type]*

Cardinality : *min 1; max n*;

PROCESS_ERS – ENTITY_TYPE

Description :

Entity Types related to a given Process_ERS.

Arguments :

Process_ERS

Roles : *Number of Process_ERS related to a given Entity Type. [An Entity Type may be related to many Process_ERS]*

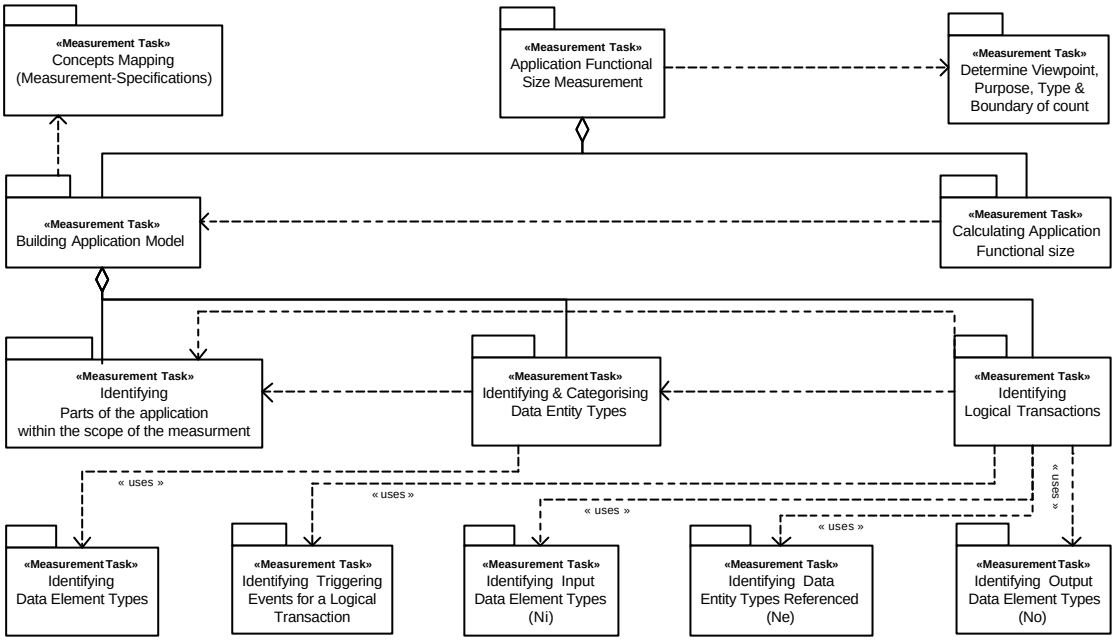
Cardinality : *min 0; max n*;

Entity_Type

Roles : *Number of Entity types related to a given Process_ERS. [A Process_ERS includes at least one Entity Type]*

Cardinality : *min 1; max n*;

LES TACHES



APPLICATION FUNCTIONAL SIZE MEASUREMENT

Definition :

Goal : *Given a set of specifications for a software application, a viewpoint, purpose, type and boundary of count, the task should produce a MKII-FPA model for the software application, and calculate its functional size in MkII-FPA Functional Size Units. It is a composition of two (2) complementary tasks: "Building Application Model", and "Calculating Application Functional size". It uses the results of the task « Determine Viewpoint, Purpose, Type & Boundary of count ».*

Inputs : *Set of specifications for a software application, Viewpoint_of_Count, Purpose_of_Count, Type_of_Count, Boundary_of_Count, Set of mapping results*

Outputs : *MkIIFPA model for the software application, functional size of the software application, [global precision/certainty of the measurement]*

Body :

Type : *composition*

Used Tasks : *« Determine Viewpoint, Purpose, Type & Boundary of count »*

Sub-Tasks : *"Building Application Model", "Calculating Application Functional size"*

Control loop : *execute sequentially :*

- *Building_Application_Model (+Set_of_specifications_for_a_software_application, +Viewpoint_of_Count, +Purpose_of_Count, +Type_of_Count, +Boundary_of_Count, +Set_Of_Mapping_Results, -MkIIFPA_model_for_the_ software_application),*
- *Calculating_Application_Functional_size (+MkIIFPA_model_for_the_software_application, -functional_size_of_the_software_application, -global_precision/certainty_of_the_measurement)*

BUILDING APPLICATION MODEL

Definition :

Goal : *Given a set of specifications for a software application, a viewpoint, purpose, type and boundary of count, and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should produce a MKII-FPA model for the software application. It's a composition of three (3) complementary tasks : « Identifying Parts of the application within the scope of the measurement », « Identifying & Categorising Data Entity Types », « Identifying Logical Transactions ». It uses the results of the task « Concepts Mapping ».*

Inputs : *Set of specifications for a software application, Viewpoint_of_Count, Purpose_of_Count, Type_of_Count, Boundary_of_Count, Set of mapping results.*

Outputs : *MkIIFPA model for the software application*

Body :

Type : *composition*

Used Tasks : *« Concepts Mapping »*

Sub-Tasks : *« Identifying Parts of the Application within the Scope of the Measurement », « Identifying & Categorising Data Entity Types », « Identifying Logical Transactions »*

Control loop : *execute sequentially :*

- *Identifying_Parts_of_the_application_within_the_scope_of_the_measurement (+Set_of_specifications_for_a_software_application, +Viewpoint_of_Count, +Purpose_of_Count, +Type_of_Count, +Boundary_of_Count, -Parts_of_the_application_within_the_scope_of_the_measurement[43])*
- *Identifying_&_Categorising_Data_Entity_Types (+Parts_of_the_application_within_the_scope_of_the_measurement[43], +Set_Of_Mapping_Results, -Entity_Type[])*
- *Identifying_Logical_Transactions (+Parts_of_the_application_within_the_scope_of_the_measurement[43], +Data_Entity_Type[], +Set_Of_Mapping_Results, -Logical_Transaction[])*

CALCULATING APPLICATION FUNCTIONAL SIZE

Definition :

Goal : *Given a MkII-FPA model for a given software application, the task should assign numeric values to some elements of the model (Logical transactions), then aggregate these values to derive the functional size (in MkII-FPA Functional Size Unit) of the considered software application according to the given viewpoint. The aggregation should take into account the uncertainty parameter associated with some elements of the model (logical transactions, Entity Types, data element types ...), to derive the global precision of the measurement. We are currently working on this issue.*

Inputs : *MkII-FPA model for the software application*

Outputs : *Functional size of the software application, [global precision/certainty of the measurement]*

Body :

Type : *simple*

Sub-Tasks : *;*

Control loop : *execute :*

- *Calculating_Application_Functional_Size (+MkII-FPA_model_for_the_software_application, -Functional_size_of_the_software_application, -Global_precision/certainty_of_the_measurement)*

[43] i.e: *subset of specifications related to parts of the software application which are in the scope of the measurement*

CONCEPTS MAPPING

Definition :

Goal : *To map measurement concepts (MkII-FPA concepts) with some concepts of a specification language (for example UML).*

Inputs : *MkII-FPA concepts, Specifications language concepts*

Outputs : *mapping results*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Concepts_Mapping (+MkII-FPA_Concept[ ], Specifications_Language_Concept[ ], -Mapping_Result[ ])*

IDENTIFYING PARTS OF THE APPLICATION WITHIN THE SCOPE OF THE MEASUREMENT

Definition :

Goal : *Given a set of specifications for a software application, a viewpoint, purpose, type and boundary of count, the task should determine the subset of specifications related to parts of the software application which are in the scope of the measurement.*

Inputs : *Set of specifications for a software application, Viewpoint_of_Count, Purpose_of_Count, Type_of_Count, Boundary_of_Count*

Outputs : *Parts_of_the_application_within_the_scope_of_the_measurement*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Parts_of_the_application_within_the_scope_of_the_measurement (+Set_of_specifications_for_a_software_application, +Viewpoint_of_Count, +Purpose_of_Count, +Type_of_Count, +Boundary_of_Count, -Parts_of_the_application_within_the_scope_of_the_measurement)*

IDENTIFYING & CATEGORIZING DATA ENTITY TYPES

Definition :

Goal : *Given a set of specifications related to parts of a software application which are in the scope of the measurement, and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine and categorize data entity types manipulated. It includes the task « Identifying Data Element Types ».*

Inputs : *Parts_of_the_application_within_the_scope_of_the_measurement, Set_Of_Mapping_Results*

Outputs : *, a list entity types manipulated*

Body :

Type : *simple*

Included Tasks : *« Identifying Data Element Types »*

Control loop : *execute :*

- *Identifying_&_Categorizing_Data_Entity_Types (+Parts_of_the_application_within_the_scope_of_the_measurement , +Set_Of_Mapping_Results, -Entity_Type[ ])*

IDENTIFYING DATA ELEMENT TYPES

Definition :

Goal : *Given a set of specifications related to parts of a software application which are in the scope of the measurement, a data entity type, and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the tasks should identify all data element types related to the data entity type.*

Inputs : *Parts_of_the_application_within_the_scope_of_the_measurement, Entity_Type, Set_Of_Mapping_Results*

Outputs : *List of data element types related to the entity type*

Body :

Type : *simple*

Control loop : *execute :*

- *Identifying_Data_Element_Types (+Parts_of_the_application_within_the_scope_of_the_measurement*

, +Entity_Type, +Set_Of_Mapping_Results, -Data_Element_Type[ ])

IDENTIFYING LOGICAL TRANSACTIONS

Definition :

Goal : *Given a set of specifications related to parts of a software application which are in the scope of the measurement, a set of entity types identified in it, and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should find the logical transactions in the specifications. The triggering event of each logical transaction and it components should be pointed out.*

Inputs : *Parts_of_the_application_within_the_scope_of_the_measurement, a set of entity types, Set_Of_Mapping_Results*

Outputs : *logical transactions found*

Body :

Type : *simple*

Used Tasks : *« Identifying triggering event for a Logical Transaction », « Identifying Input Data Element Types (Ni) », « Identifying Output Data Element Types (No) », « Identifying Data Entity Types Referenced (Ne) »*

Control loop : *execute :*

- *Identifying_Logical_Transactions (+Parts_of_the_application_within_the_scope_of_the_measurement , +Entity_Type[], +Set_Of_Mapping_Results, -Logical_Transaction[])*

IDENTIFYING TRIGGERING EVENT FOR A LOGICAL TRANSACTION

Definition :

Goal : *Given a set of specifications related to parts of a software application which are in the scope of the measurement, a logical transaction and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine the event that triggers the logical transaction.*

Inputs : *Parts_of_the_application_within_the_scope_of_the_measurement, a logical transaction, Set_Of_Mapping_Results*

Outputs : *event triggering the given logical transaction*

Body :

Type : *simple*

Used Tasks : *«Identifying user related to a triggering event»*;

Control loop : *execute :*

- *Identifying_Triggering_Event_For_A_Logical_Transaction (+Parts_of_the_application_within_the_scope_of_the_measurement , +Logical_Transaction, +Set_Of_Mapping_Results, -Triggering_Event)*

IDENTIFYING_USER_RELATED_TO_A_TRIGGERING_EVENT

Definition :

Goal : *Given a set of specifications related to parts of a software application which are in the scope of the measurement, a logical transaction, the triggering event for the logical transaction and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine the user who produced the triggering event.*

Inputs : *Parts_of_the_application_within_the_scope_of_the_measurement, a logical transaction, the triggering event for the logical transaction, Set_Of_Mapping_Results*

Outputs : *user who produced the triggering event*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Related_To_A_Triggering_Event (+Parts_of_the_application_within_the_scope_of_the_measurement , +Logical_Transaction, +Triggering_event_for_the_logical_transaction, +Set_Of_Mapping_Results, -User_who_produced_the_Triggering_Event)*

IDENTIFYING INPUT DATA ELEMENT TYPES (NI)

Definition :

Goal : *Given a set of specifications related to parts of a software application which are in the scope of the measurement, a logical transaction, the set of data element types identified in the specifications and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine the subset of data element types input by the logical transaction.*

Inputs : *Parts_of_the_application_within_the_scope_of_the_measurement, a logical transaction, set of data element types, Set_Of_Mapping_Results*

Outputs : *a set of data element types input by the logical transaction (Input_DETS)*

Body :

Type : *simple*

Control loop : *execute :*

- *Identifying_Input_Data_Element_Types (+Parts_of_the_application_within_the_scope_of_the_measurement , +Logical_Transaction, +data_Element_Type[], +Set_Of_Mapping_Results, -Input_DETS)*

IDENTIFYING OUTPUT DATA ELEMENT TYPES (NO)

Definition :

Goal : *Given a set of specifications related to parts of a software application which are in the scope of the measurement, a logical transaction, the set of data element types identified in the specifications and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine the subset of data element types output by the logical transaction.*

Inputs : *Parts_of_the_application_within_the_scope_of_the_measurement, a logical transaction, set of data element types, Set_Of_Mapping_Results*

Outputs : *a set of data element types output by the logical transaction (Output_DETS)*

Body :

Type : *simple*

Control loop : *execute :*

- *Identifying_Output_Data_Element_Types (+Parts_of_the_application_within_the_scope_of_the_measurement , +Logical_Transaction, +data_Element_Type[], +Set_Of_Mapping_Results, -Output_DETS)*

IDENTIFYING DATA ENTITY TYPES REFERENCED (NE)

Definition :

Goal : *Given a set of specifications related to parts of a software application which are in the scope of the measurement, a logical transaction, the set of data entity types identified in the specifications and a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), the task should determine the subset of data entity types processed/referenced by the logical transaction.*

Inputs : *Parts_of_the_application_within_the_scope_of_the_measurement, a logical transaction, set of data entity types, Set_Of_Mapping_Results*

Outputs : *a set of data entity types processes by the logical transaction (Process_ERS)*

Body :

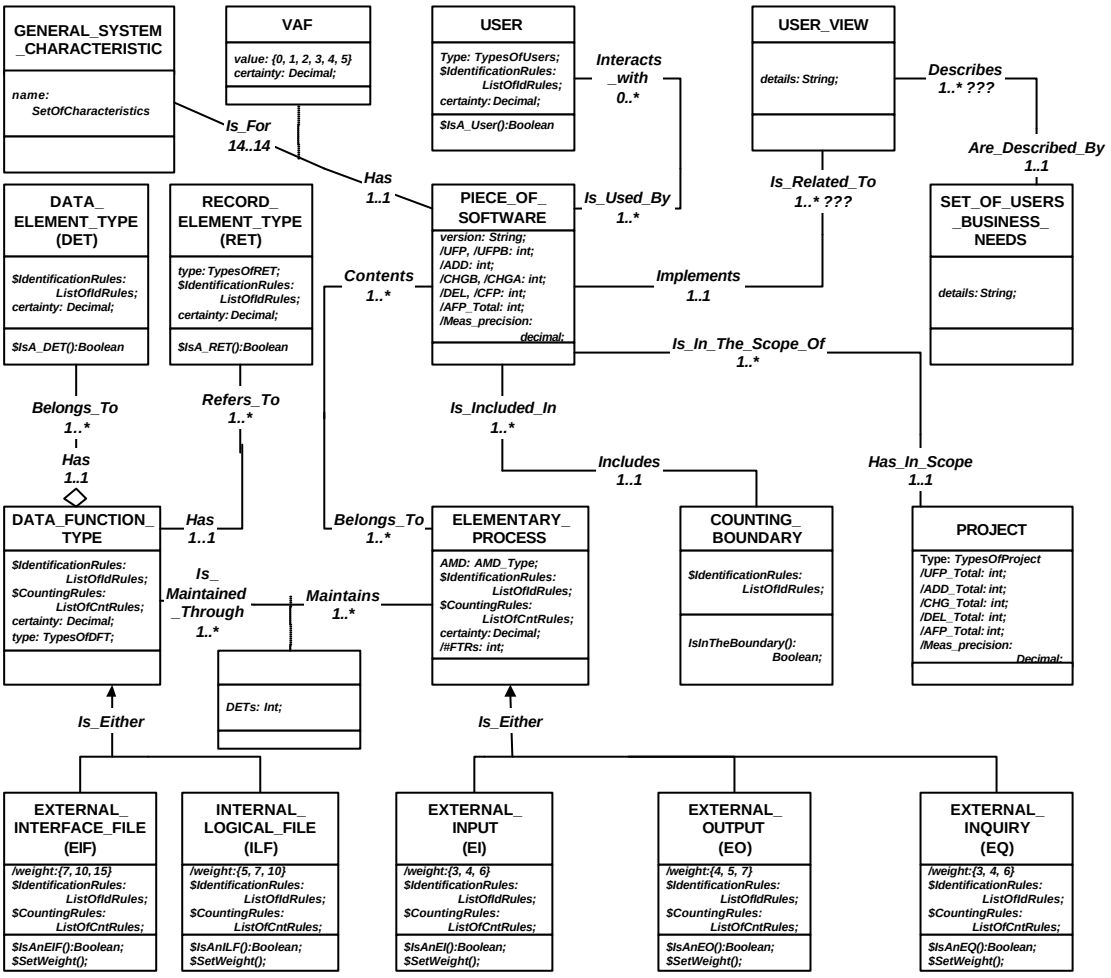
Type : *simple*

Control loop : *execute :*

- *Identifying_ Data_Entity_Types_Referenced (+Parts_of_the_application_within_the_scope_of_the_measurement , +Logical_Transaction, +data_Entity_Type[], +Set_Of_Mapping_Results, -Process_ERS)*

ANNEXE D : DESCRIPTION DES CONCEPTS ET TACHES FPA

LES CONCEPTS



* TypesOfUsers = {'Person', 'Thing'} * TypesOfUsers = {'Person', 'Thing'} * AMD_Type = {'added', 'deleted', 'updated'} * TypesOfProject = {'Development', Enhancement'"}
* SetOfCharacteristics = {'Data communication', 'System distribution', 'Performance', 'Intensive use configuration', 'Transaction rate', 'Interactive input', 'Ease of use', 'Real-time update', 'Processing complexity', 'Reuse', 'Ease of installation', 'Ease of exploitation', 'Portability', 'Ease of adaptation'}

SET_OF_USERS_BUSINESS_NEEDS

Definition :

A Set of users business needs is a set of information provided by users and translated by developers into an information technology language in order to

provide a solution. A function point count is accomplished using the information in a language that is common to both users and developers.

Properties :

- *Description : {String};*
- *Details : {String};*

PROJECT

Definition :

A project stands here for a measurement project (development, enhancement or application function point count).

Properties :

- *Name : {String};*
- *Description : {String};*
- *TypeOfProject : { TypeOfProject }; /* TypeOfProject = {'Development', 'Enhancement', 'Application'} */*
- *ADD : {Integer}; /* This value indicates the amount of unadjusted Functional Size Units related to adding of functionalities for this project (enhancement or development project) */*
- *CHG : {Integer}; /* This value indicates the amount of unadjusted Functional Size Units related to modifications of functionalities for this project (enhancement project) */*
- *DEL : {Integer}; /* This value indicates the amount of unadjusted Functional Size Units related to deletions of functionalities for this project (enhancement project) */*
- *Unadjusted_FPA_Count : {Integer}; /* This value indicates the specific countable functionality provided to the user by the project or application. It has two function types: Data and transactional */*
- *Adjusted_FPA_Count : {Integer}; /* This value indicates the total function point count based on the unadjusted function point count multiplied by the value adjustment factor, for this project. The count is calculated using a specific formula for development project, enhancement project and application. The adjusted count is commonly called the function point count */*
- *Measurement_Precision : {decimal}; /* This value indicates the confidence related to the functional size obtained */*

PIECE_OF_SOFTWARE

Definition :

A piece of software stands here for a software application or part of it. It's a cohesive collection of automated procedures and data supporting a business objective. It consists of one or more components, modules, or subsystems. It is frequently used as a synonym for system.

Properties :

- *Name : {String};*
- *Description : {String};*
- *Version : {String};*
- *ADD : {Integer}; /* This value indicates the amount of unadjusted Functional Size Units related to adding of functionalities for this project (enhancement or development project) */*
- *CFP : {Integer}; /* This value indicates for a development project the amount of unadjusted Functional Size Units related to converting data and/or providing other user-specified conversion requirement, such as special conversion reports. For an enhancement project this value indicates the amount of unadjusted Functional Size Units related to converting functionality required by the user (enhancement or development project) */*
- *CHGB : {Integer}; /* This value indicates the amount of unadjusted function points for this project **before** modifications of functionalities (enhancement project) */*
- *CHGA : {Integer}; /* This value indicates the amount of unadjusted function points for this project **after** modifications of functionalities (enhancement project) */*
- *DEL : {Integer}; /* This value indicates the amount of unadjusted function points related to deletions of functionalities for this project (enhancement project) */*
- *Unadjusted_FPA_Count : {Integer}; /* This value indicates the specific countable functionality provided to the user by the project or application. It has two function types: Data and transactional */*
- *Adjusted_FPA_Count : {Integer}; /* This value indicates the total function point count based on the unadjusted function point count multiplied by the value adjustment factor, for this project. The count is calculated using a specific formula for development project, enhancement project and application. The adjusted count is commonly called the function point count */*

- *Measurement_Precision : {decimal}; /* This value indicates the confidence related to the functional size obtained */*

GENERAL SYSTEM CHARACTERISTIC (GSC)

Definition:

A GSC is specific characteristic identified by the FPA measurement method for an application.

Properties:

- *name: { 'Data communication', 'System distribution', 'Performance', 'Intensive use configuration', 'Transaction rate', 'Interactive input', 'Ease of use', 'Real-time update', 'Processing complexity', 'Reuse', 'Ease of installation', 'Ease of exploitation', 'Portability', 'Ease of adaptation' };*

VALUE ADJUSTMENT FACTOR (VAF)

Definition:

A VAF is a value that indicates the general functionality provided to the user of the application. The VAF is calculated based on an assessment of the 14 general system characteristics (GSCs) for an application.

Properties:

- *Value: {Integer};*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in affecting this value to a given system characteristic for the given application. It will be used when calculating the precision associated with the functional size */*

MEASUREMENT BOUNDARY

Definition:

An application Boundary refers to the conceptual interface between "internal" application and "external" user world. The application boundary defines what is external to the application. It acts as a "membrane" through which data passes into and out from the application; it encloses the logical data maintained by the application, assists in identifying logical data referenced by, not maintained within this application. It is dependent upon the user's external business view of the application.

Properties:

- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

USER_VIEW

Definition :

A user view represents a formal description of the user's business needs in the user's language. Developers translate the user information into information technology language in order to provide a solution. A function point count is accomplished using the information in a language that is common to both users and developers.

Properties :

- *Description : {String};*
- *Details : {String};*

USER

Definition :

(ISO Definition) A user is any person that specifies Functional User Requirements and/or any person or thing that communicates or interacts with the software at any time.

Note: *(1) A user can be seen as the person or organization that uses the measured application. Included would be the requirements author, end users, management users, auditors, and operations. (2) A user can also be seen as the human being that use the application.*

Properties :

- *Name : {String};*
- *Description : {String};*
- *Type : {TypeOfUser}; /* TypeOfUser = {'Human Being', 'Thing' } */*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a user. It will be used when calculating the precision associated with the functional size */*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

ELEMENTARY_PROCESS / TRANSACTIONAL FUNCTION TYPE

Definition:

An Elementary Process is the smallest unit of activity that is meaningful to the end user in the business. It must be self-contained and leave the business of the application being counted in a consistent state. It is also called a ***transactional function type.***

Properties:

- *AMD (AddedModifiedDeleted) : {AMD_Type}; /* AMD_Type = {'added', 'deleted', 'updated'} */*
- *name : {String};*
- *Description: {String};*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as an elementary process. It will be used when calculating the precision associated with the functional size */*
- *#FTRs : {Integer}; /* This value indicates the total number of File Types Referenced by this elementary process */*
- *Weight : {Integer}; /* This value indicates the number of unadjusted function points to be counted for this elementary process */*
- *IdentificationRule[]: {IdRule}*
- *MeasurementRule[]: {CntRule}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

EXTERNAL_INPUT(EI)

Definition :

An External Input (EI) is an elementary process that processes data or control information that comes from outside the application boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behaviour of the system.

Properties :

- *AMD (AddedModifiedDeleted) : {AMD_Type}; /* AMD_Type = {'added', 'deleted', 'updated'} */*
- *name : {String};*
- *Description: {String};*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as an external input. It will be used when calculating the precision associated with the functional size */*
- *#FTRs : {Integer}; /* This value indicates the total number of File Types Referenced by this external input */*
- *Weight : {3, 4, 5}; /* This value indicates the number of unadjusted function points to be counted for this external input */*
- *IdentificationRule[]: {IdRule}*
- *MeasurementRule[]: {CntRule}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

EXTERNAL_OUTPUT (EO)

Definition :

An External Output (EO) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information. The processing logic must contain at least one mathematical formula or calculation, or create derived data. An external output may also maintain one or more ILFs and/or alter the behaviour of the system.

Properties :

- *AMD (AddedModifiedDeleted) : {AMD_Type}; /* AMD_Type = {'added', 'deleted', 'updated'} */*
- *name : {String};*
- *Description: {String};*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as an external output. It will be used when calculating the precision associated with the functional size */*
- *#FTRs : {Integer}; /* This value indicates the total number of File Types Referenced by this external output */*
- *Weight : {4, 5, 7}; /* This value indicates the number of unadjusted function points to be counted for this external output */*
- *IdentificationRule[]: {IdRule}*
- *MeasurementRule[]: {CntRule}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

EXTERNAL_INQUIRY (EQ)

Definition :

An External Inquiry (EQ) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF or EIF. The processing logic contains no mathematical formulas or calculations, and creates no derived data. No ILF is maintained during the processing, nor is the behaviour of the system altered.

Properties :

- *AMD (AddedModifiedDeleted) : {AMD_Type}; /* AMD_Type = {'added', 'deleted', 'updated'} */*
- *name : {String};*
- *Description: {String};*

- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as an External Inquiry. It will be used when calculating the precision associated with the functional size */*
- *#FTRs : {Integer}; /* This value indicates the total number of File Types Referenced by this External Inquiry */*
- *Weight : {3, 4, 6}; /* This value indicates the number of unadjusted function points to be counted for this External Inquiry */*
- *IdentificationRule[]: {IdRule}*
- *MeasurementRule[]: {CntRule}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

DATA FUNCTION TYPE

Definition:

A. Data function type represents the functionality provided to the user to meet internal and external data requirement. It is either an internal logical file or an external interface file.

Properties:

- *name :{String};*
- *Description: {String};*
- *Type : { TypeOfDFT}; /* TypeOfDFT = {'logically related data', 'control information' } */*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as a data function type. It will be used when calculating the precision associated with the functional size */*
- *Weight : {Integer}; /* This value indicates the number of unadjusted function points to be counted for this data function type */*
- *IdentificationRule[]: {IdRule}*
- *MeasurementRule[]: {CntRule}*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

INTERNAL LOGICAL FILE (ILF)

Definition :

An Internal Logical File (ILF) is a user identifiable group of logically related data or control information maintained within the boundary of the application. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted.

Properties :

- *Name : {String};*
- *Description : {String};*
- *Type : { TypeOfILF}; /* TypeOfILF = {'logically related data', 'control information' } */*
- *Weight : {5, 7, 10}; /* This value indicates the number of function points related to adding of functionalities for this measurement viewpoint (enhancement or development project) */*
- *IdentificationRule[ ] : {IdRule}*
- *MeasurementRule[ ] : {CntRule}*
- *Certainty: {Decimal}; /* This value indicates how certain the measurer is in identifying this as an Internal Logical File. It will be used when calculating the precision associated with the functional size */*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

EXTERNAL INTERFACE FILE (EIF)

Definition :

An External Interface File (EIF) is a user identifiable group of logically related data or control information referenced by the application, but maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application being counted. This means that an EIF counted for an application must be an ILF in another application.

Properties :

- *Type : { TypeOfEIF}; /* TypeOfEIF = {'logically related data', 'control information' } */*
- *Name : {String};*
- *Description : {String};*
- *Weight : {7, 10, 15}; /* This value indicates the number of function points related to adding of functionalities for this measurement viewpoint (enhancement or development project) */*
- *IdentificationRule[ ] : {IdRule}*
- *MeasurementRule[ ] : {CntRule}*
- *Certainty: {Decimal}; /* This value indicates how certain the measurer is in identifying this as an External Interfacel File. It will be used when calculating the precision associated with the functional size */*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

Concept: *CntRule*

Condition: {logical expression}

Action: {procedure}

DATA_ ELEMENT_TYPE (DET)

Definition :

A data element type is a unique user recognizable, non recursive field of a data function type. The number of DETs is used to determine the complexity each function type and the function type's contribution to the unadjusted function point count.

Properties :

- *Name : {String};*
- *Description : {String};*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as data element type. It will be used when calculating the precision associated with the functional size */*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

RECORD_ELEMENT_TYPE (RET)

Definition :

A record element type is a user recognizable subgroup of data elements within an ILF or an EIF. There are two types of RETs: mandatory and optional.

Properties :

- *Name : {String};*
- *Description : {String};*
- *Type : { TypeOfRET}; /* TypeOfRET ={ 'mandatory', 'optional'} */*
- *IdentificationRule[ ] : {IdRule}*
- *Certainty : {Decimal}; /* This value indicates how certain the measurer is in identifying this as record element type. It will be used when calculating the precision associated with the functional size */*

Concept: *IdRule*

Condition: {logical expression}

Action: {procedure}

LES ASSOCIATIONS ENTRE CONCEPTS

SET_OF_USERS_BUSINESS_NEEDS – USER_VIEW

Description :

User views for a set of users business needs.

Arguments :

User_View

Roles : *Number of user views related to a given set of users business needs. [A given set of users business needs can be related to many user views]*

Cardinality : *min 0; max n*;

Set_Of_Users_Business_Needs

Roles : *Number of sets of users business needs related to a given user view. [A given user view describes only one set of users business needs]*

Cardinality : *min 1; max 1*;

PIECE_OF_SOFTWARE – USER_VIEW

Description :

Pieces of software implementing a given User view.

Arguments :

User_View

Roles : *Number of user views implemented in a given piece of software. [A given piece of software can implement many user views ???]*

Cardinality : *min 1; max n*;

Piece_Of_Software

Roles : *Number of pieces of software implementing a given user view. [A given user view can be implemented in many pieces of software ???]*

Cardinality : *min 0; max n*;

PIECE_OF_SOFTWARE – USER

Description :

Users for a given piece of software.

Arguments :

User

Roles : *Number of users interacting with a given piece of software. [A given piece of software can be used by many users]*

Cardinality : *min 1; max n*;

Piece_Of_Software

Roles : *Number of pieces of software used by a given user. [A given user can interact with many pieces of software]*

Cardinality : *min 0; max n*;

PIECE_OF_SOFTWARE – PROJECT

Description :

Pieces of software within the scope of a given project.

Arguments :

Project

Roles : *Number of projects having in scope a given piece of software. [A given piece of software can be in the scope of only one project]*

Cardinality : *min 1; max 1*;

Piece_Of_Software

Roles : *Number of pieces of software within the scope of a given project. [A given project can have in scope many pieces of software]*

Cardinality : *min 1; max n*;

PIECE_OF_SOFTWARE - MEASUREMENT_BOUNDARY

Description :

The Measurement boundary for a set of given pieces of software.

Arguments :

Piece_Of_Software

Roles : *Number of pieces of software related to a Measurement boundary. [A Measurement boundary may be defined for many pieces of software]*

Cardinality : *min 1; max n*;

Measurement_Boundary

Roles : *Number of Measurement boundaries for a given piece of software. [A given piece of software is related to one Measurement boundary]*

Cardinality : *min 1; max 1*;

VAF: PIECE_OF_SOFTWARE - GENERAL_SYSTEM_CHARACTERISTIC

Description :

The value adjustment factor affected to a general system characteristic for a given piece of software.

Arguments :

Piece_Of_Software

Roles : *Number of pieces of software for which a given general system characteristic has a VAF set. [A general system characteristic may have a VAF set for many pieces of software]*

Cardinality : *min 1; max n*;

General_System_Characteristic

Roles : *Number of general system characteristic which have a VAF set for a given piece of software. [A given piece of software may have many (0 to 14) general system characteristics with a VAF set for the piece of software]*

Cardinality : *min 0; max 14*;

PIECE_OF_SOFTWARE – ELEMENTARY_PROCESS

Description :

The number of elementary processes for a given piece of software.

Arguments :

Piece_Of_Software

Roles : *Number of pieces of software to which a given elementary process is related. [An elementary process belongs to only one piece of software]*

Cardinality : *min 1; max 1*;

Elementary_Process

Roles : *Number of elementary processes for a given piece of software. [A given piece of software may have many elementary processes]*

Cardinality : *min 1; max n*;

ELEMENTARY_PROCESS – EXTERNAL_INPUT (EI) – EXTERNAL_OUTPUT (EO) – EXTERNAL_INQUIRY(EQ)

Description :

The nature of an elementary process (EI, EO or EQ).

Arguments :

Elementary_Process

Roles : *Number of elementary processes for a given EI, EO, or EQ. [A given EI, EO, or EQ is related to only one elementary process]*

Cardinality : *min 1; max 1*;

External_Input (EI)

Roles : *Number of External_Inputs related to a given elementary process. [An elementary process is related to only one external input]*

Cardinality : *min 0; max 1*;

External_Output (EO)

Roles : *Number of External_Outputs related to a given elementary process. [An elementary process is related to only one external output]*

Cardinality : *min 0; max 1*;

External_Inquiry (EQ)

Roles : *Number of External_Inquiries related to a given elementary process. [An elementary process is related to only one external Inquiry]*

Cardinality : *min 0; max 1*;

ELEMENTARY_PROCESS – DATA_FUNCTION_TYPE (DFT)

Description :

The data function types maintained by an elementary process.

Arguments :

Elementary_Process

Roles : *Number of elementary processes maintaining a given data function type. [A given data function type can be maintained through many elementary processes]*

Cardinality : *min 1; max n*;

Data_Function_Type (DTF)

Roles : *Number of Data function Types maintained through a given elementary process. [An elementary process may maintain many data function types]*

Cardinality : *min 1; max n*;

DATA_FUNCTION_TYPE (DFT) – EXTERNAL_INTERFACE_FILE (EIF) – INTERNAL_LOGICAL_FILE (ILF)

Description :

The nature of a data function type.

Arguments :

Data_Function_Type (DTF)

Roles : *Number of Data function Types for a given external interface file or internal logical file. [An external interface file or internal logical file is related to one data function type]*

Cardinality : *min 1; max 1*;

External_Interface_File (EIF)

Roles : *Number of External_Interface_Files related to a given data function type. [A data function type is related to only one external interface file]*

Cardinality : *min 0; max 1*;

Internal_Logical_File (ILF)

Roles : *Number of Internal_Logical_Files related to a given data function type. [A data function type is related to only one internal logical file]*

Cardinality : *min 0; max 1*;

DATA_FUNCTION_TYPE (DFT) – DATA_ELEMENT_TYPE (DET)

Description :

The data element types for a data function type.

Arguments :

Data_Function_Type (DTF)

Roles : *Number of Data function Types for a given data element type. [A data element type is related to one data function type]*

Cardinality : *min 1; max 1*;

Data_Element_Type (DET)

Roles : *Number of data element types related to a given data function type. [A data function type may be related to many data element types]*

Cardinality : *min 1; max n*;

DATA_FUNCTION_TYPE (DFT) – RECORD_ELEMENT_TYPE (RET)

Description :

The record element types for a data function type.

Arguments :

Data_Function_Type (DTF)

Roles : *Number of Data function Types for a given record element type. [A record element type is related to one data function type]*

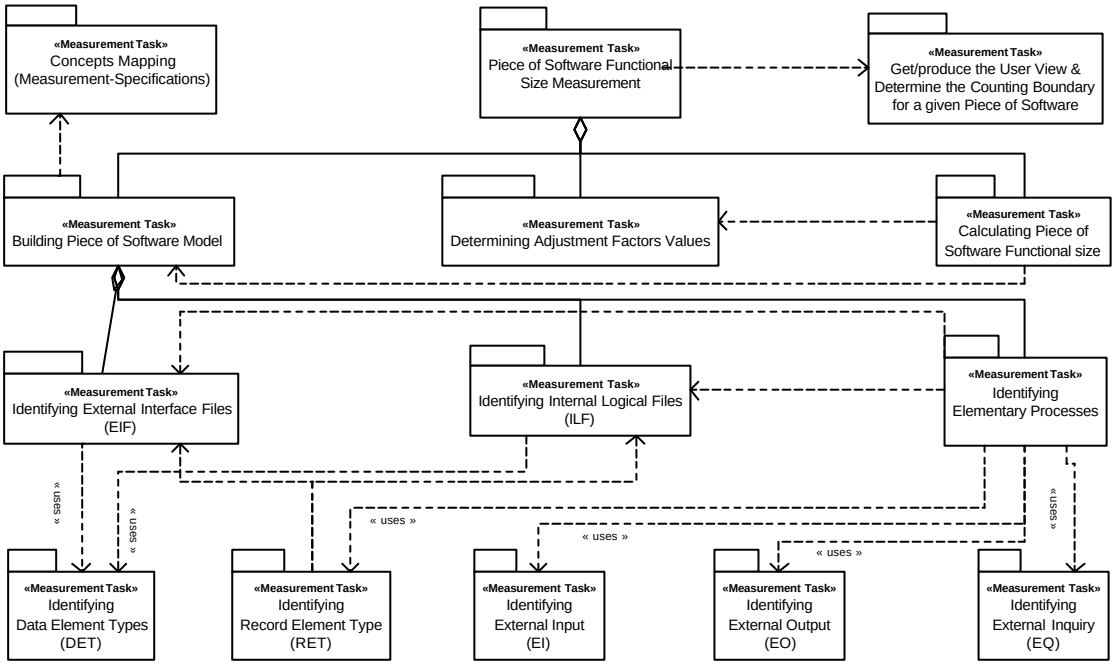
Cardinality : *min 1; max 1*;

Record_Element_Type (RET)

Roles : *Number of record element types related to a given data function type. [A data function type may be related to many record element types]*

Cardinality : *min 1; max n;*

LES TACHES



PIECE OF SOFTWARE FUNCTIONAL SIZE MEASUREMENT

Definition:

Goal: *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software and a Measurement boundary, the task should produce a FPA model for the piece of software, determine the related adjustment factors values and calculate its functional size in FPA Functional Size Units. It is a composition of three (3) complementary tasks: "Building Application*

Model", "Determining Adjustment Factors Values" and "Calculating Application Functional size".

Inputs: *User view, Measurement boundary, Set of mapping results, Project characteristics, Adjustment factor values*

Outputs: *FPA model for the piece of software, functional size of the piece of software, [global precision/certainty of the measurement]*

Body:

Type: *composition*

Sub-Tasks: *"Building Piece of Software Model", "Determining Adjustment Factors Values" and "Calculating Piece of Software Functional size"*

Control loop: *execute sequentially:*

- *Building_Piece_of_Software_Model (+User_View, +Measurement_Boundary, +Set_Of_Mapping_Results[], -FPA_model_for_the_piece_of_software),*
- *Determining_Adjustment_Factors_Values (+User_View, +Project_Characteristics, - Adjustment_Factors_Values[]),*
- *Calculate_Piece_of_Software_Functional_Size (+FPA_model_for_the_piece_of_software, +Adjustment_Factors_Values[], -functional_size_of_the_piece_of_software, -global_precision/certainty_of_the_measurement)*

BUILDING PIECE OF SOFTWARE MODEL

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software, a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts) and a Measurement boundary, the task should produce a FPA model for the piece of software. It's a composition of four (4) complementary tasks : « Identifying External Interface Files », « Identifying Internal Logical Files », «Identifying Elementary processes », and « Produce the FPA model of the piece of software ». It uses the results of the task « Concepts Mapping ».*

Inputs : *User view, Measurement boundary, Set of mapping results.*

Outputs : *FPA model for the piece of software*

Body :

Type : *composition*

Used Tasks : *« Concepts Mapping »*

Sub-Tasks : *« Identifying External Interface Files », « Identifying Internal Logical Files », « Identifying Elementary processes »*

Control loop : *execute sequentially :*

- *Identifying_External_Interface_Files (+User_Viewpoint, +Set_Of_Mapping_Results[], +Measurement_Boundary, -External_Interface_File[])*
- *Identifying_Internal_Logical_Files (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, -Internal_Logical_File[])*
- *Identifying_Elementary_Processes (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, -Elementary_Processes[])*
- *Produce_The_FPA_Model_Of_The_Piece_Of_Software(+User_Viewpoint , +External_Interface_File[], +Internal_Logical_File[], +Elementary_Processes[], -FPA_model_for_the_piece_of_software)*

DETERMINING ADJUSTMENT FACTORS VALUES

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software and the set of general system characteristics proposed in FPA, the task should determine the value adjustment factor related to the piece of software for each general system characteristic.*

Inputs : *User view, General system characteristics*

Outputs : *Value adjustment factor[]*

Body :

Type : *simple*

Sub-Tasks : *;*

Control loop : *execute :*

- *Determining_Value_Adjustment_Factors (+User_View, +General_System_Characteristics, -Value_Adjustment_Factor[])*

CALCULATING PIECE OF SOFTWARE FUNCTIONAL SIZE

Definition :

Goal : *Given a FPA model for a given piece of software, the values adjustment factors related to the piece of software, the task should derive the functional size (unadjusted & adjusted) (in FPA Functional Size Unit) of the considered piece of software. The calculation should take into account the uncertainty parameter associated with some elements of the model (data function types, transactional function types ...), to derive the global precision of the measurement. We are currently working on this issue.*

Inputs : *FPA model for the piece of software*

Outputs : *Functional size (unadjusted & adjusted) of the piece of software, [global precision/certainty of the measurement]*

Body :

Type : *simple*

Sub-Tasks : *;*

Control loop : *execute :*

- *Calculating_Piece_of_Software_Functional_Size (+FPA_model_for_the_piece_of_software, -Functional_size_of_the_piece_of_software, -Global_precision/certainty_of_the_measurement)*

CONCEPTS MAPPING

Definition :

Goal : *To map measurement concepts (FPA concepts) with some concepts of a specification language (for example UML).*

Inputs : *FPA concepts, Specifications language concepts*

Outputs : *mapping results*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Concepts_Mapping (+FPA_Concept[ ], Specifications_Language_Concept[ ], -Set_Of_Mapping_Results[ ])*

IDENTIFYING EXTERNAL INTERFACE FILES

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software (an application), a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts) and a Measurement boundary, the task should identify all External Interface Files for this piece of software.*

Inputs : *User viewpoint, Set of mapping results, Measurement_Boundary*

Outputs : *External_Interface_Files[]*

Body :

Type : *simple*

Used Tasks : *« Identifying Data Element Types », « Identifying Record Element Types »*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_External_Interface_Files (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, - External_Interface_File[])*

IDENTIFYING INTERNAL LOGICAL FILES

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software, a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts) and a Measurement boundary, the task should identify all Internal Logical Files for this piece of software.*

Inputs : *User viewpoint, Set of mapping results, Measurement_Boundary*

Outputs : *Internal_Logical_Files[]*

Body :

Type : *simple*

Used Tasks : *« Identifying Data Element Types », « Identifying Record Element Types »*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Internal_Logical_Files (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, - Internal_Logical_File[])*

IDENTIFY ELEMENTARY PROCESSES

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software (an application), a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts) and a Measurement boundary, the task should identify all Internal Logical Files for this piece of software.*

Inputs : *User viewpoint, Set of mapping results, Measurement_Boundary*

Outputs : *Elementary Process []*

Body :

Type : *simple*

Used Tasks : *« Identifying External Inputs », « Identifying External Outputs », « Identifying External Inquiries », « Identifying Record Element Types »*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Elementary_Processes (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, - Elementary_Process[])*

IDENTIFY DATA ELEMENT TYPES

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software, a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), an ILF or an EIF, and a Measurement boundary, the task should identify all Data Element Types related to the ILF or the EIF.*

Inputs : *User viewpoint, Set of mapping results, Measurement_Boundary, ILF or EIF*

Outputs : *Data Element Type []*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Data_Element_Types (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, +ILF or EIF, -Data_Element_Type[])*

IDENTIFY RECORD ELEMENT TYPES

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software (an application), a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts), an ILF or an EIF, and a Measurement boundary, the task should identify all Record Element Types related to the ILF or the EIF.*

Inputs : *User viewpoint, Set of mapping results, Measurement_Boundary, ILF or EIF*

Outputs : *Record Element Type []*

Body :

Type : *simple*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_Record_Element_Types (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, +ILF or EIF, -Record_Element_Type[])*

IDENTIFY EXTERNAL OUTPUTS

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software, a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts) and a Measurement boundary, the task should identify all External Outputs related to the piece of software.*

Inputs : *User viewpoint, Set of mapping results, Measurement_Boundary*

Outputs : *External Output []*

Body :

Type : *simple*

Used Tasks : *« Identifying Record Element Types »*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_External_Outputs (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, - External_Output[])*

IDENTIFY EXTERNAL INPUTS

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software, a set of mapping results (resulting from a mapping between measurement concepts and some specifications language concepts) and a Measurement boundary, the task should identify all External Inputs related to the piece of software.*

Inputs : *User viewpoint, Set of mapping results, Measurement_Boundary*

Outputs : *External Input []*

Body :

Type : *simple*

Used Tasks : *« Identifying Record Element Types »*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_External_Inputs (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, - External_Input[])*

IDENTIFY EXTERNAL INQUIRIES

Definition :

Goal : *Given a user view (formal description of the user's business functions in the user's language) related to a piece of software, a set of mapping results (resulting from a mapping between measurement concepts and some*

specifications language concepts) and a Measurement boundary, the task should identify all External Inquiries related to the piece of software.

Inputs : *User viewpoint, Set of mapping results, Measurement_Boundary*

Outputs : *External Inquiry[]*

Body :

Type : *simple*

Used Tasks : *« Identifying Record Element Types »*

Sub-Tasks : ;

Control loop : *execute :*

- *Identifying_External_Inquiries (+User_Viewpoint, +Set_Of_Mapping_Results[], + Measurement_Boundary, - External_Inquiry[])*

ANNEXE E : CORRESPONDANCES ENTRE LES CONCEPTS COSMIC-FFP[44] ET DES CONCEPTS U.M.L.[45]

CAS D'UTILISATION & PROCESSUS FONCTIONNEL COSMIC-FFP

DÉFINITION DE CONCEPTS COSMIC-FFP

Fonctionnalité utilisateur requise (Functional user requirements (FUR)): Il s'agit d'une expression ISO désignant un sous ensemble des besoins de l'utilisateur. Le FUR représente les pratiques et procédures de l'utilisateur que le logiciel doit accomplir pour répondre aux besoins de celui-ci. Le FUR exclut les besoins en matière de qualité et les besoins techniques[46].

Processus fonctionnel (Functional process): Un processus fonctionnel est un ensemble unique et ordonné de mouvements de données (entry, read, write, exit) implémentant un ensemble cohésif de requis fonctionnels d'utilisateurs. (un ensemble cohésif de processus fonctionnels réalise un tache unique et nécessite peu d'interactions avec les autres ensembles de processus fonctionnels)

DÉFINITION DE CONCEPTS UML

Cas d'utilisation : Il s'agit d'une classe représentant une unité de fonctionnalité fournie par un système, un sous système ou une classe, traduite par des séquences de messages échangés entre le système (sous système, classe) et un ou plusieurs interacteurs extérieurs (appelées acteurs), ainsi que les actions effectuées par le système (sous système, classe).

Instance de cas d'utilisation : Il s'agit d'une séquence d'actions effectuées par un système, qui produit pour un acteur donné un résultat de valeur observable. *(généralement, les scénarios illustrent les instances de cas d'utilisation, à raison d'un scénario par instance)*

RAPPROCHEMENT

De ce qui précède, il apparaît qu'un processus fonctionnel COSMIC-FFP se rapproche d'une instance de cas d'utilisation UML. En effet, qu'il s'agisse d'une instance de cas d'utilisation UML ou d'un processus fonctionnel COSMIC-FFP, tous les deux représentent une suite d'actions (considérées comme mouvements de données dans COSMIC-FFP) effectuées par le système, et dont la finalité est la satisfaction d'un des requis fonctionnels des utilisateurs (considérés comme acteurs dans UML) du système.

---

44 Issus du document [ISO/IEC 19761: 2003]

45 Issus du document *UML Semantics Appendix M1-UML Glossary, version 1.0, 13 january 1997, p 16*

46 From "ISO/IEC 14143-1998 – Software Engineering – Software measurement – Functional size measurement – Part 1: Definition of concepts", clause 3.8.

DIAGRAMME DE SEQUENCES & SÉQUENCE DE MOUVEMENTS DE DONNÉES COSMIC-FFP

DÉFINITION DE CONCEPTS COSMIC-FFP

Composant de base fonctionnel (Base Functional Component) : Un composant de base fonctionnel est une unité élémentaire des besoins fonctionnels de l'utilisateur, définie par la méthode de mesure de la taille fonctionnelle (MTF) pour les fins de la mesure[47].

Mouvement de données [type] (Data movement type): Un mouvement de données est un composant de base fonctionnel qui déplace un ou plusieurs attributs appartenant à un seul groupe de données *(Il y a quatre sous-types de mouvements de données: entrée (entry), sortie (exit), lecture (read), écriture (write)).*

Entrée (Entry): Une entrée déplace les valeurs d'attributs d'un groupe de données depuis un utilisateur à travers la frontière vers le processus fonctionnel où il est requis. *(une entrée ne met pas à jour la donnée qu'elle déplace*.

Sortie (Exit): Une sortie (S) est un type de mouvement de donnée qui déplace un groupe de donnée d'un processus fonctionnel à travers la frontière vers l'utilisateur qui le demande. *(une sortie ne lit pas la donnée qu'elle déplace*.

Lecture (Read): Une lecture (L) est un type de mouvement de données qui place un groupe de données de la partie de stockage à la portée du processus fonctionel auquel il appartient. Une lecture fait référence aux attrbuts de données d'un groupe de données sans en modifier les valeurs. Sur le plan fonctionnel, une lecture prend des données se trouvant à l'extérieur de la frontière du logiciel, dans la partie stockage, et les met à la disposition du processus fonctionnel auquel elle appartient.

Ecriture (Write): Une écriture est un type de mouvement de donnée qui déplace un groupe de donnée se trouvant à l'intérieur d'un processus fonctionnel vers une partie de stockage. Une écriture fait référence aux attributs de données d'un groupe de données et en modifie les valeurs. Sur le plan fonctionnel, une écriture envoie des données manipulées par le processus fonctionnel auquel elle appartient vers l'extérieur de la frontière du logiciel, dans la partie stockage.

REM0 : La méthode COSMIC-FFP FSM défini un mouvement de donnée comme un CBF

REM1 : Une écriture inclut une certaines manipulations de données associées nécessairement pour réaliser l écriture voir

REM2 : Dans COSMIC-FFP, une entrée inclut aussi les manipulations (ex. : validation de la donnée entrante) associées.

REM3 : Une manipulation de donnée est tout ce qui se passe sur une donnée différent d'un mouvement.

REM4 : Une lecture inclut certaines manipulations de données associées nécessairement pour accomplir la lecture.

REM5 : Une sortie inclut aussi les manipulations de données associées (ex. formatage et routage associés à la donnée afin d'être sortie).

[47] De "ISO/IEC 14143-1:1998 – Génie logiciel – Software measurement – Functional size measurement – Partie 1: Definition of concepts", clause 3.1.

DÉFINITION DE CONCEPTS UML

Diagramme de séquences: Un diagramme de séquences présente explicitement une séquence de communications/messages entre « ClassifierRoles » (acteurs, éléments structuraux). Les diagrammes de séquences peuvent être utilisés pour décrire les interactions entre acteurs et cas d'utilisation.

N.B. : Nous considérons dans notre analyse qu'un diagramme de séquences est **une** (car il peut y en avoir plusieurs pour le même cas d'utilisation) séquence d'interactions représentatives d'un cas d'utilisation.

RAPPROCHEMENT

De ce qui précède, il apparaît que ce que présente un diagramme de séquences UML se rapproche d'une séquence de mouvements de données COSMIC-FFP. En effet, les messages mentionnées dans un diagramme de séquences UML peuvent être exprimés en termes de mouvements élémentaires de données (entrées, sorties, lectures ou écritures COSMIC-FFP) et donc de mouvements de données COSMIC-FFP.

EVENEMENT DÉCLENCHEUR COSMIC-FFP

DÉFINITION DE CONCEPTS COSMIC-FFP

Événement déclencheur (triggering event) : Un événement déclencheur est un événement qui se produit en dehors de la frontière du logiciel mesuré et initie un ou plusieurs processus fonctionnels. Dans un ensemble de FUR, chaque type d'événement qui déclenche un processus fonctionnel est indivisible pour cet ensemble de FUR.

REM 1 : Les horloges et les événements temporels peuvent être des événements déclencheurs.

REM 2 : Un événement est soit passé soit pas, il est instantané.

DÉFINITION DE CONCEPTS UML

Nous n'avons pas trouvé d'équivalent strict en UML du concept de déclencheur (un déclencheur COSMIC-FFP se rapproche beaucoup plus du premier message, de la première communication d'un diagramme de séquences UML)

RAPPROCHEMENT

Nous faisons le rapprochement entre un déclencheur COSMIC-FFP et le premier message, la première communication d'un diagramme de séquences UML

CLASSE & GROUPE DE DONNÉES COSMIC-FFP

DÉFINITION DE CONCEPTS COSMIC-FFP

Groupe de données [type] (Data group type): Un groupe de données est un ensemble (non ordonné) non vide et non redondant d'attributs. Chaque attribut décrit un aspect complémentaire du même objet d'intérêt. *(un objet d'intérêt est identifié du point de vue des requis fonctionnels de l'utilisateur et pourrait représenter un objet (ou une partie d'un objet) que l'on trouve dans le monde réel, ou encore un objet conceptuel (ou une partie dun objet conceptuel), nécessaire pour supporter les opérations propres d'un logiciel)*

Groupe de données persistant : C'est une qualité indiquant la persistance d'un groupe de données dans le contexte du FUR. Trois types de persistance sont définis : transitoire (pour la durée de vie du processus seulement), courte (au-delà de la durée du processus fonctionnel tant que le logiciel est opérationnel) et durable (au-delà de la durée des opérations du logiciel).

DÉFINITION DE CONCEPTS UML

Classe : Il s'agit d'une description d'un ensemble d'objets ayant une structure, un comportement et des associations similaires.

RAPPROCHEMENT

Si nous nous limitons uniquement aux attributs (sans considérer les méthodes, ni les associations), il apparaît qu'une classe UML devient tout simplement un ensemble (non ordonné) non vide et non redondant d'attributs de données, et donc un groupe de données COSMIC-FFP.

ATTRIBUT UML & ATTRIBUT COSMIC-FFP

DÉFINITION DE CONCEPT COSMIC-FFP

Attribut [type] (Data attribute type): Un attribut est la plus petite parcelle d'information codée, dans un groupe de données, portant une signification dans la perspective fonctionnelle des besoins de l'utilisateur.

DÉFINITION DE CONCEPTS UML

Attribut : Il s'agit d'une propriété nommée d'un objet.

RAPPROCHEMENT

De ce qui précède il apparaît qu'un attribut de données COSMIC-FFP se rapproche bien d'un attribut UML. En effet, les attributs UML ne sont que de l'information codée et significative du point de vue des requis fonctionnels des utilisateurs *(ils sont d'ailleurs obtenus à partir des requis fonctionnels des utilisateurs)*.

DIAGRAMME DES CAS D'UTILISATION & FRONTIÈRE COSMIC-FFP D'UN LOGICIEL

DÉFINITION DE CONCEPTS COSMIC-FFP

Environnement d'opération du logiciel (Operating environment (software)): L'environnement d'opération du logiciel est l'ensemble des logiciels opérant de manière concurrente sur un système informatique spécifié.

Frontière (Boundary): La frontière d'un morceau de logiciel est une interface conceptuelle entre le morceau de logiciel et ses utilisateurs.

REM : La frontière d'un logiciel est la ligne conceptuelle séparant ce logiciel de l'environnement dans lequel il opère, tel que perçu par ses utilisateurs d'un point de vue externe. La frontière permet à la personne qui mesure de distinguer, sans ambiguïté, ce qui est inclus dans le logiciel de ce qui fait partie de l'environnement dans lequel fonctionne ce logiciel.

DÉFINITION DE CONCEPTS UML

Diagramme des cas d'utilisation : Il s'agit d'un graphe d'acteurs, un ensemble de cas d'utilisation à l'intérieur de la frontière du système, d'associations de communication (participation) entre acteurs et cas d'utilisation, et de généralisations parmi les cas d'utilisation.

RAPPROCHEMENT

De ce qui précède, il apparaît que le diagramme des cas d'utilisation définit de manière intrinsèque la frontière d'un système.

ACTEUR UML & UTILISATEUR (OU USAGER) COSMIC-FFP

DÉFINITION DE CONCEPTS COSMIC-FFP

Utilisateur COSMIC-FFP (COSMIC-FFP User): Il s'agit d'un être humain, d'un autre logiciel ou d'un dispositif, qui interagit avec le logiciel mesuré.

DÉFINITION DE CONCEPTS UML

Acteur (UML) : Il s'agit d'un ensemble cohérent de rôles qu'un utilisateur d'une entité d'un système peut jouer lorsqu'il interagit avec l'entité. Un acteur peut être considéré comme jouant un rôle spécifique lorsqu'il communique avec un cas d'utilisation.

RAPPROCHEMENT

De ce qui précède, il apparaît que la notion d'utilisateur COSMIC-FFP est très proche de la notion d'acteur UML.

COUCHE COSMIC-FFP D'UN LOGICIEL

DÉFINITION DE CONCEPTS COSMIC-FFP

Couche (Layer): Une couche est le résultat du partitionnement fonctionnel de l'environnement du logiciel où tous les processus fonctionnels inclus montrent un haut degré de cohésion et s'exécutent au même niveau d'abstraction.

Dans un environnement logiciel à plusieurs couches, celles-ci interagissent les unes avec les autres via leurs processus fonctionnels respectifs. Ces interactions sont hiérarchiques par nature; lorsque considérée par paire, une couche est « cliente » de l'autre. Une couche « cliente » utilise les services fonctionnels fournis par des couches subordonnées. Les éléments du logiciel dans la même couche peuvent aussi échanger des données. Ce type d'échange de données est habituellement appelé un échange de données « peer-to-peer ».

RAPPROCHEMENT

Le seul rapprochement que nous pouvons faire c'est de considérer une couche comme un système à part entière ou un composant. Il est bien entendu qu'un système est susceptible d'interagir avec d'autres systèmes, tout comme une couche peut bel et bien interagir avec un autre couche (tel que mentionné dans les spécifications COSMIC-FFP).

ETUDE DE CAS : APPLICATION DE CONTRÔLE D'ACCÈS A UN BÂTIMENT [MULLER 00]

FRONTIÈRE

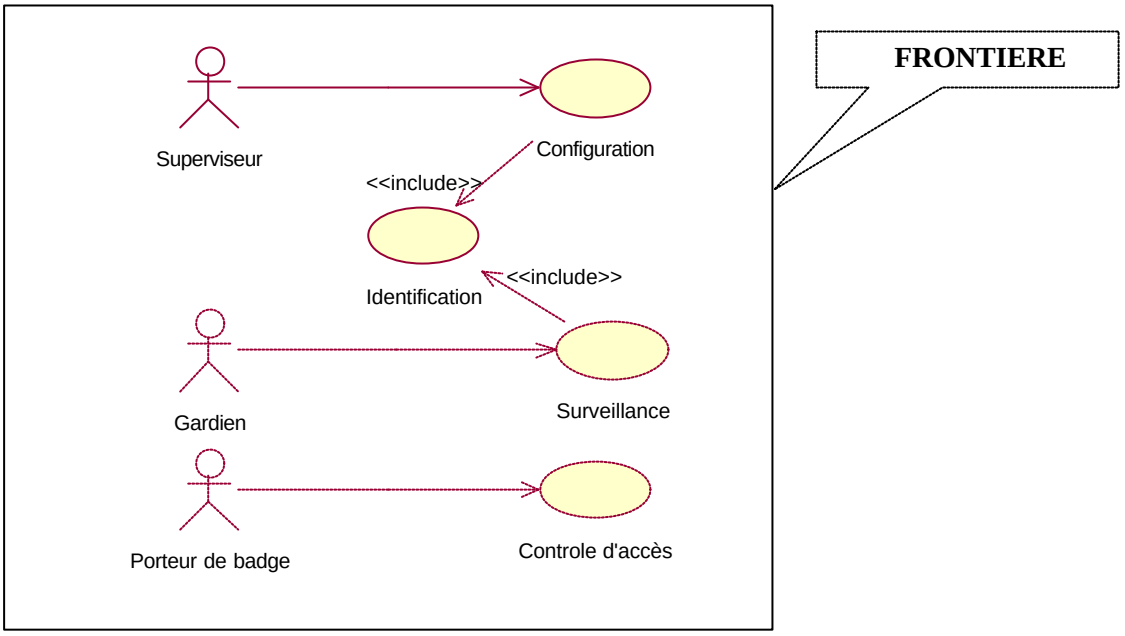


Figure 25 : Diagramme des cas d'utilisation pour le système de contrôle d'accès

Question : Dans la mesure où un cas d'utilisation est susceptible de comporter plusieurs scénarios et que les scénarios sont relativement indépendants les uns par rapport aux autres ⇒ Ne serait-il pas plus judicieux de considérer chacun des scénarios comme un processus fonctionnel ? *(auquel cas, il faudrait faire un rapprochement entre processus fonctionnel COSMIC-FFP et scénario UML)*

Examinons un peu plus en détail les cas d'utilisation identifiés ci-dessus (Figure 25). Ils comportent pour certains plusieurs scénarios, lesquels scénarios sont relativement indépendant les uns par rapport aux autres.

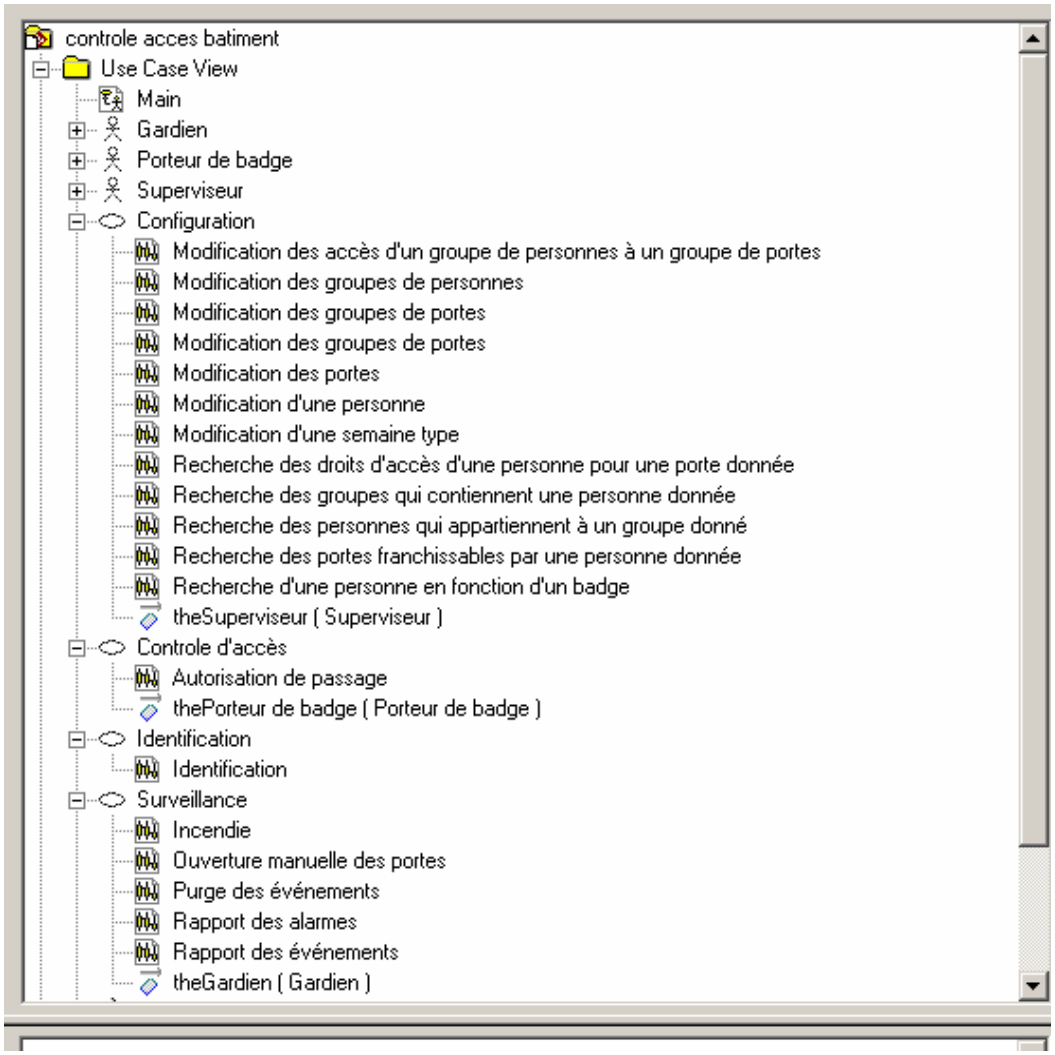


Figure 26: Détails de s cas d'utilisation pour le système de contrôle d'accès

COUCHES

Le système ne comporte qu'une seule couche.

GROUPES DE DONNÉES

Ils sont obtenus à partir du diagramme des classes du domaine (lui-même obtenu à partir des diagrammes de collaboration).

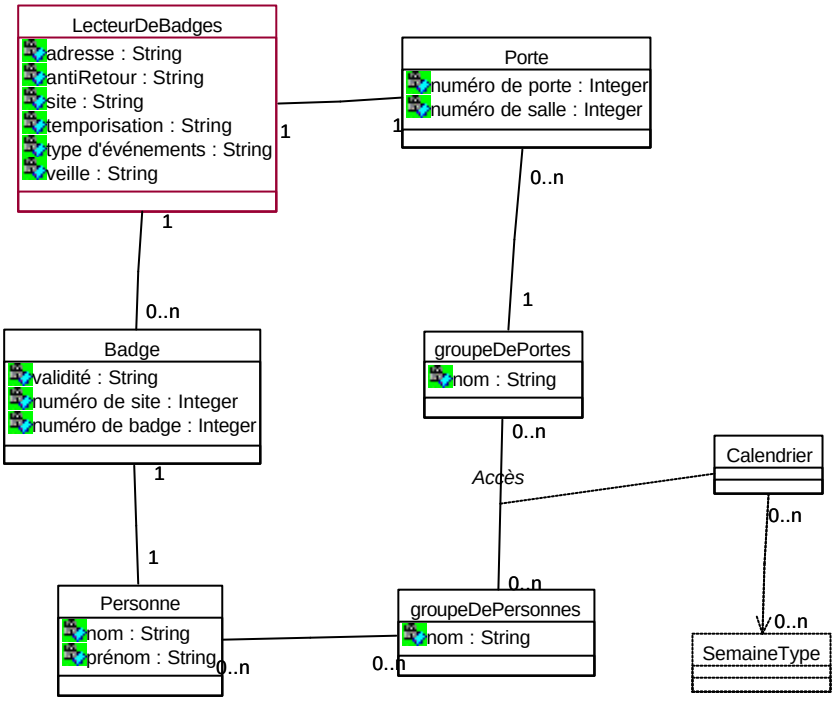


Figure 27: Diagramme de classes du domaine pour le système de contrôle d'accès

- Lecteur de badges
- Badge
- Porte
- Personne
- Groupe de portes
- Groupe de personnes
- Semaine type
- Calendrier

ATTRIBUTS DE DONNÉES

- Lecteur de badges (adresse, anti-retour, site, temporisation, type d'événements, veille)
- Badge (validité, numéro de site, numéro de badge,…)
- Porte (numéro de porte, numéro de salle,…)

- Personne (prénom, nom,…)
- Groupe de portes (nom,…)
- Groupe de personnes (nom,…)
- Semaine type (…)

PROCESSUS FONCTIONNELS

(en supposant qu'ils correspondent aux différents cas d'utilisation recensés).

- Configuration
- Identification
- Contrôle d'accès
- Surveillance

UTILISATEURS

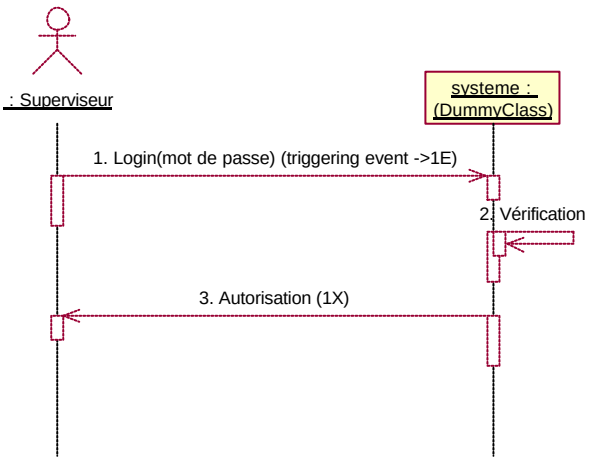
Ils correspondent aux différents acteurs recensés :

- Superviseur
- Gardien
- Utilisateur (porteur de badge)

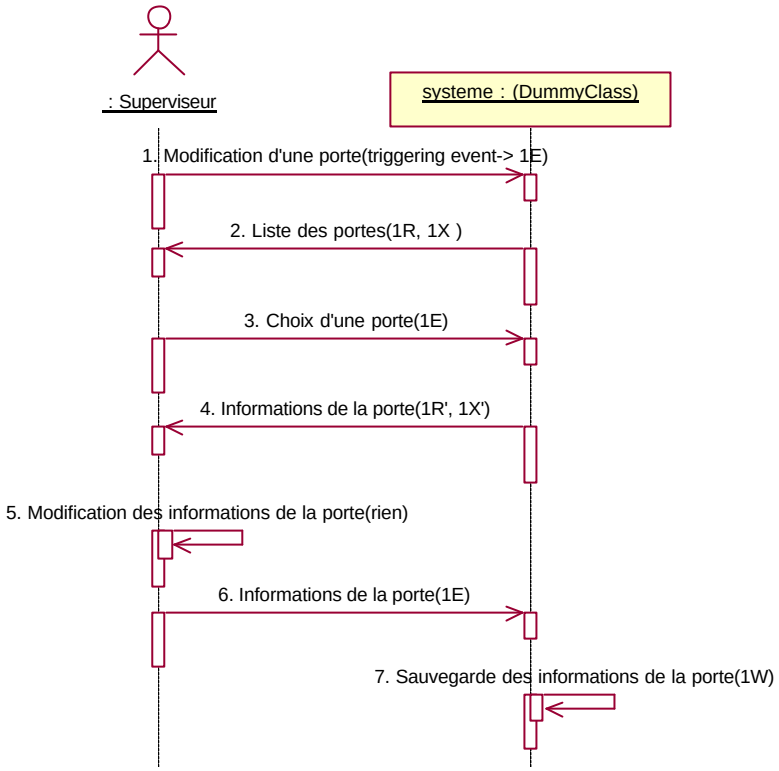
SOUS PROCESSUS

Processus Configuration

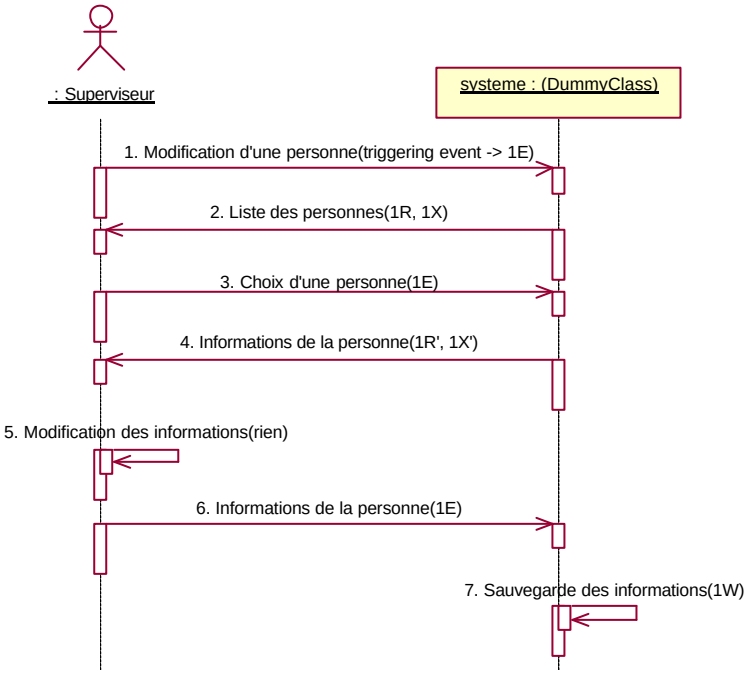
Scénario ***Identification***



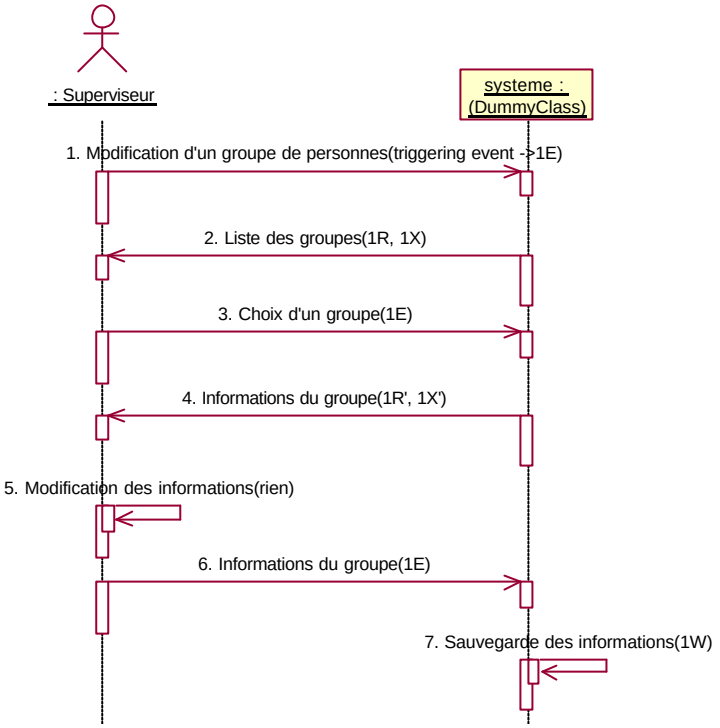
*Scénario **Modification des portes***



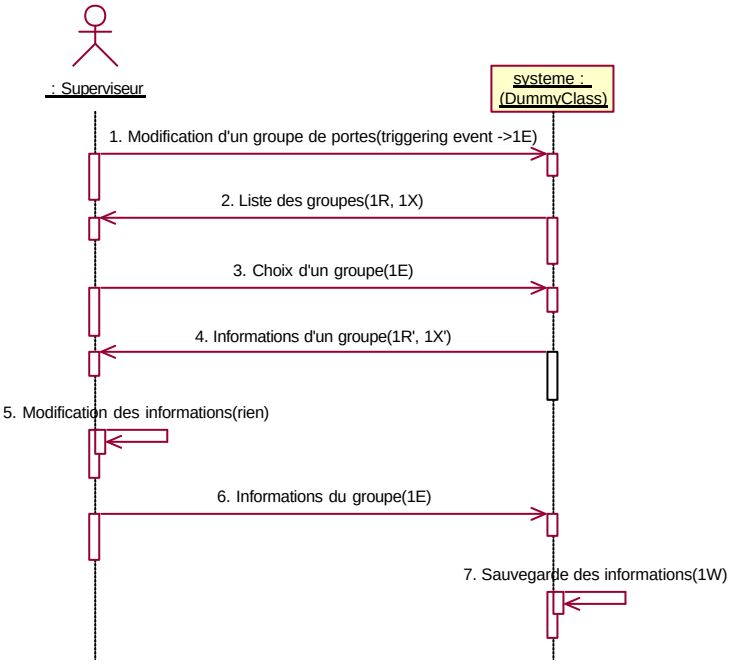
*Scénario **Modification d'une personne***



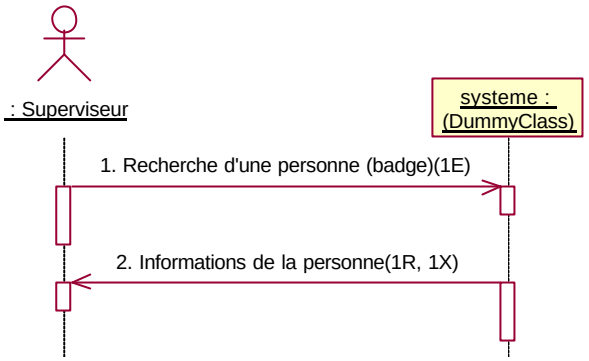
Scénario ***Modification des groupes de personnes***



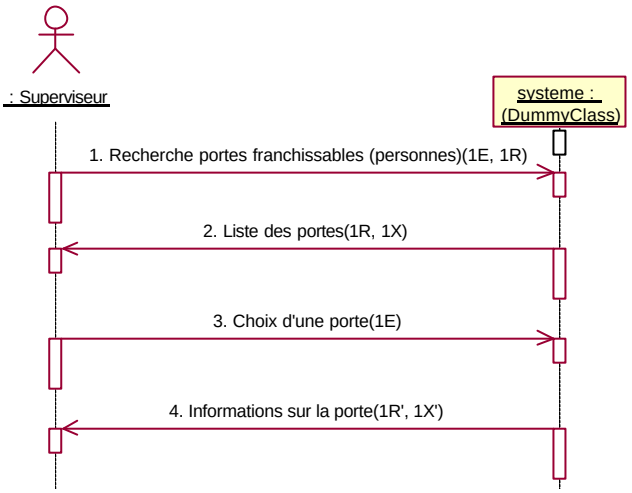
Scénario ***Modification des groupes de portes***



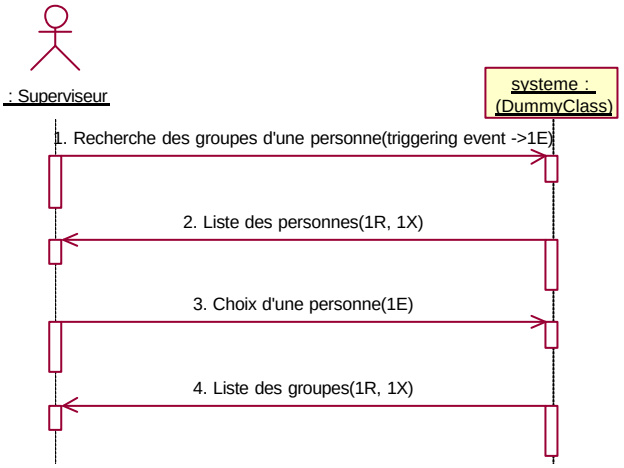
Scénario ***Recherche d'une personne en fonction d'un badge***



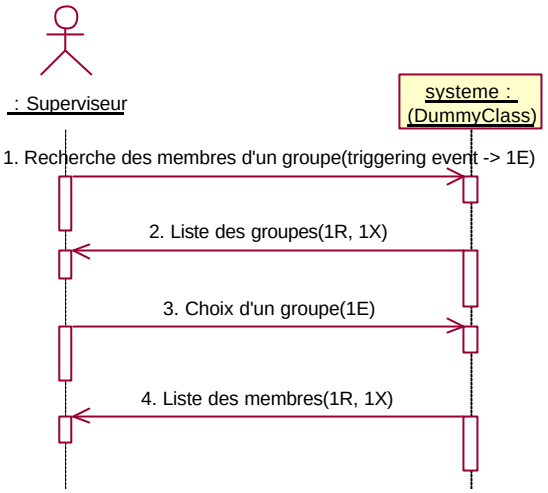
Scénario ***Recherche des portes franchissables par une personne donnée***



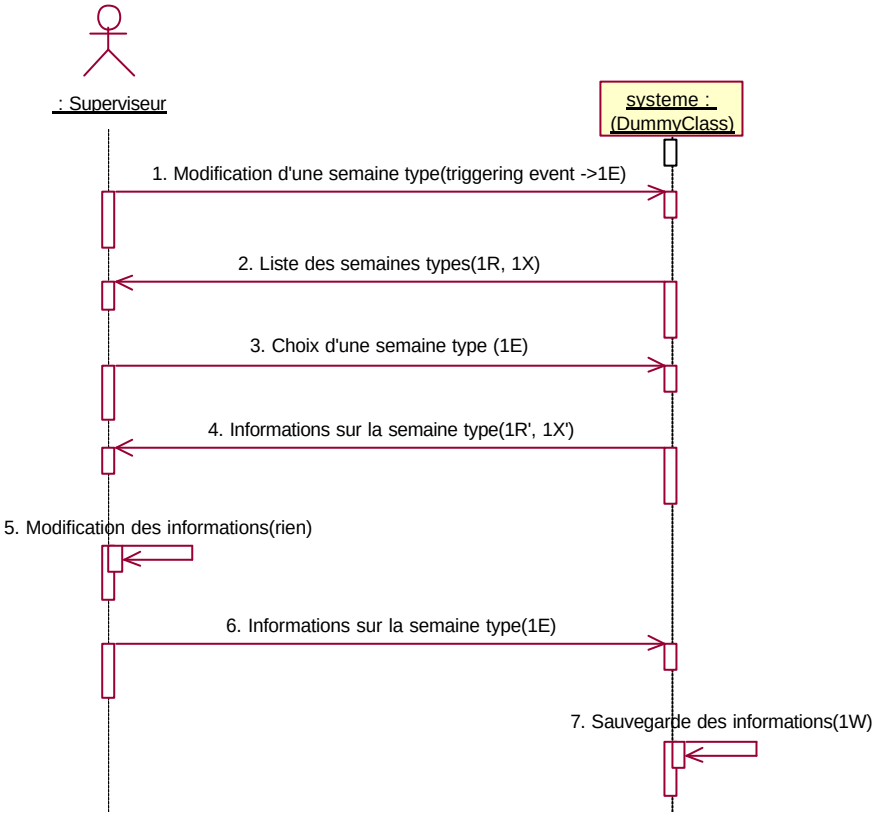
Scénario ***Recherche des groupes qui contiennent une personne donnée***



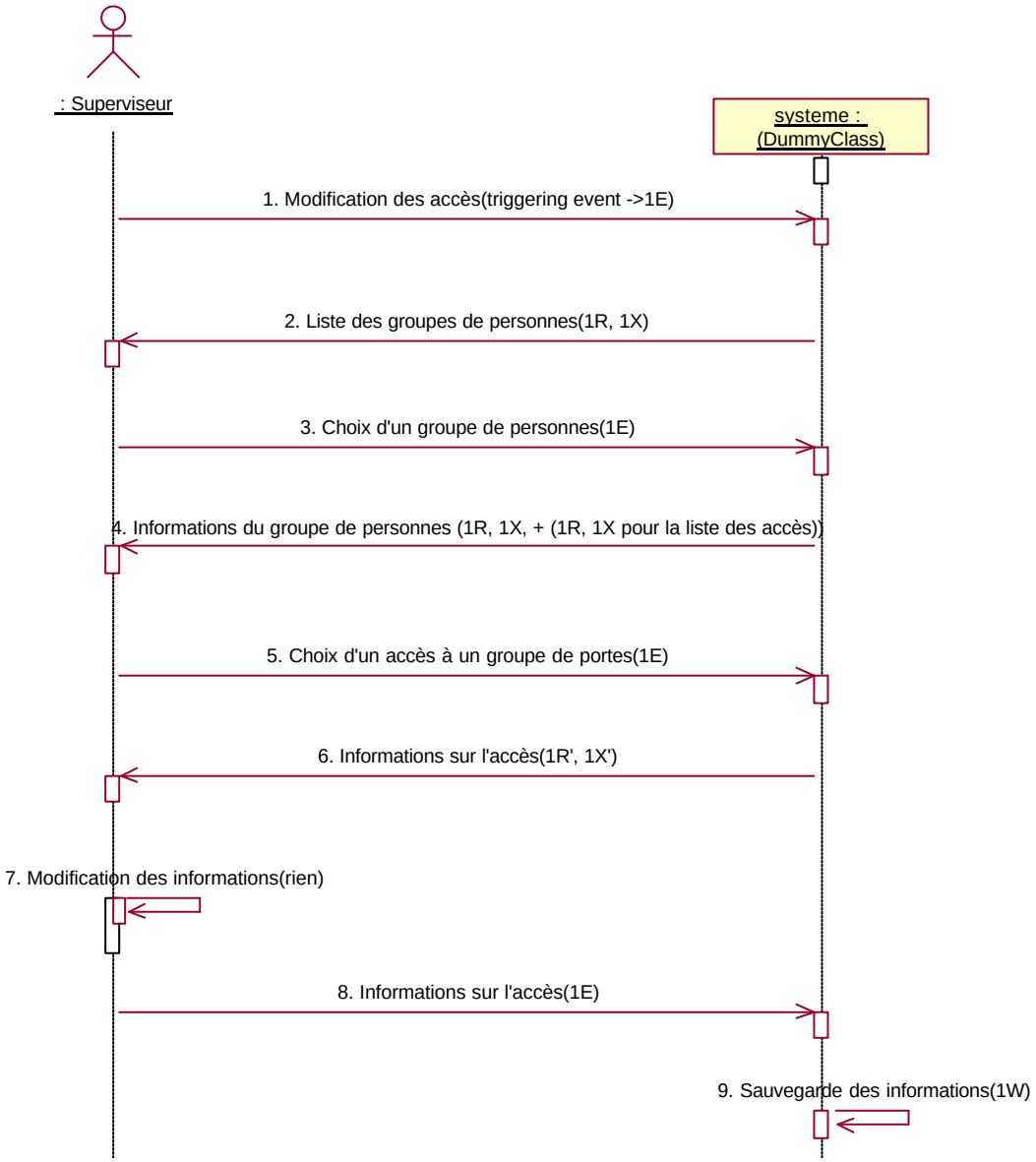
Scénario ***Recherche des personnes qui appartiennent a un groupe donné***



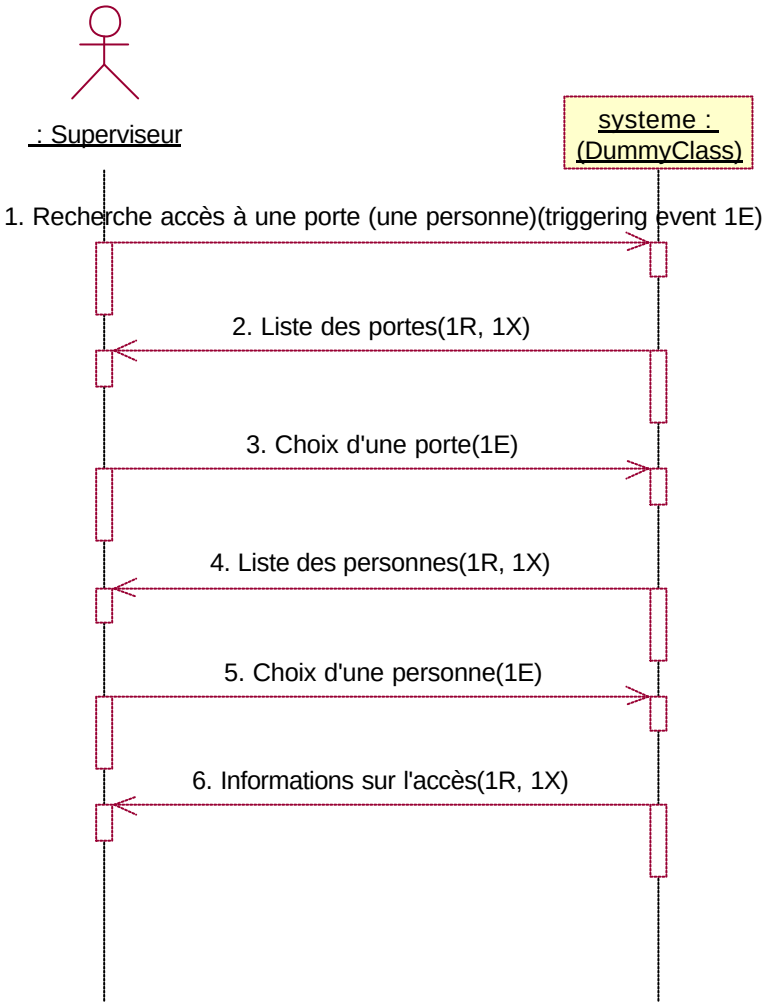
*Scénario **Modification d'une semaine type***



Scénario ***Modification des accès d'un groupe de personnes à un groupe de portes***



Scénario ***Recherche des droits d'accès d'une personne pour une porte donnée***

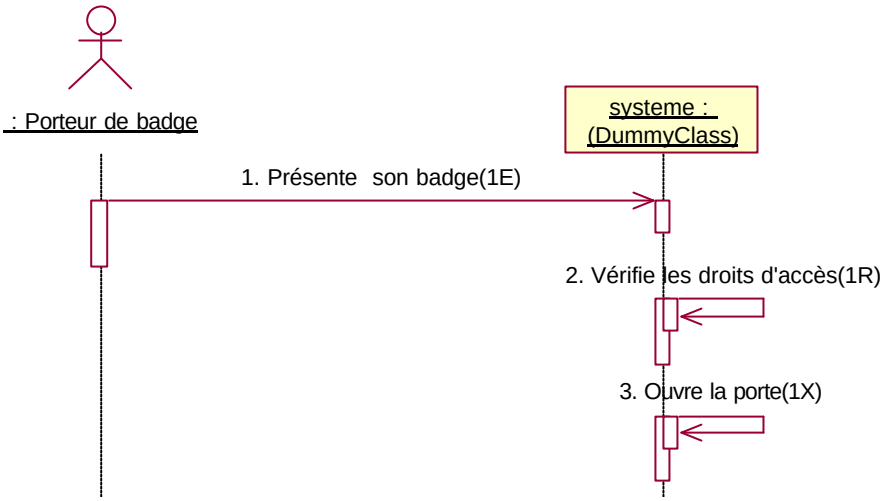


Processus Identification

*Scénario **Identification** (idem que précédemment)*

Processus Contrôle d accès

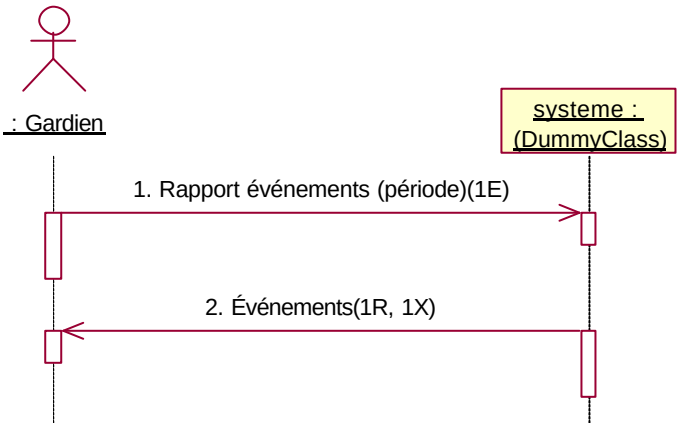
*Scénario **Autorisation de passage***



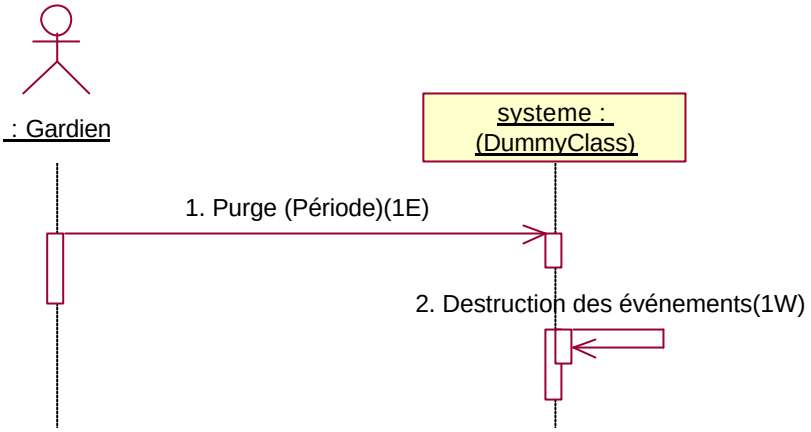
Processus Surveillance

*Scénario **Identification** (idem que plus haut)*

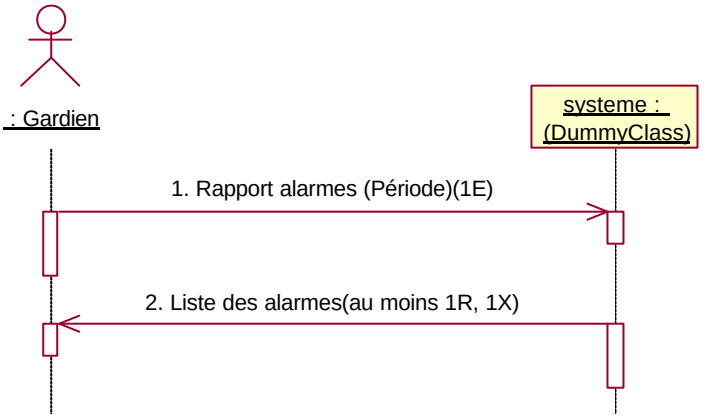
*Scénario **Rapport des événements***



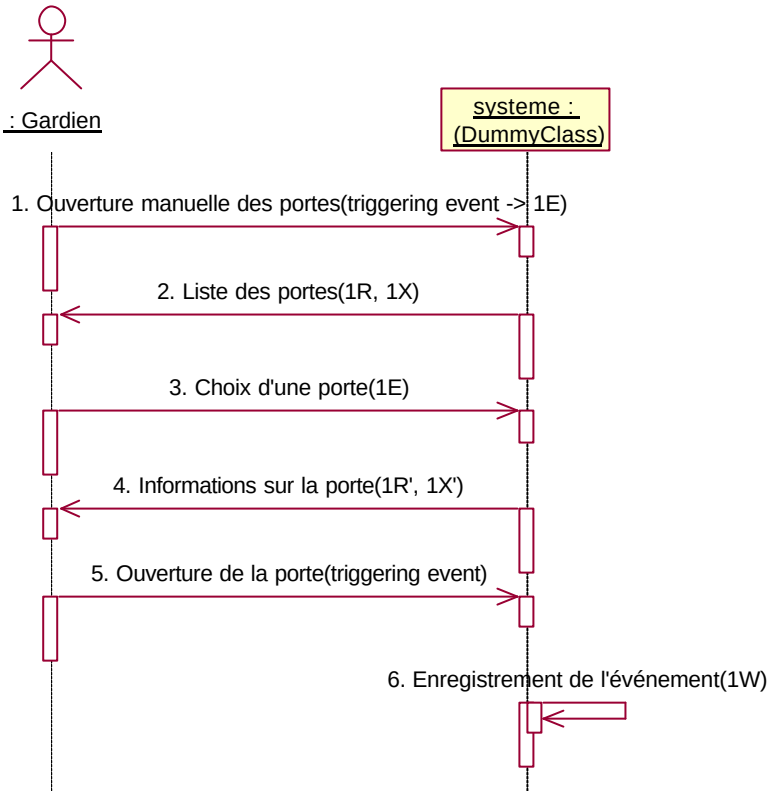
Scénario ***Purge des événements***



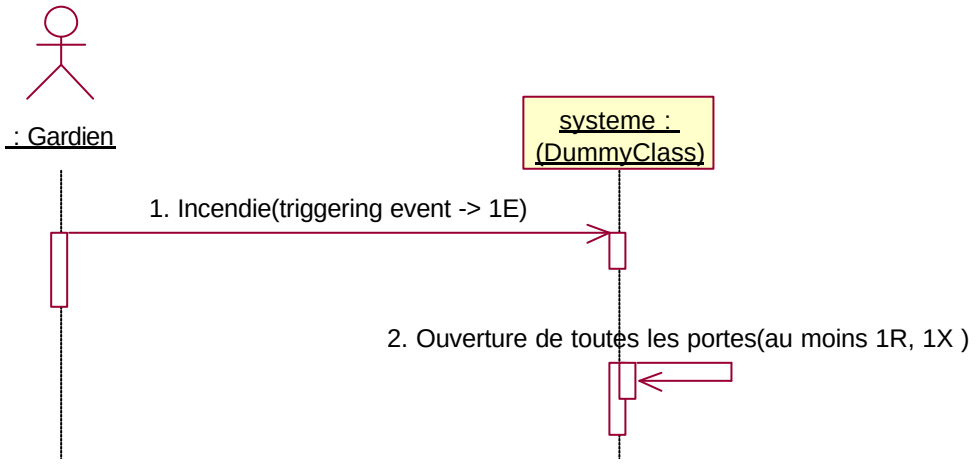
Scénario ***Rapport des alarmes***



Scénario ***Ouverture manuelle des portes***



Scénario ***Incendie***



MODÈLE COSMIC-FFP DU SYSTÈME (EN FAISANT LE RAPPROCHEMENT CAS D'UTILISATION/PROCESSUS FONCTIONNEL)

Layers	**Functional processes**	**Data groups**								**Sub processes**			
		Badge	***Porte***	***Personne***	***Groupe de portes***	***Groupe de personnes***	***Lecteur de badge***	***Semaine type***	***Message ou Signal***	**ENTRY (E)**	**EXIT (X)**	**READ (R)**	**WRITE (W)**
LAYER 1	Configuration	E	R, X, E, W	R, X, E, W	R, X, E, W	R, X, E, W		R, X, E, W	X	6	6	5	5
	Identification			R, X, E					X	1	2	1	
	Contrôle d'accès	R, E, & (X)[48]							X	1	2	1	
	Surveillance		R, X					R, X, E, W	X	1	3	2	1
									TOTAL – layer 1	9	13	9	6
									GRAND TOTAL	**37 CFSUs[49]**			

[48] pour le message d'autorisation

[49] CFSU = COSMIC Functional Size Unit

MODÈLE COSMIC-FFP DU SYSTÈME (EN FAISANT LE RAPPROCHEMENT SCÉNARIO UML/PROCESSUS FONCTIONNEL COSMIC-FFP)

Layers	**Functional processes**	**Data groups**								**Sub processes**			
LAYER 1		***Badge***	***Porte***	***Personne***	***Groupe de portes***	***Groupe de personnes***	***Lecteur de badge***	***Semaine type***	***Message ou Signal***	**ENTRY (E)**	**EXIT (X)**	**READ (R)**	**WRITE (W)**
	(1)			E, R					X	1	1	1	
	(2)		R, X, E, E, W						E, X	3	2	1	1
	(3)			R, X, E, E, W					E, X	3	2	1	1
	(4)					R, X, E, E, W			E, X	3	2	1	1
	(5)				R, X	R, X, E		R, X, E, E, W	E, X	4	4	3	1
	(6)							R, X, E, E, W	E, X	3	2	1	1
	(7)				R, X, E, E, W				E, X	3	2	1	1
	(8)	E, R		R, X						1	1	2	
	(9)		R, X, E	E, R						2	1	2	
	(10)			R, X, E		R, X			E	2	2	2	
	(11)			R, X		R, X, E			E	2	2	2	
	(12)		R, X, E	R, X, E				R, X	E	3	3	3	
	(13)	E, R	R					R	X	1	1	3	

	(14)							E, R, X		1	1	1	
	(15)							E, W	X	1	1		1
	(16)							E, R, X		1	1	1	
	(17)		R, X, E					W	E, X	2	2	1	1
	(18)		R						E, X	1	1	1	
TOTAL – layer 1										**37**	**31**	**27**	**8**
GRAND TOTAL										**103 CFSUs**			

Correspondance entre les numéros du tableau précédent et les processus fonctionnels

(1) Identification
(2) Modification des portes
(3) Modification des personnes
(4) Modification des groupes de personnes
(5) Modification des accès d'un groupe de personnes à un groupe de portes
(6) Modification d'une semaine type
(7) Modification des groupes de portes
(8) Recherche d'une personne en fonction d'un badge
(9) Recherche des portes franchissables par une personne donnée
(10) Recherche des groupes qui contiennent une personne donnée
(11) Recherche des personnes qui appartiennent à un groupe donné
(12) Affichage des droits d'accès d'une personne pour une porte donnée
(13) Autorisation de passage
(14) Rapport des événements
(15) Purge des événements
(16) Rapport des alarmes
(17) Ouverture manuelle des portes
(18) Incendi

ANNEXE F: LE PROTOTYPE : DÉTAILS TECHNIQUES

SPECIFICATIONS FONCTIONNELES

Le système MetricXpert compte quatre composants principaux : Un composant de mesure, un composant de spécifications, un composant ontologique et un composant d'administration. Dans la suite nous décrivons **brièvement** chacun des composants. Tous les détails seront fournis dans un document de spécifications et exigences logicielles qui sera produit ultérieurement dans le cadre de la construction d'un outil commercial.

COMPOSANTE DE SPÉCIFICATIONS

DIAGRAMME DES CAS D'UTILISATION

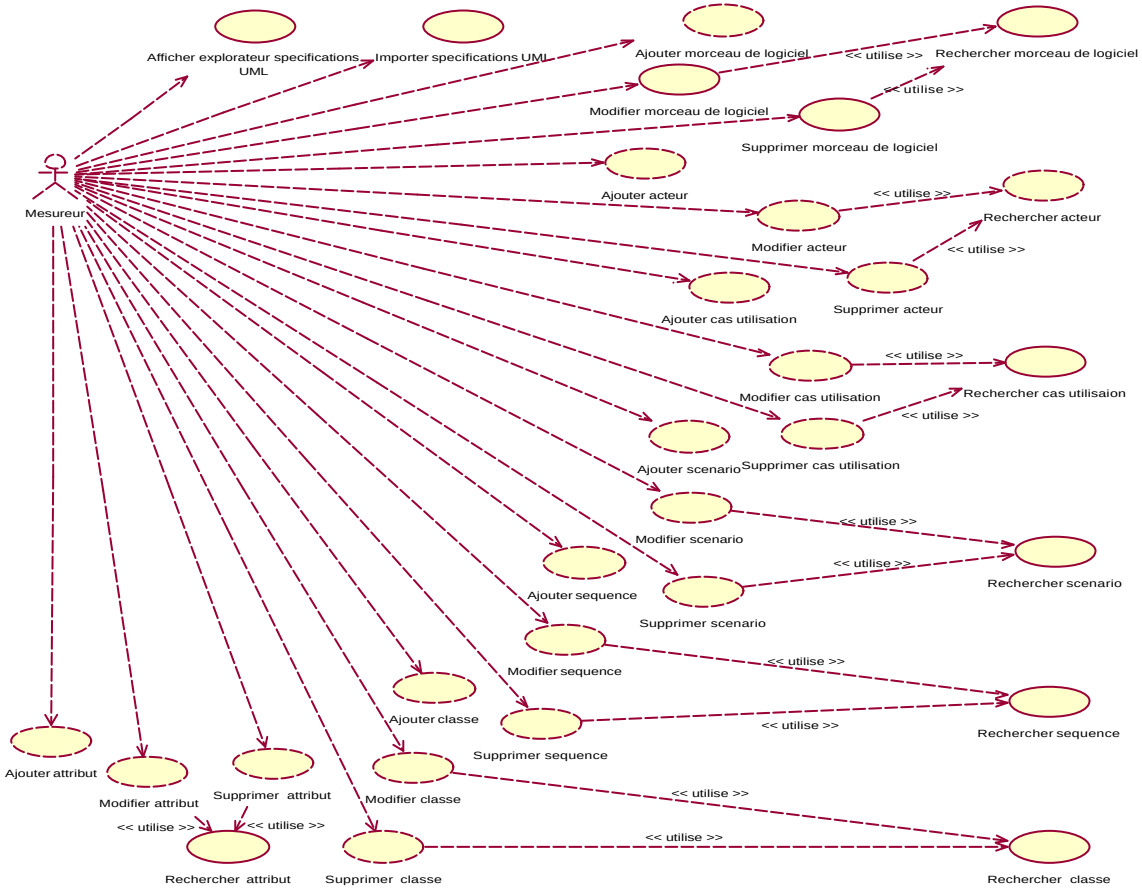


Figure 28 : Diagramme des cas d'utilisation pour la composante de spécifications

L.1 Cas d'utilisation 1 : Importer spécifications UML

No 1	**Titre : Importer spécifications UML**

Acteurs : **Mesureur**

Objectifs **: Permettre au mesureur d'importer des spécifications UML d'un morceau de logiciel dont on veut déterminer la taille fonctionnelle (spécifications produites à l'aide d'un outil CASE)**

Pré-conditions : **Le mesureur accède à l'interface d'importation des spécifications via le menu « Spécifications » (sous-menu «Importer spécifications UML ») de l'interface principale de MetricXpert ou à l'explorateur de spécifications UML du système (puis au menu contextuel obtenu grâce au bouton droit de la souris).**

Post-conditions : **Les éléments de spécifications UML du morceau de logiciel considéré jugés pertinents pour la mesure sont stockés dans la base de connaissances du système et affichés dans la fenêtre principale du système.**

Exceptions : **Aucune**

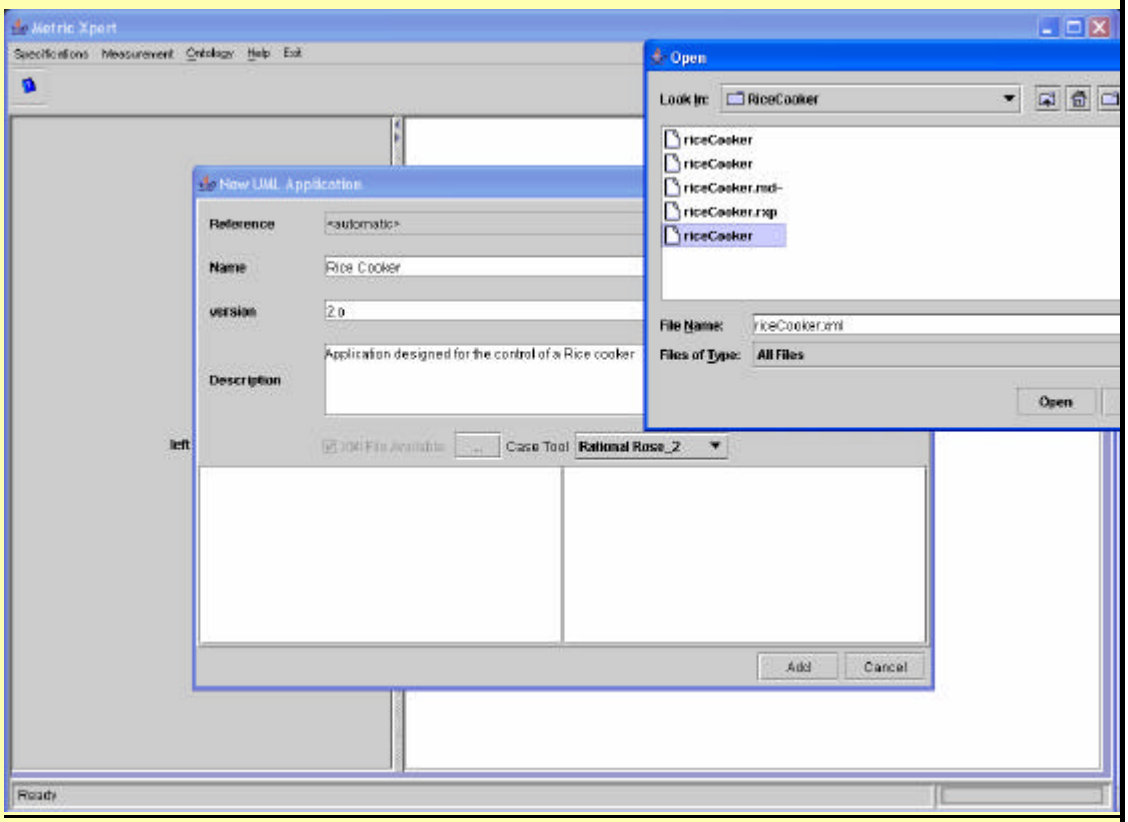
Description sommaire **: Le mesureur dispose d'un fichier '.xmi' ou '.dtd' ou '.xml' contenant les spécifications UML d'un morceau de logiciel dont il veut déterminer la taille fonctionnelle. A partir de l'interface d'importation, un explorateur de fichier lui est présenté pour la sélection du fichier de spécifications. Du fichier sélectionné, sont extraits les éléments de spécifications jugées pertinents pour la mesure. Les éléments extraits, ainsi que les informations générales (nom, description, version) relatives au morceau de logiciel sont stockés dans la base de connaissances du système et affichés dans la fenêtre principale du système.**

Liste des données en entrée : **fichier '.xmi', '.dtd', ou '.xml'; nom, version et description sommaire du morceau de logiciel dont on veut importer les spécifications; outil CASE dans lequel les spécifications ont été produites.**

Liste des données en sortie **: UML_MORCEAU_DE_LOGICIEL_T, UML_CAS_UTILISATION_T, UML_SCENARIO_T, UML_SEQUENCE_T, UML_CLASSE_T, UML_ATTRIBUT_T**

Notes et messages : **…**

Interface graphique :



L.2 Cas d'utilisation 2 : Afficher explorateur spécifications UML

No 2	**Titre : Afficher explorateur spécifications UML**

Acteurs : **Mesureur**

Objectifs : **Permettre de présenter au mesureur une vue arborescente des éléments de spécifications UML des morceaux de logiciels qui sont contenus dans la base de connaissances du système.**

Pré-conditions : **Le mesureur accède à l'interface principale de MetricXpert. L'explorateur des modèles COSMIC-FFP est fermé.**

Post-conditions **: Une vue arborescente des éléments de spécifications UML de logiciels qui sont jugés pertinents pour la mesure est présentée dans la fenêtre principale du système.**

Exceptions : **aucune.**

Description sommaire **: Le mesureur accède au menu « Spécifications » de l'interface principale du système et sélectionne le sous menu « Afficher explorateur spécifications UML ». Le système extrait alors de la base de connaissances les éléments de spécifications UML de logiciels qui y**

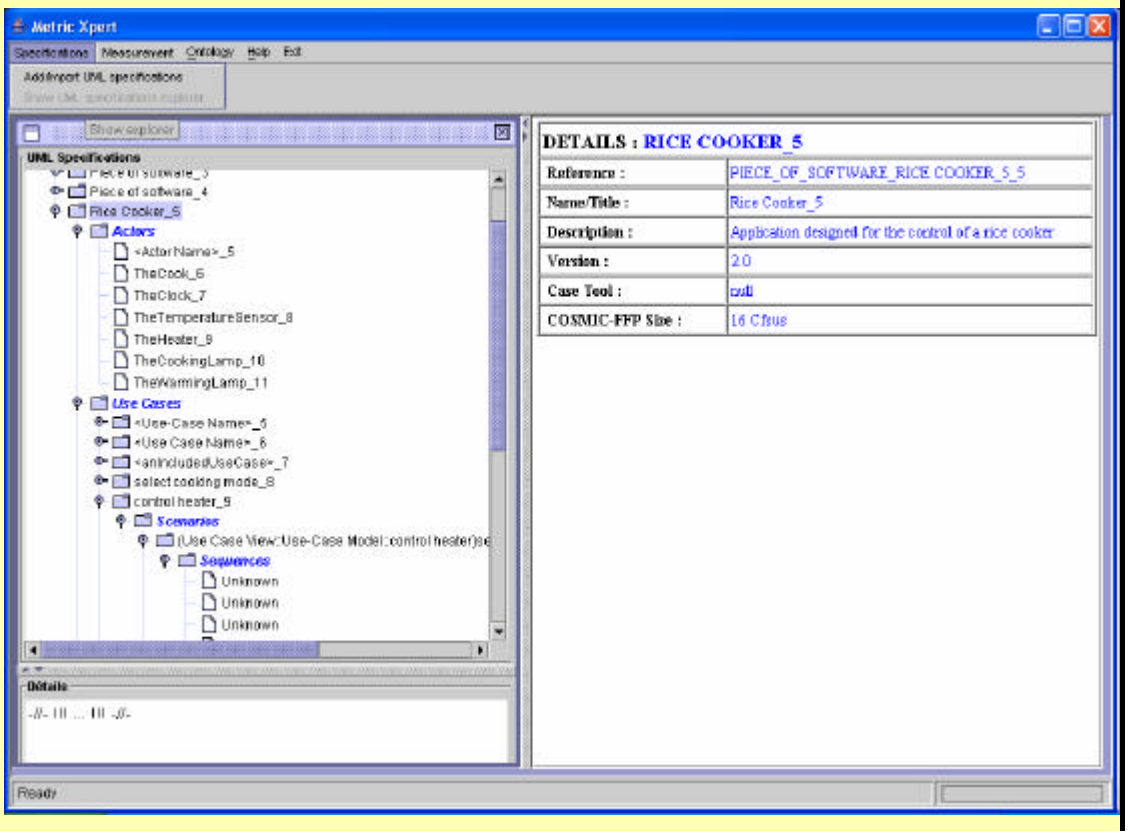
sont stockés et en présente une vue arborescente dans fenêtre principale du système.

Liste des données en entrée **: aucune**

Liste des données en sortie : **UML_MORCEAU_DE_LOGICIEL_T, UML_CAS_UTILISATION_T, UML_SCENARIO_T, UML_SEQUENCE_T, UML_CLASSE_T, UML_ATTRIBUT_T**

Notes et messages **: …**

Interface graphique :



L.3 Cas d'utilisation 3 : Ajouter morceau de logiciel UML

No 1	**Titre : Ajouter morceau de logiciel UML**

Acteurs : **Mesureur**

Objectifs : **Permettre au mesureur d'ajouter dans la base de connaissances du système, des informations générales (nom, description, version) relatives à un nouveau morceau de logiciel dont les éléments de spécifications UML seront insérés un à un dans la base.**

Pré-conditions **: Le mesureur accède à l'explorateur de spécifications UML du système (puis au menu contextuel obtenu grâce au bouton droit de la souris).**

Post-conditions : **Les informations générales (nom, description, version) relatives à un nouveau morceau de logiciel sont stockées dans la base de connaissances du système.**

Exceptions : **aucune.**

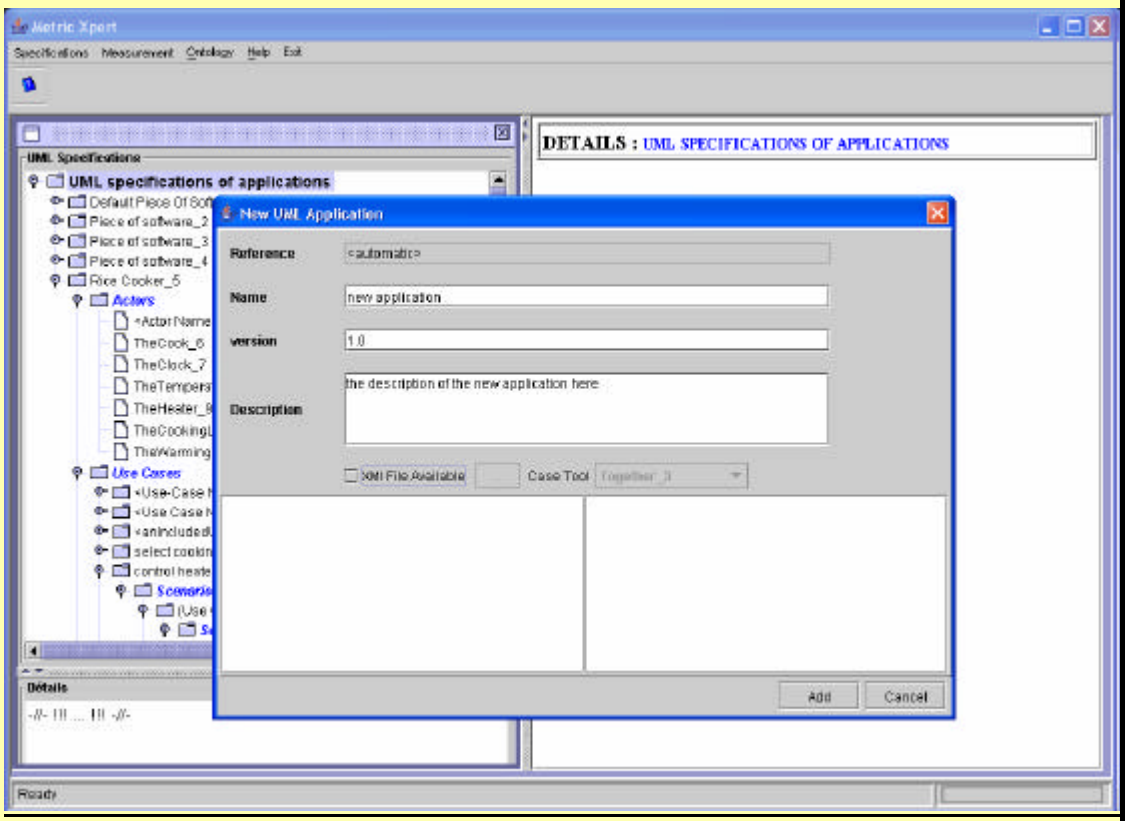
Description sommaire : **Le mesureur accède au menu contextuel obtenu grâce au bouton droit de la souris lorsque la celle-ci est positionnée sur le nœud « Spécifications UML des applications » de l'explorateur des spécifications UML du système. L'interface d'ajout de morceaux de logiciel lui est présentée. Il inscrit les informations générales (nom, description, version) relatives au nouveau morceau de logiciel dans les espaces appropriés prévus et clique sur le bouton «ok ». Les informations inscrites sont alors validées puis stockées dans la base de connaissances du système.**

Liste des données en entrée : **nom, version et description sommaire de la nouvelle application.**

Liste des données en sortie : **_MORCEAU_DE_LOGICIEL_T**

Notes et messages **: …**

Interface graphique :

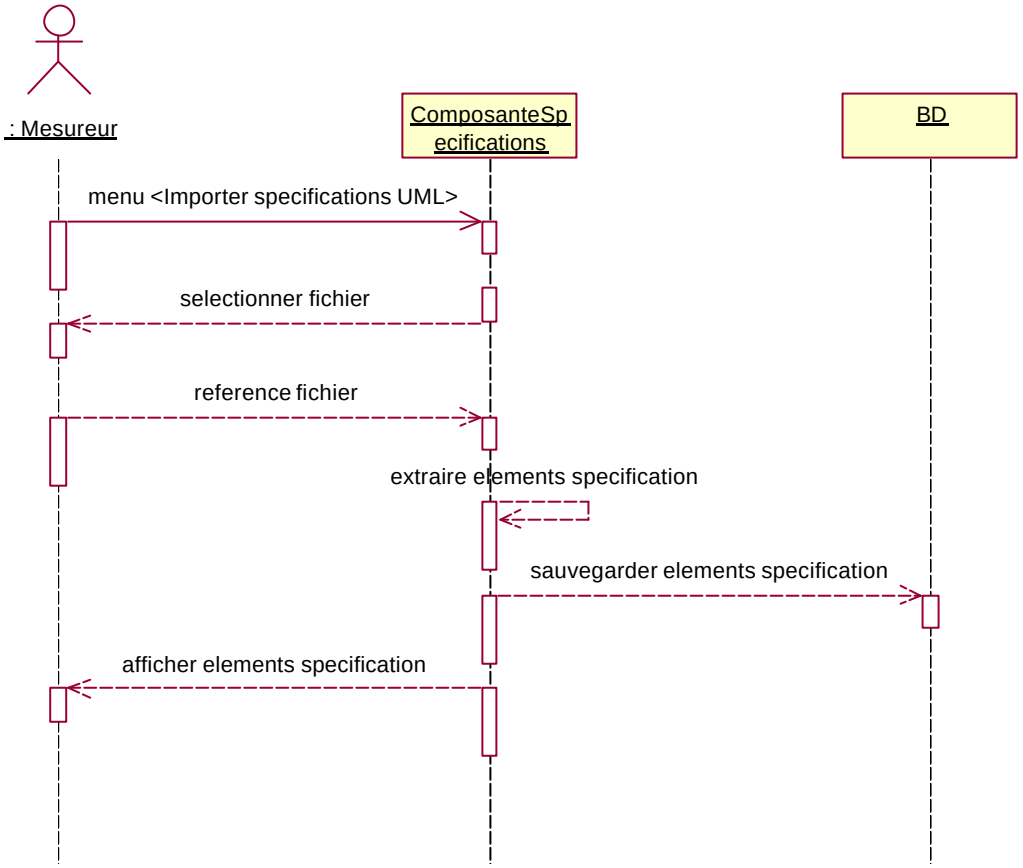


Nota Benne : Le détail des autres cas d'utilisation du composant de spécifications sera fourni dans un document de spécifications et exigences logicielles qui sera produit ultérieurement dans le cadre de la construction d'un outil commercial.

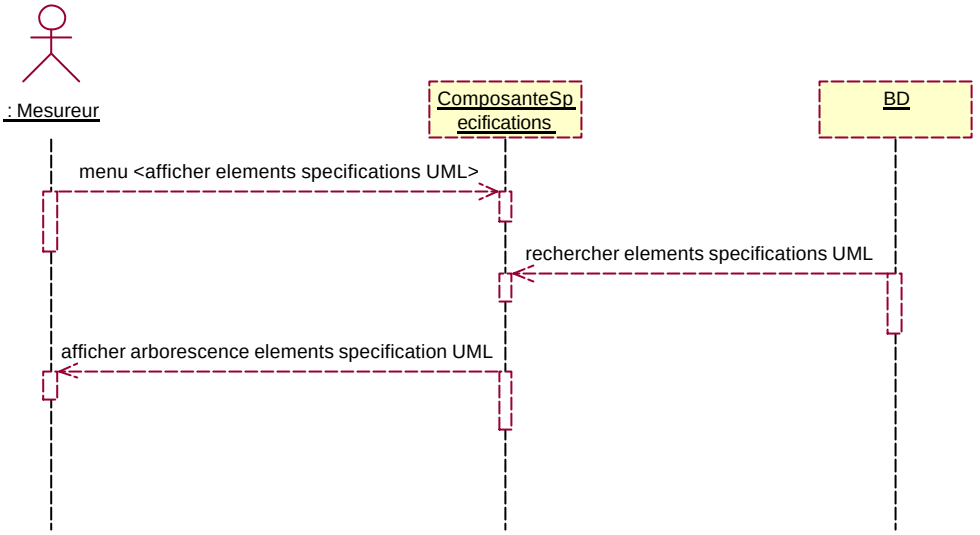
Nous présentons dans la section suivante les diagrammes de séquences associées aux cas d'utilisation que nous avons examinés plus haut.

DIAGRAMMES DE SEQUENCES

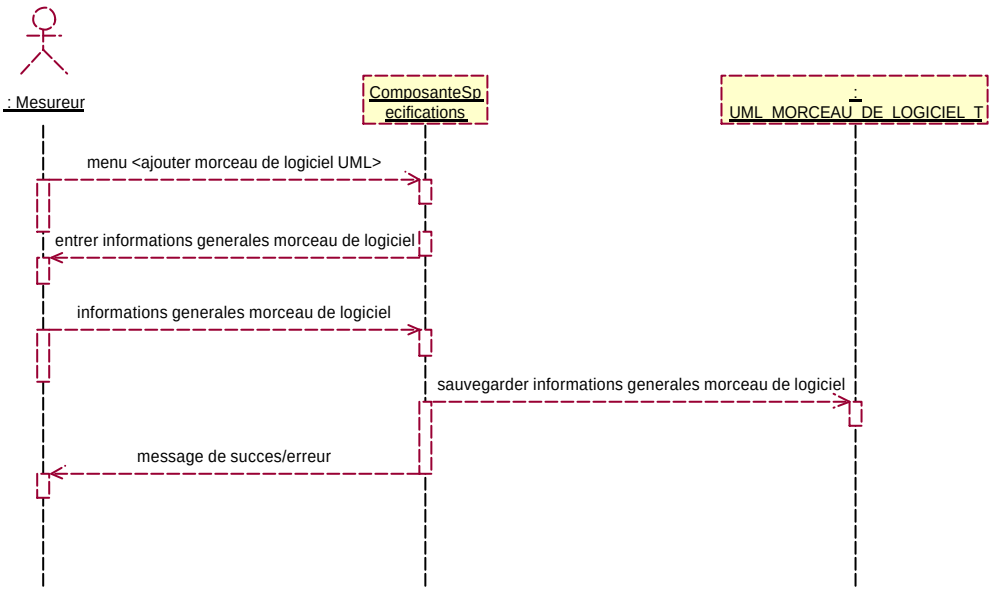
L.1 Importer spécifications UML



L.2 Afficher explorateur spécifications UML



L.3 Ajouter morceau de logiciel UML



COMPOSANTE DE MESURE

DIAGRAMME DES CAS D'UTILISATION

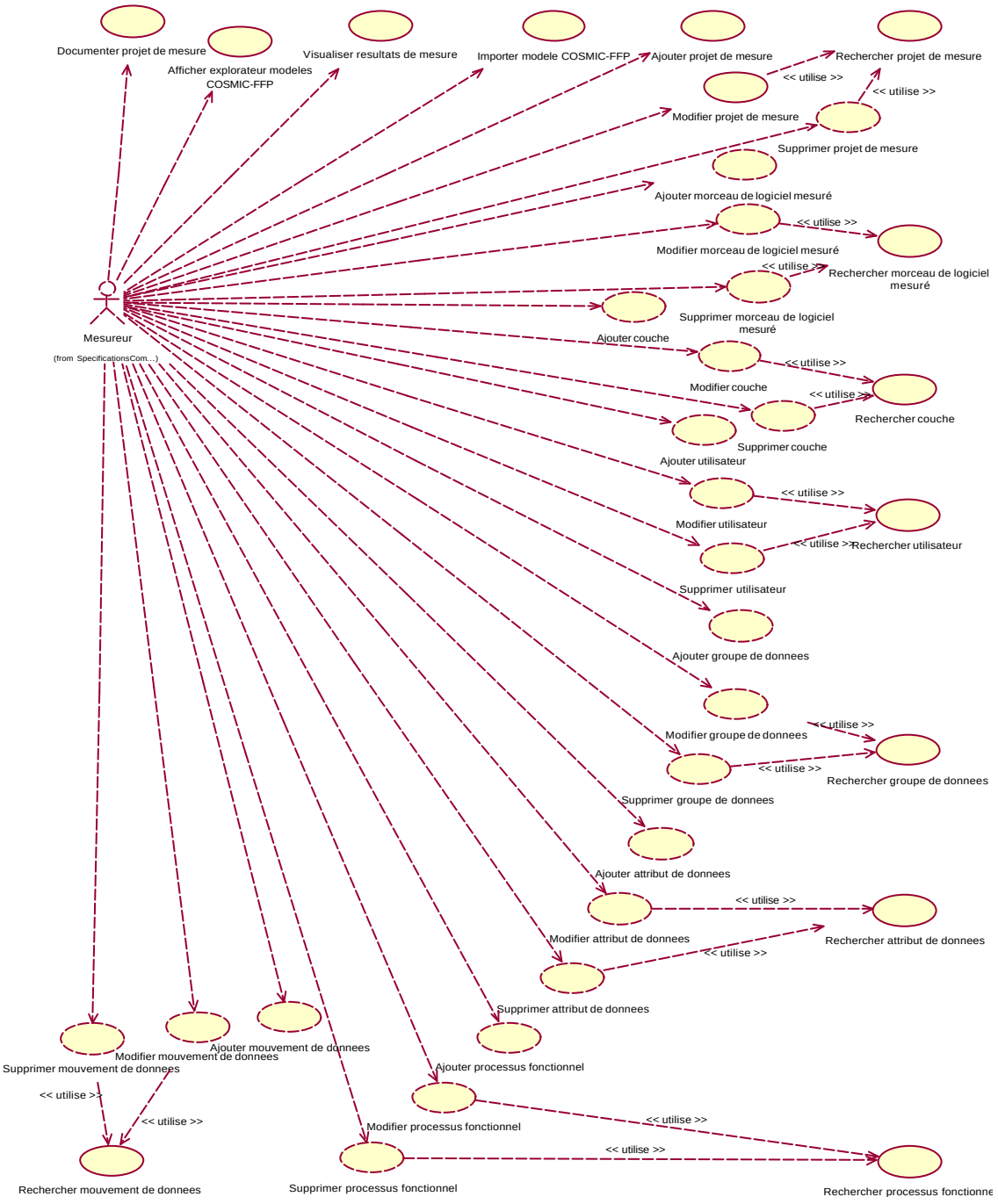


Figure 29 : Diagramme des cas d'utilisation pour la composante de mesure

L.1 Cas d'utilisation 1 : Afficher explorateur modèles COSMIC-FFP

No 2	**Titre : Afficher explorateur modèles COSMIC-FFP**

Acteurs : **Mesureur**

Objectifs : **Permettre de présenter au mesureur une vue arborescente des modèles COSMIC-FFP des applications dont les éléments de spécifications UML sont stockés dans la base de connaissances du système.**

Pré-conditions : **Le mesureur accède à l'interface principale de MetricXpert. L'explorateur des spécifications UML est fermé.**

Post-conditions : **Une vue arborescente des modèles COSMIC-FFP des applications dont les éléments de spécifications UML sont stockés dans la base de connaissances du système, est présentée dans la fenêtre principale du système.**

Exceptions : **aucune.**

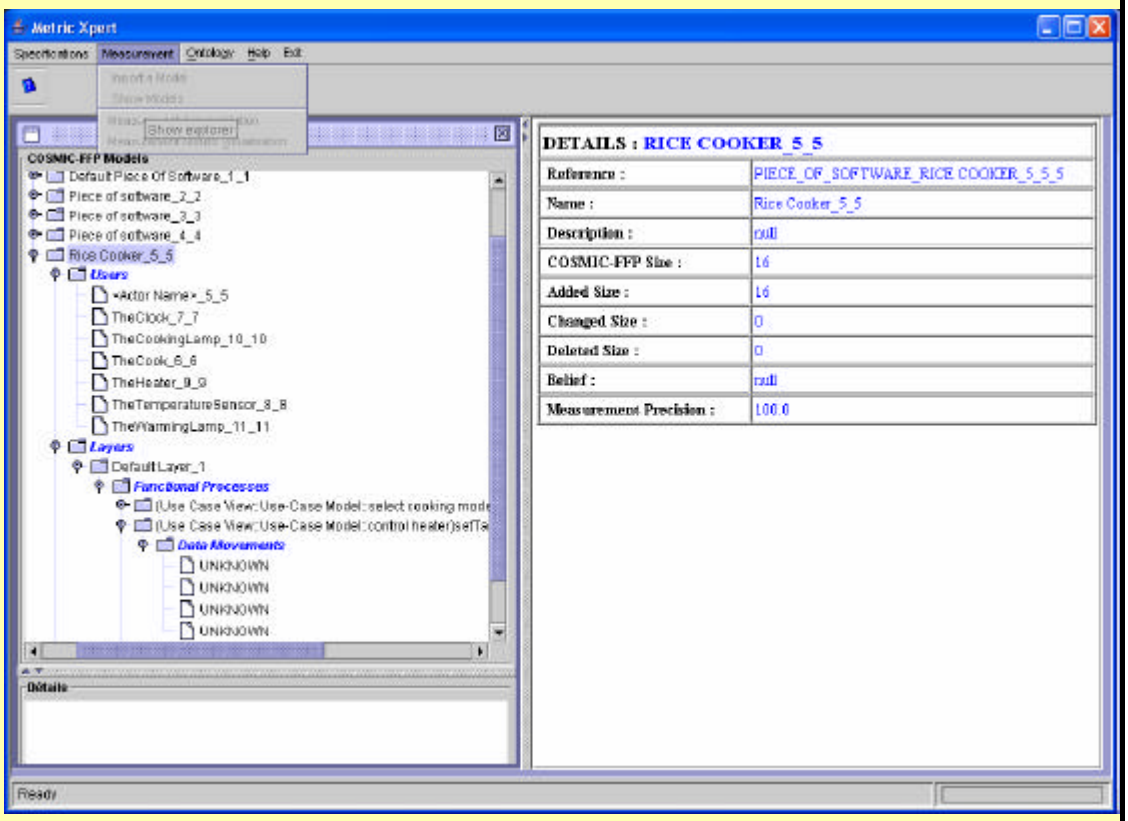
Description sommaire : **Le mesureur accède au menu «Mesure » de l'interface principale du système et sélectionne le sous-menu « Afficher explorateur modèles COSMIC-FFP ». Le système extrait alors de la base de connaissances les modèles COSMIC-FFP des applications dont les éléments de spécifications UML sont stockés et en présente une vue arborescente dans fenêtre principale du système (les modèles sont regroupés par projet de mesure COSMIC-FFP).**

Liste des données en entrée **:** **aucune**

Liste des données en sortie :
COSMIC_PROJET_DE_MESURE_T,COSMIC_MORCEAU_DE_LOGICIEL_T, COSMIC_COUCHE_T, COSMIC_EVENEMENT_T, COSMIC_PROCESSUS_COUCHE_LOGICIEL_T, COSMIC_UTILISATEUR_T, COSMIC_GROUPE_DE_DONNEES_T, COSMIC_ATTRIBUT_T, COSMIC_PROCESSUS_FONCTIONNEL_T, COSMIC_MOUVEMENT_DE_DONNEES_T.

Notes et messages : **…**

Interface graphique :



L.2 Cas d'utilisation 2 : Ajouter Projet de mesure COSMIC-FFP

No 1	**Titre : Ajouter Projet de mesure COSMIC-FFP**

Acteurs : **Mesureur**

Objectifs : **Permettre au mesureur d'ajouter dans la base de connaissances du système, des informations générales (titre, description) relatives à un nouveau projet de mesure COSMIC-FFP (un projet de mesure inclut un ou plusieurs morceaux de logiciel mesurés).**

Pré-conditions : **Le mesureur accède à l'explorateur des modèles COSMIC-FFP (puis au menu contextuel obtenu grâce au bouton droit de la souris).**

Post-conditions : **Les informations générales (titre, description) relatives à un nouveau projet de mesure COSMIC-FFP sont stockées dans la base de connaissances du système.**

Exceptions : **Aucune.**

Description sommaire : **Le mesureur accède au menu contextuel obtenu grâce au bouton droit de la souris lorsque la celle-ci est positionnée sur le nœud « Modèles COSMIC-FFP » de l'explorateur des modèles COSMIC-FFP. L'interface d'ajout de projets lui est présentée. Il**

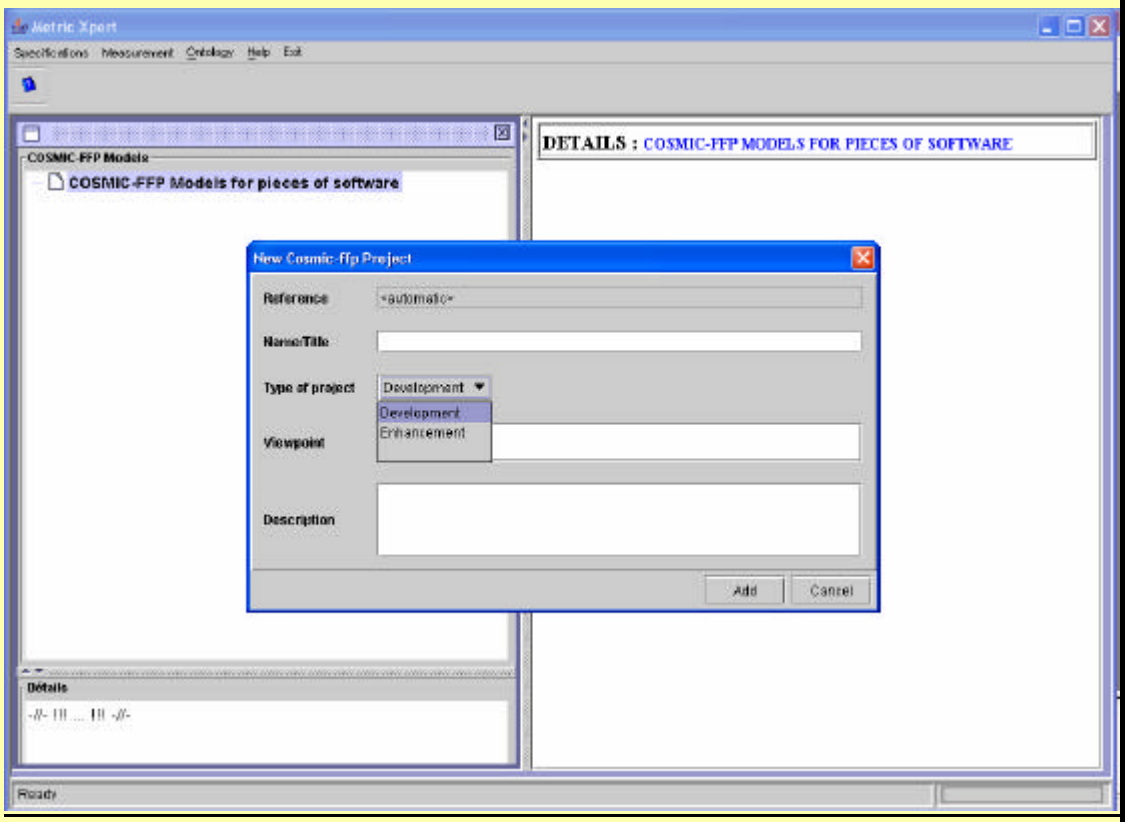
inscrit les informations générales (titre, description) relatives au nouveau projet de mesure COSMIC-FFP dans les espaces appropriés prévus et clique sur le bouton « ok ». Les informations inscrites sont alors validées puis stockées dans la base de connaissances du système.

Liste des données en entrée **: titre, perspective de la mesure, type de projet et description sommaire du nouveau projet de mesure COSMIC-FFP.**

Liste des données en sortie **: COSMIC_PROJET_DE_MESURE_T**

Notes et messages : **…**

Interface graphique **:**

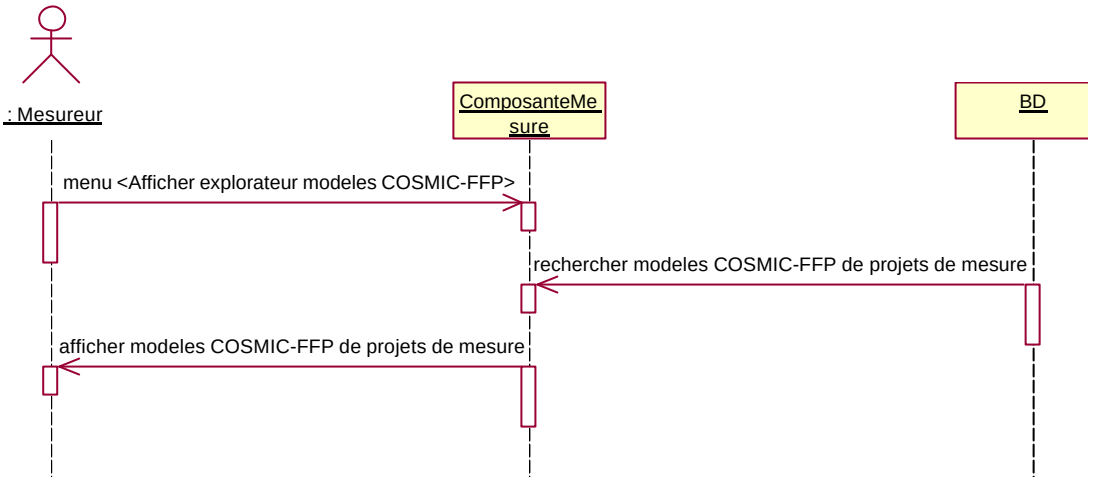


Nota Benne : Le détail des autres cas d'utilisation de la composante de mesure sera fourni dans un document de spécifications et exigences logicielles qui sera produit ultérieurement dans le cadre de la construction d'un outil commercial.

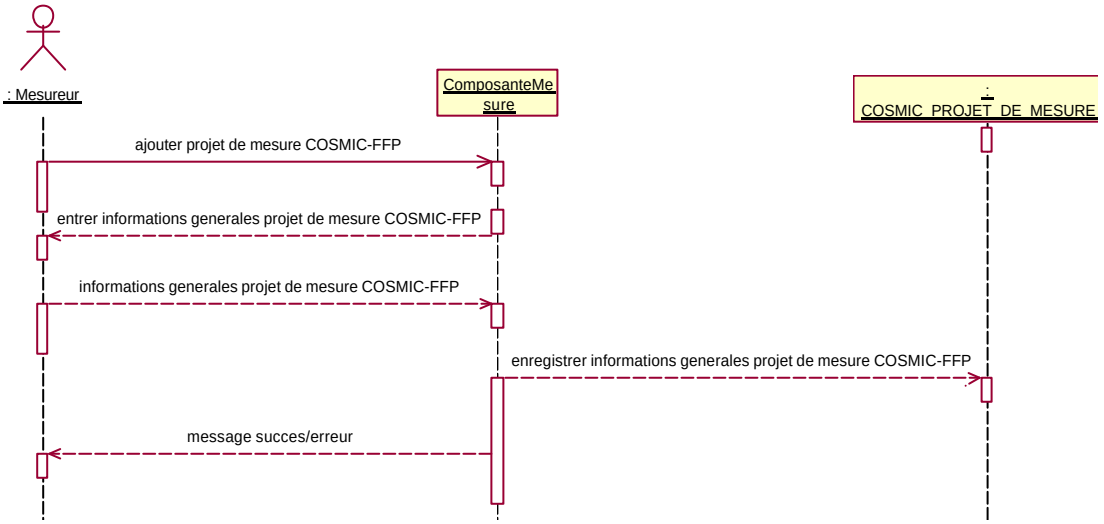
Nous présentons dans la section suivante les diagrammes de séquences associées aux cas d'utilisation que nous avons examinés plus haut.

DIAGRAMMES DE SEQUENCE

L.1 Afficher explorateur modèles COSMIC-FFP



L.2 Ajouter projet de mesure COSMIC-FFP



COMPOSANTE ONTOLOGIQUE

DIAGRAMME DES CAS D'UTILISATION

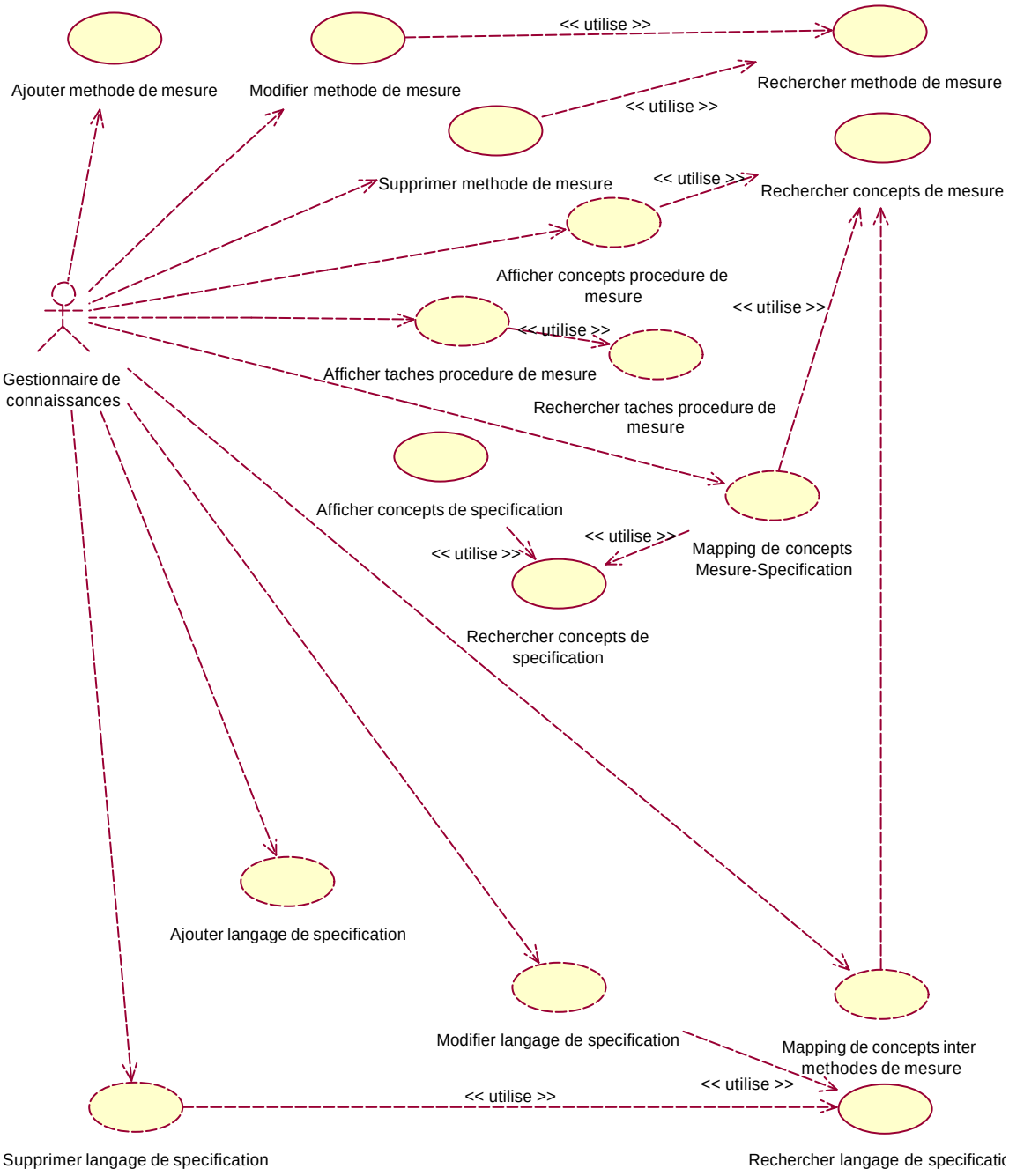


Figure 30 : Diagramme des cas d'utilisation pour la composante ontologique

L.1 « Mapping » de concepts Mesure-Spécifications

No 1	**Titre : « Mapping » de concepts Mesure - Spécifications**

Acteurs : **Gestionnaire de connaissances**

Objectifs : **Permettre au gestionnaire de connaissances d'établir des correspondances entre concepts de mesure et concepts de spécifications. Ces correspondances sont traduites au niveau de la base de connaissances du système par des «triggers » (déclencheurs) et des tables de correspondances.**

Pré-conditions : **Le gestionnaire de connaissances a les droits requis pour une telle opération. Il s'est connecté au système avec le profil «gestionnaire de connaissances ».**

Post-conditions : **Une nouvelle correspondance est établie entre un concept de mesure et un concept de spécification. La correspondance est matérialisée au niveau de la base de connaissances du système par la création de 2 «triggers » (déclencheurs) et une table de correspondances.**

Exceptions : **Aucune.**

Description sommaire : **Le gestionnaire de connaissances accède à l'interface de « mapping » via le menu « Ontology » puis le sous menu « Concepts mapping » puis le sous menu « Measurement – Specification » de la fenêtre principale de MetricXpert. La liste des concepts de mesure et celle des concepts de spécification lui sont présentées. La liste des correspondances déjà établies lui est également affichée. Il sélectionne un concept de mesure et un concept de spécification, puis clique sur le bouton «map ». Une correspondance est établie entre le concept de mesure et le concept de spécification sélectionnés. La correspondance est matérialisée au niveau de la base de connaissances du système par la création de 2 «triggers » (déclencheurs) et une table de correspondances.**

Liste des données en entrée : **MAPPING_MESURE_SPECIFICATION, CONCEPT_DE_MESURE, CONCEPT_DE_SPECIFICATION.**

Liste des données en sortie : **MAPPING_MESURE_SPECIFICATION**

Notes et messages : **…**

Interface graphique : ***non disponible.***

L.2 « Mapping » de concepts inter méthodes de mesure

No 1	**Titre : « Mapping » de concepts inter méthodes de mesure**

Acteurs : **Gestionnaire de connaissances**

Objectifs : **Permettre au gestionnaire de connaissances d'établir des correspondances entre concepts de mesure issus de deux méthodes de mesure distinctes. Ces correspondances permettront le passage automatique d'une méthode à une autre. Elles sont traduites au niveau de la base de connaissances du système par des « triggers » (déclencheurs) et des tables de correspondances.**

Pré-conditions : **Le gestionnaire de connaissances a les droits requis pour une telle opération. Il s'est connecté au système avec le profil « gestionnaire de connaissances ».**

Post-conditions : **Une nouvelle correspondance est établie entre concepts de mesure issus de deux méthodes de mesure distinctes. La correspondance est matérialisée au niveau de la base de connaissances du système par la création de 2 « triggers » (déclencheurs) et une table de correspondances.**

Exceptions **: Aucune.**

Description sommaire : **Le gestionnaire de connaissances accède à l'interface de « mapping » via le menu « Ontology » puis le sous menu « Concepts mapping » puis le sous menu « Measurement – Measurement » de la fenêtre principale de MetricXpert. La liste des concepts de mesure lui est présentée. La liste des correspondances déjà établies entre concepts de mesure lui est également affichée. Il sélectionne deux concepts de mesure, puis clique sur le bouton « map ». Une correspondance est établie entre les deux concepts de mesure sélectionnés. La correspondance est matérialisée au niveau de la base de connaissances du système par la création de 2 « triggers » (déclencheurs) et une table de correspondances.**

Liste des données en entrée **: MAPPING_INTER_METHODES, CONCEPT_DE_MESURE.**

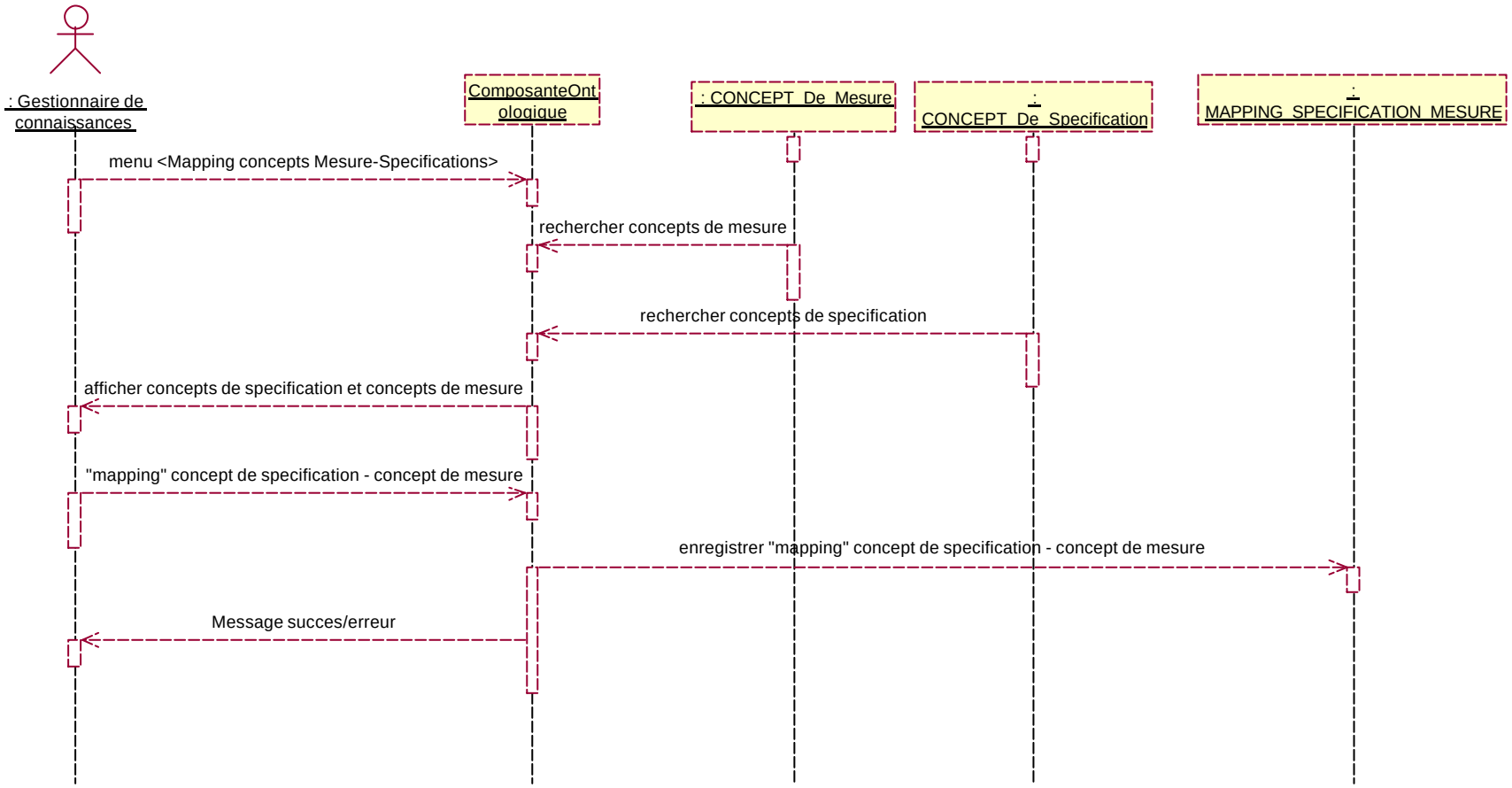
Liste des données en sortie : **MAPPING_INTER_METHODES**

Notes et messages : **…**

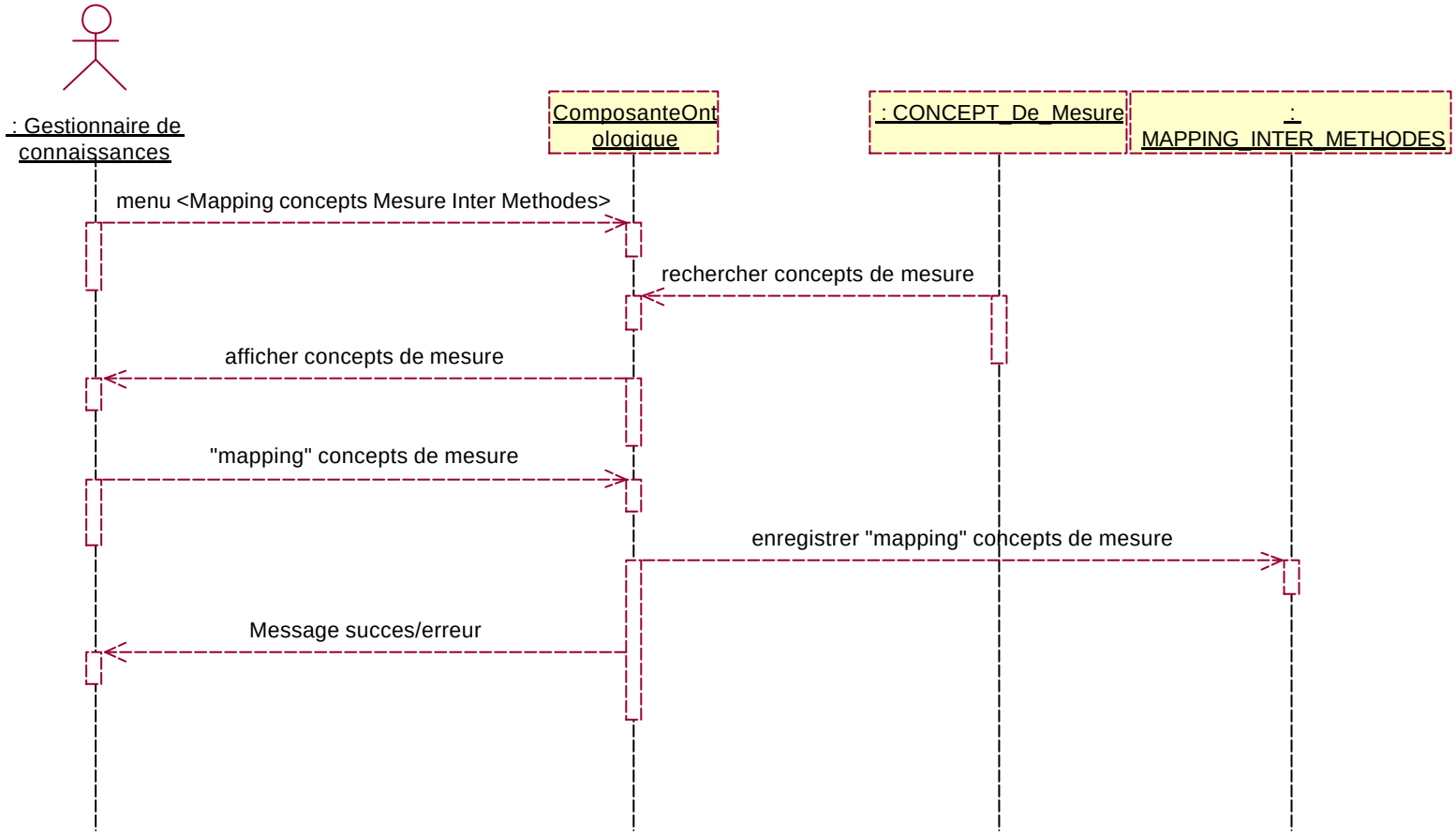
Interface graphique : ***non disponible.***

DIAGRAMMES DE SEQUENCE

L.1 Mapping de concepts Mesure-Spécifications



L.2 Mapping de concepts inter méthodes de mesure



COMPOSANTE D'ADMINISTRATION

DIAGRAMME DES CAS D'UTILISATION

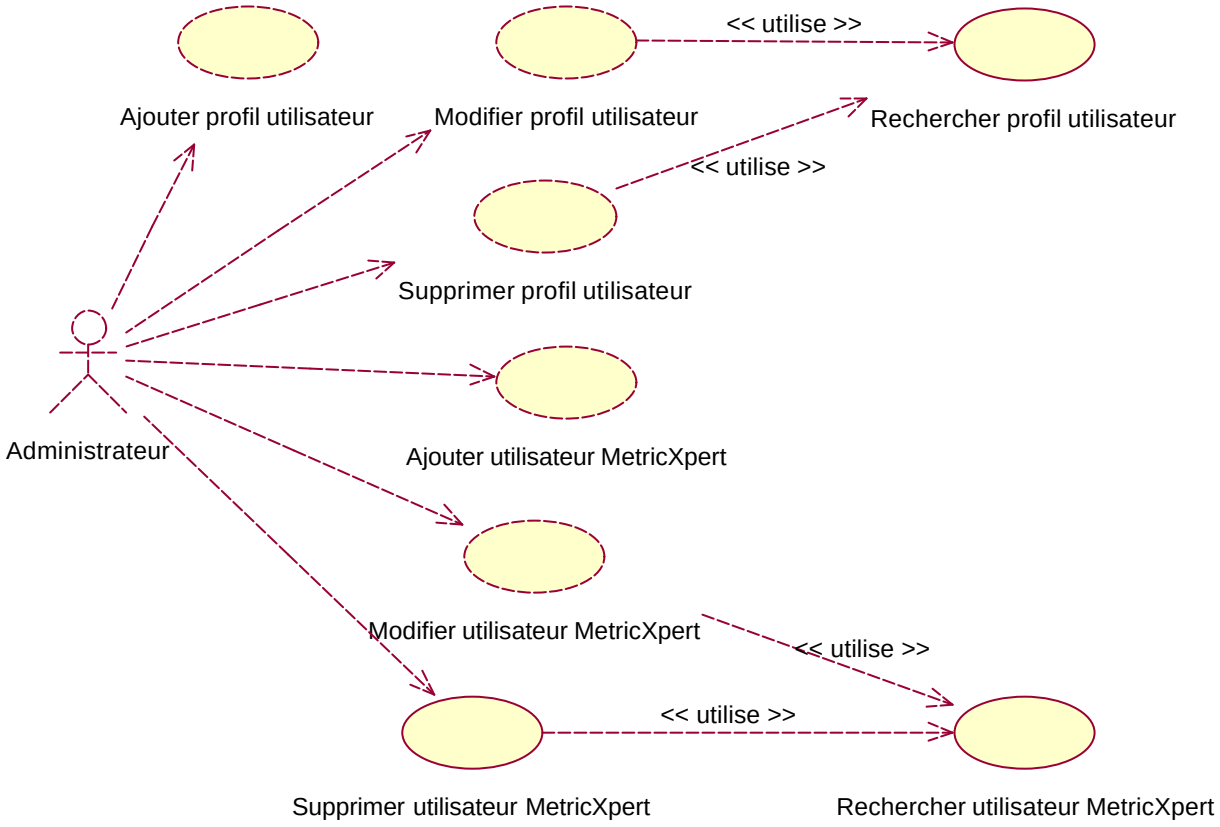


Figure 31 : Diagramme des cas d'utilisation pour la composante d'administration

L.1 Cas d'utilisation 2 : Ajouter Profil utilisateur

No 1	**Titre : Ajouter Profil utilisateur**
Acteurs **: Administrateur de MetricXpert** Objectifs **: Permettre à l'administrateur de MetricXpert d'ajouter dans la base de connaissances du système, un nouveau profil d'utilisateur du système.** Pré-conditions **: L'administrateur accède à l'interface d'administration du système.** Post-conditions **: un nouveau profil d'utilisateur du système est créé et stocké dans la base de connaissances du système.** Exceptions **: Aucune.**	
Description sommaire : **L'administrateur accède à l'interface d'administration du système. Il inscrit les informations (nom, description) relatives au nouveau profil d'utilisateur dans les espaces appropriés prévus et clique sur le bouton « ok ». Les informations inscrites sont alors**	

validées puis stockées dans la base de connaissances du système.
Liste des données en entrée **: nom et description sommaire du nouveau profil d'utilisateur.** Liste des données en sortie : **ProfilUtilisateur** Notes et messages : **…** Interface graphique **: *non disponible.***

L.2 Cas d'utilisation 2 : Ajouter utilisateur

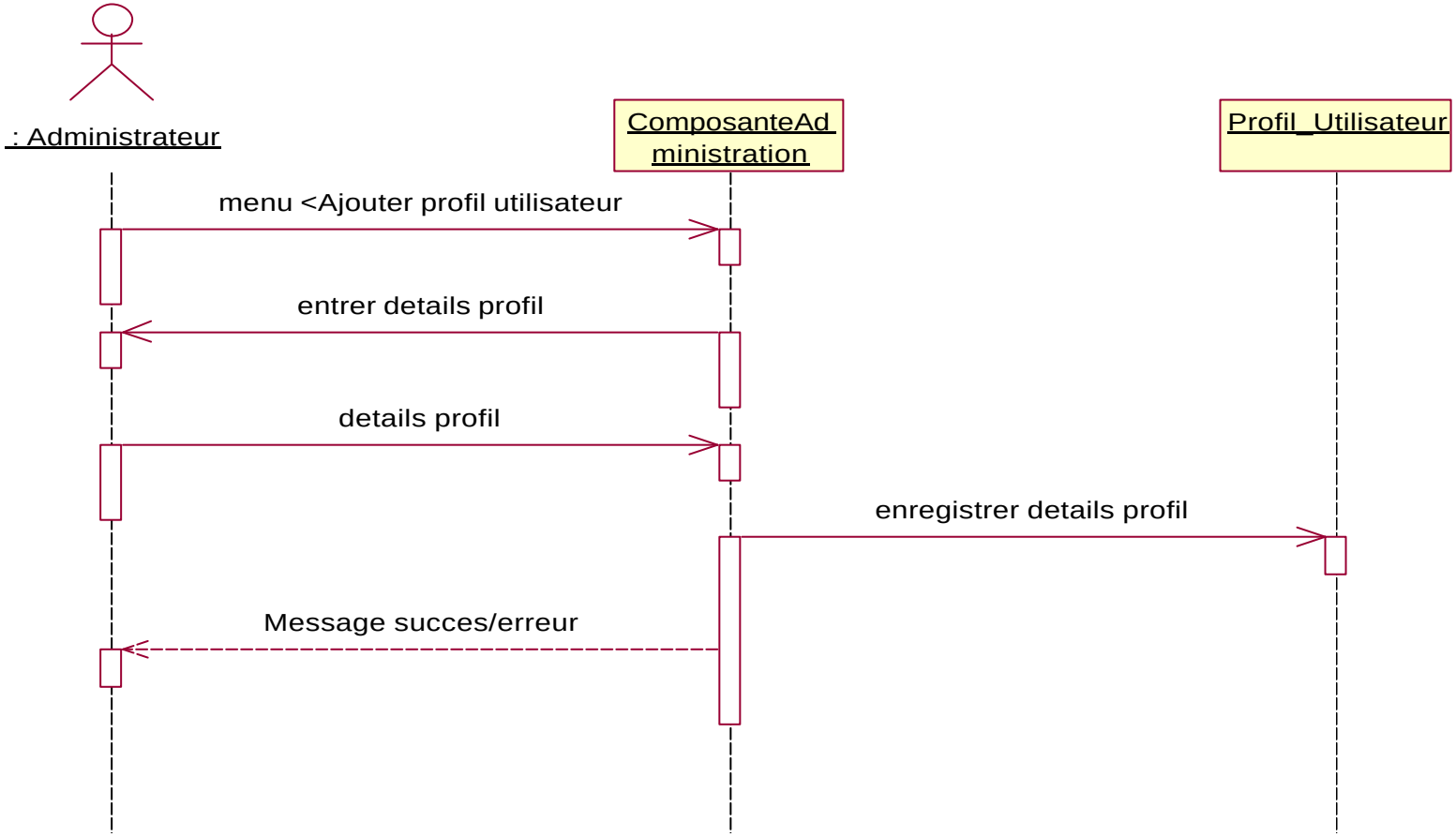
No 1	**Titre : Ajouter Profil utilisateur**
Acteurs : **Administrateur de MetricXpert** Objectifs **: Permettre à l'administrateur de MetricXpert d'ajouter dans la base de connaissances du système, un nouvel d'utilisateur du système.** Pré-conditions : **L'administrateur accède à l'interface d'administration du système.** Post-conditions : **un nouvel utilisateur du système est créé et stocké dans la base de connaissances du système.** Exceptions **: Aucune.**	
Description sommaire : **L'administrateur accède à l'interface d'administration du système. Il inscrit les informations (nom d'utilisateur, mot de passe, description, profil) relatives au nouvel utilisateur dans les espaces appropriés prévus et clique sur le bouton «ok ». Les informations inscrites sont alors validées puis stockées dans la base de connaissances du système.**	
Liste des données en entrée **: nom d'utilisateur, mot de passe, description sommaire et profil du nouvel utilisateur.** Liste des données en sortie : **Utilisateur** Notes et messages : **…** Interface graphique **: *non disponible.***	

Nota Benne : Le détail des autres cas d'utilisation de la composante d'administration sera fourni dans un document de spécifications et exigences logicielles qui sera produit ultérieurement dans le cadre de la construction d'un outil commercial.

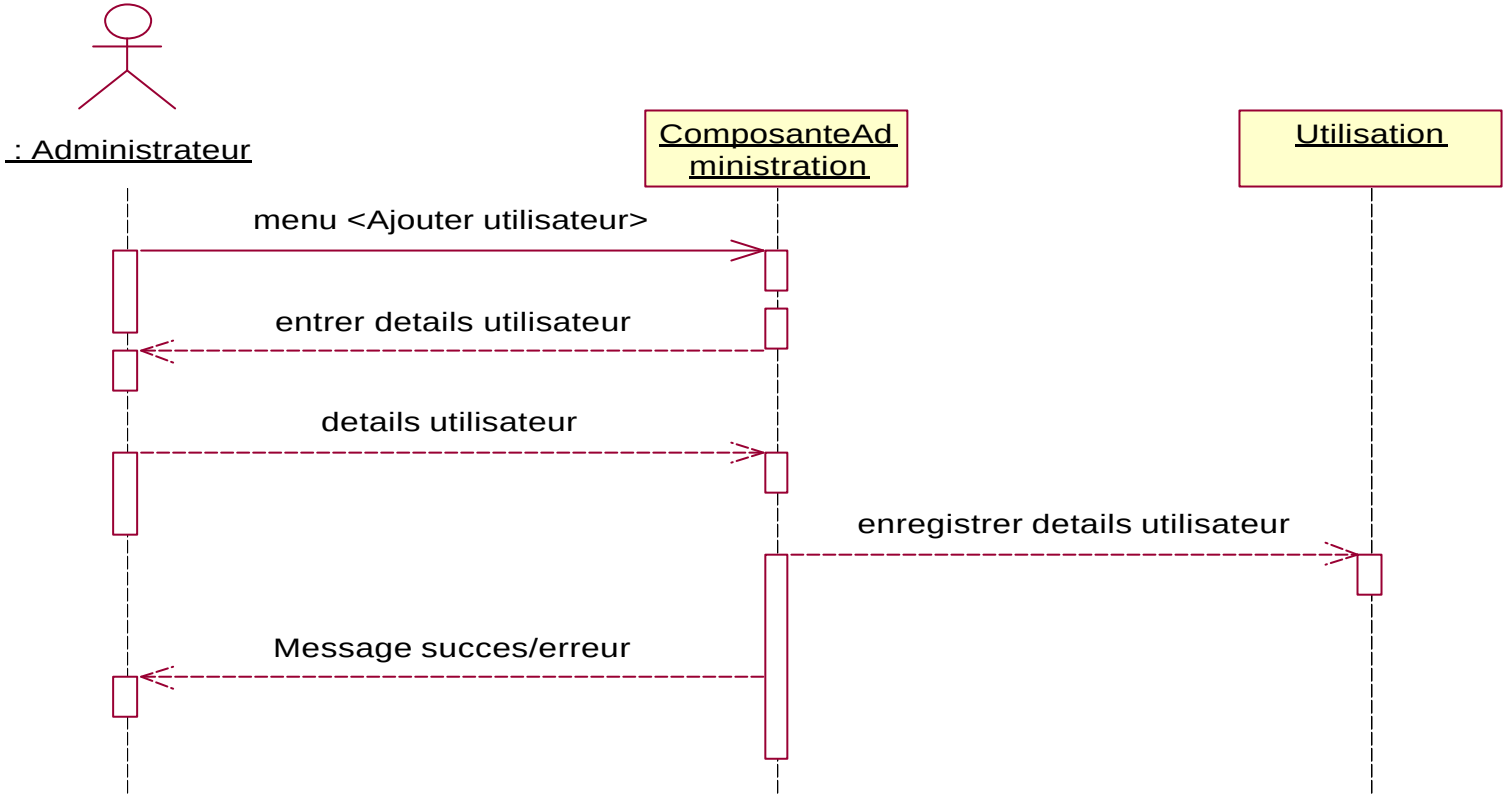
Nous présentons dans la section suivante les diagrammes de séquences associées aux cas d'utilisation que nous avons examinés plus haut.

DIAGRAMMES DE SEQUENCE

L.1 Ajouter Profil utilisateur



L.2 Ajouter Utilisateur



MODÈLE STRUCTURAL

DIAGRAMME DE CLASSES ASSOCIÉES A LA COMPOSANTE DE SPECIFICATIONS

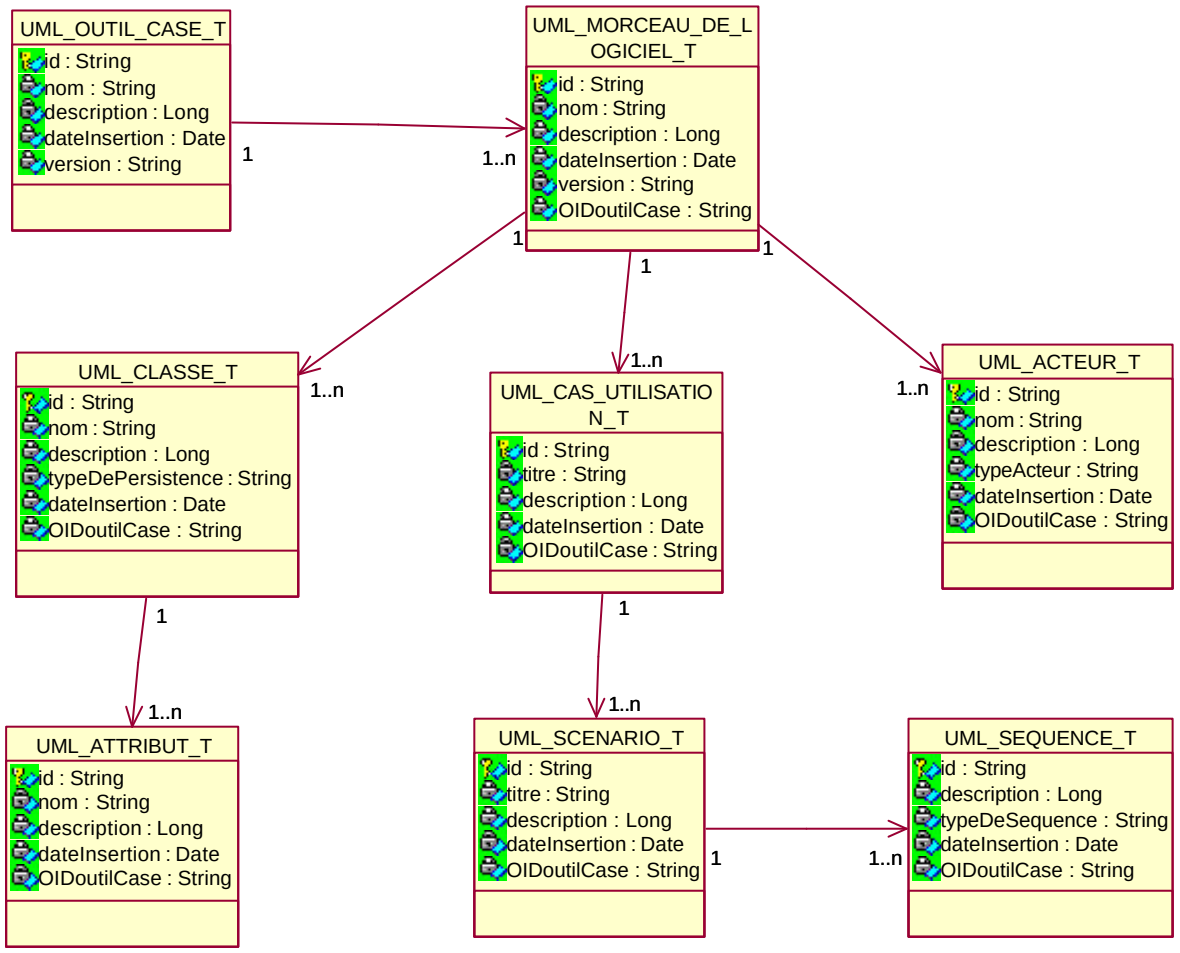


Figure 32 : Diagramme de classes associées a la composante de spécifications

Dans ce diagramme, l'on retrouve les concepts du langage de spécification UML, jugés pertinents relativement à la mesure de la taille fonctionnelle. Il s'agit en fait d'un ensemble d'éléments que l'on retrouve dans le méta-modèle associé au langage UML.

DIAGRAMME DE CLASSES ASSOCIÉES A LA COMPOSANTE DE MESURE

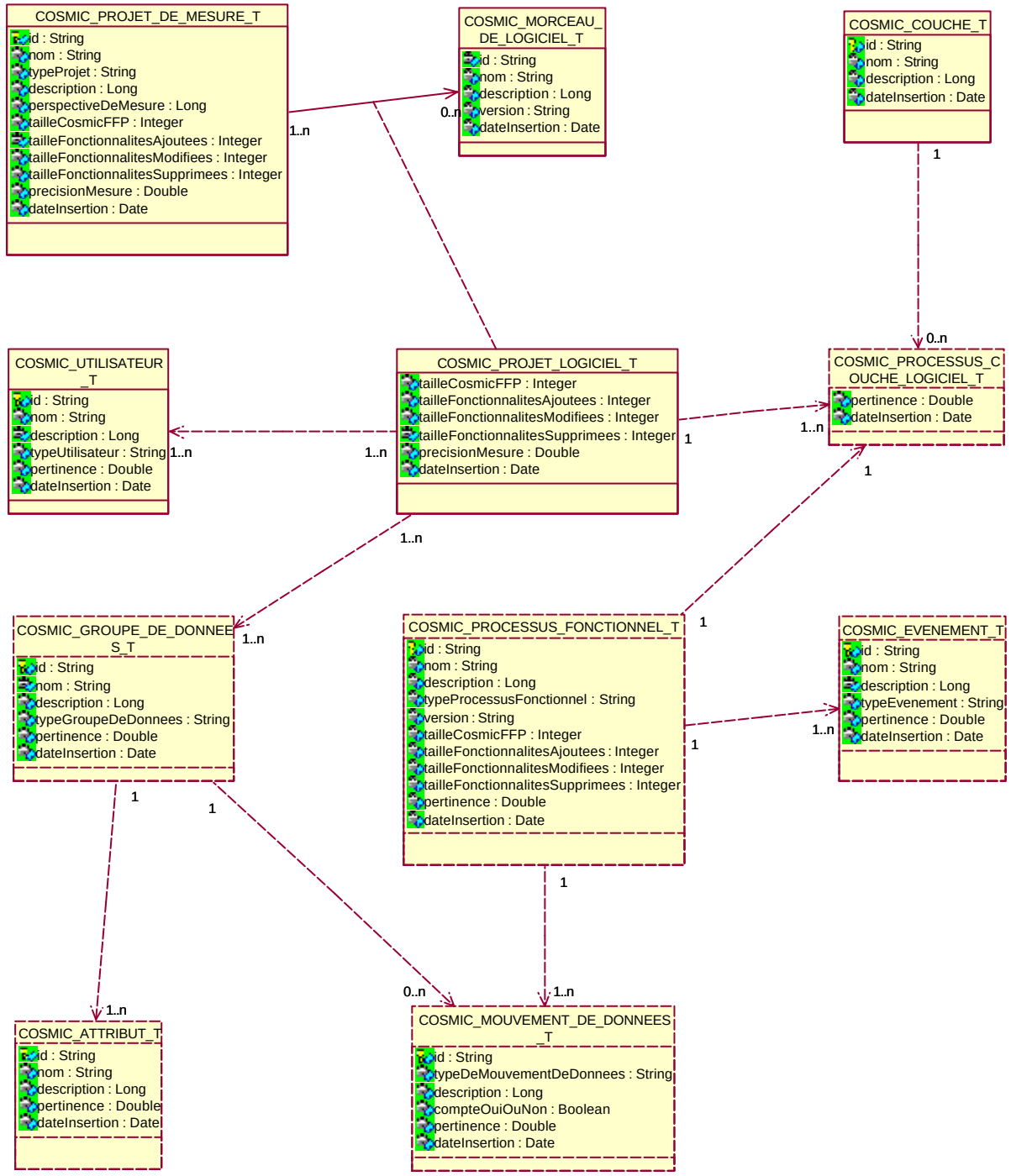


Figure 33 : Diagramme de classes associées a la composante de spécifications

Dans ce diagramme, l'on retrouve les concepts de mesure associés à la méthode de mesure COSMIC-FFP. Il s'agit en fait d'une matérialisation dans le prototype MetricXpert de l'ontologie de domaine associée à la méthode de mesure COSMIC-FFP.

DIAGRAMME DE CLASSES ASSOCIÉES A LA COMPOSANTE ONTOLOGIQUE

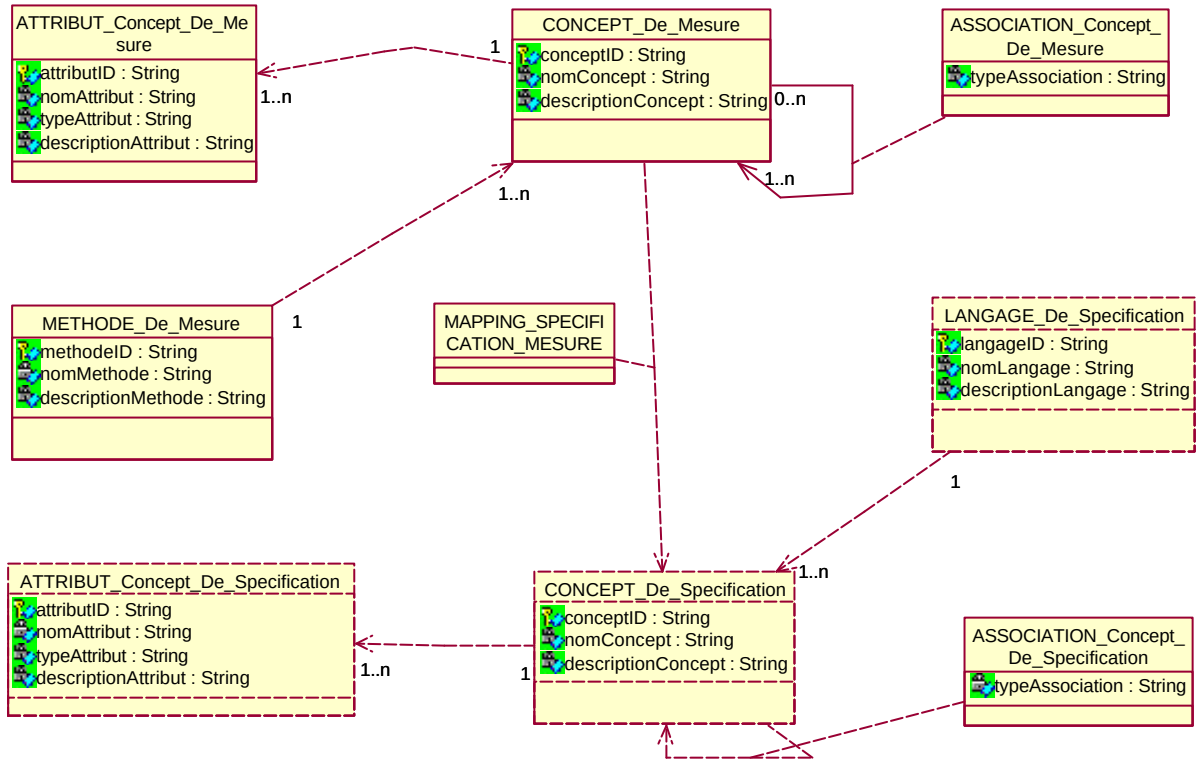


Figure 34 : Diagramme de classes associées a la composante ontologique

Ce diagramme constitue une sorte de méta modèle pour la description d'éléments ontologiques (concepts et liens entre concepts) associés aux méthodes de mesure et aux langages de spécification. Les liens entre concepts sont des associations ou des correspondances (« mapping ») entre lesdits concepts.

DIAGRAMME DE CLASSES ASSOCIÉES A LA COMPOSANTE D'ADMINISTRATION

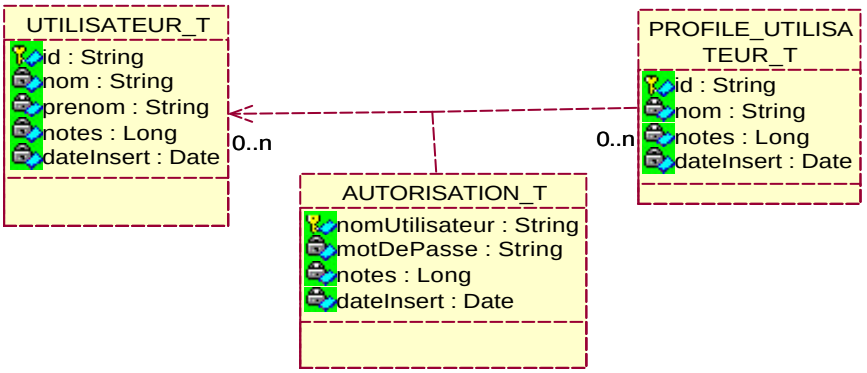


Figure 35 : Diagramme de classes associées a la composante d'administration

CHOIX TECHNOLOGIQUES POUR LA CONSTRUCTION DU PROTOTYPE

Pour la construction du prototype, nous faisons appel à la plate-forme de développement Sun One Studio, Community Edition, avec le langage de programmation Java. Le SGBD Oracle 8i est utilisé pour l'implantation de la base de connaissances et de faits du système (connaissances et faits relatifs aux spécifications UML de morceaux de logiciels, aux modèles COSMIC-FFP de morceaux de logiciels). L'ontologie de domaine associée à la méthode COSMIC-FFP ainsi que le méta-modèle associé à UML sont mis à contribution pour l'implantation de la base de connaissances. Les correspondances entre concepts UML et concepts COSMIC-FFP sont matérialisées au niveau de la base de connaissances par des «gâchettes » (« triggers ») et des tables de correspondances. Ces correspondances pourraient ainsi être mises à jour via une interface spécialisée (ceci fera l'objet de nos travaux post-thèse).

Illustration :

Les concepts de «scénario » (UML) et de «processus fonctionnel » sont matérialisés au niveau de la base de données par les tables «UML_SCENARIO_T» et « COSMIC_PROCESSUS_FONCTIONNEL_T» respectivement. La correspondance entre ces deux concepts est matérialisée au niveau de la base de connaissances du système par la table « MAPPING_SCENARIO_T» et les « gâchettes » (« triggers »). Ci-dessous, nous présentons les scripts de création de ces différentes tables et « gâchettes » (« triggers »).

Le format « xmi » a été choisi pour le stockage des spécifications exploitables par le prototype. Certains outils CASE [Rational Rose, Together, Tau, etc.] permettent le stockage des spécifications dans ce format. Le module d'extraction des éléments de spécifications UML jugés pertinents pour la mesure (à partir de spécifications produites à l'aide de tels outils) ainsi que le module de documentation des mesures ont été confiés à des étudiants de maîtrise dans le cadre de leurs projets de maîtrise. Quant aux modules de gestion de la précision des mesures et d'analyse des résultats de mesure, ils seront complétés ultérieurement : Ils sont importants pour un produit commercial, mais relativement aux objectifs que nous nous sommes fixés dans le cadre de la thèse, ils ne sont pas prioritaires. Dans la section qui suit nous proposons une trace d'exécution du prototype dans son état actuel.

TRACE D'EXÉCUTION DU PROTOTYPE: MESURER LA TAILLE FONCTIONNELLE D'UN MORCEAU DE LOGICIEL DONT LES SPECIFICATIONS UML SONT STOCKEES DANS UN FICHIER AU FORMAT XMI

Pour cette trace d'exécution du prototype, nous allons utiliser un morceau de logiciel dont nous disposons des spécifications UML stockées dans un fichier au format «.xmi ». Il s'agit

d'une étude de cas contenue dans le livre de Pierre Alain Muller [Muller 00]. Cette étude de cas est relative à un système pour la gestion des accès à un bâtiment.

Le diagramme des cas d'utilisation pour le morceau de logiciel est le suivant :

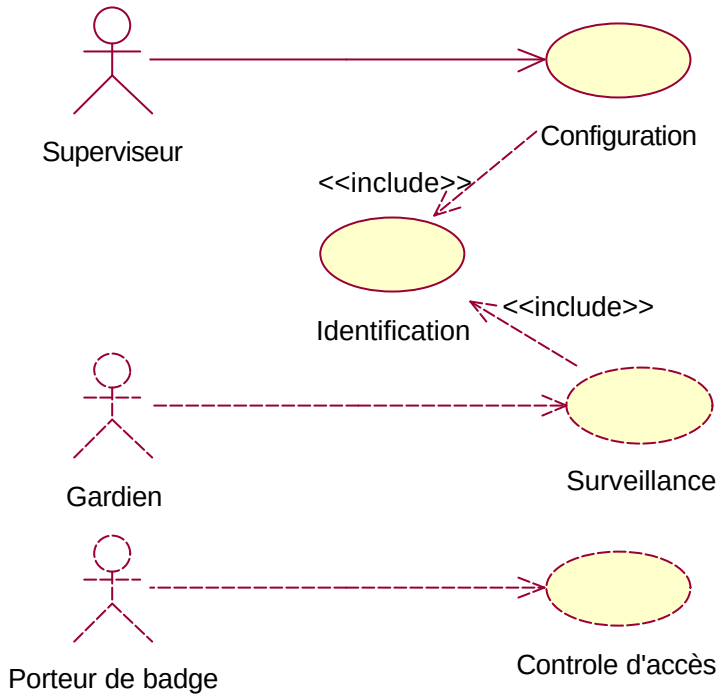
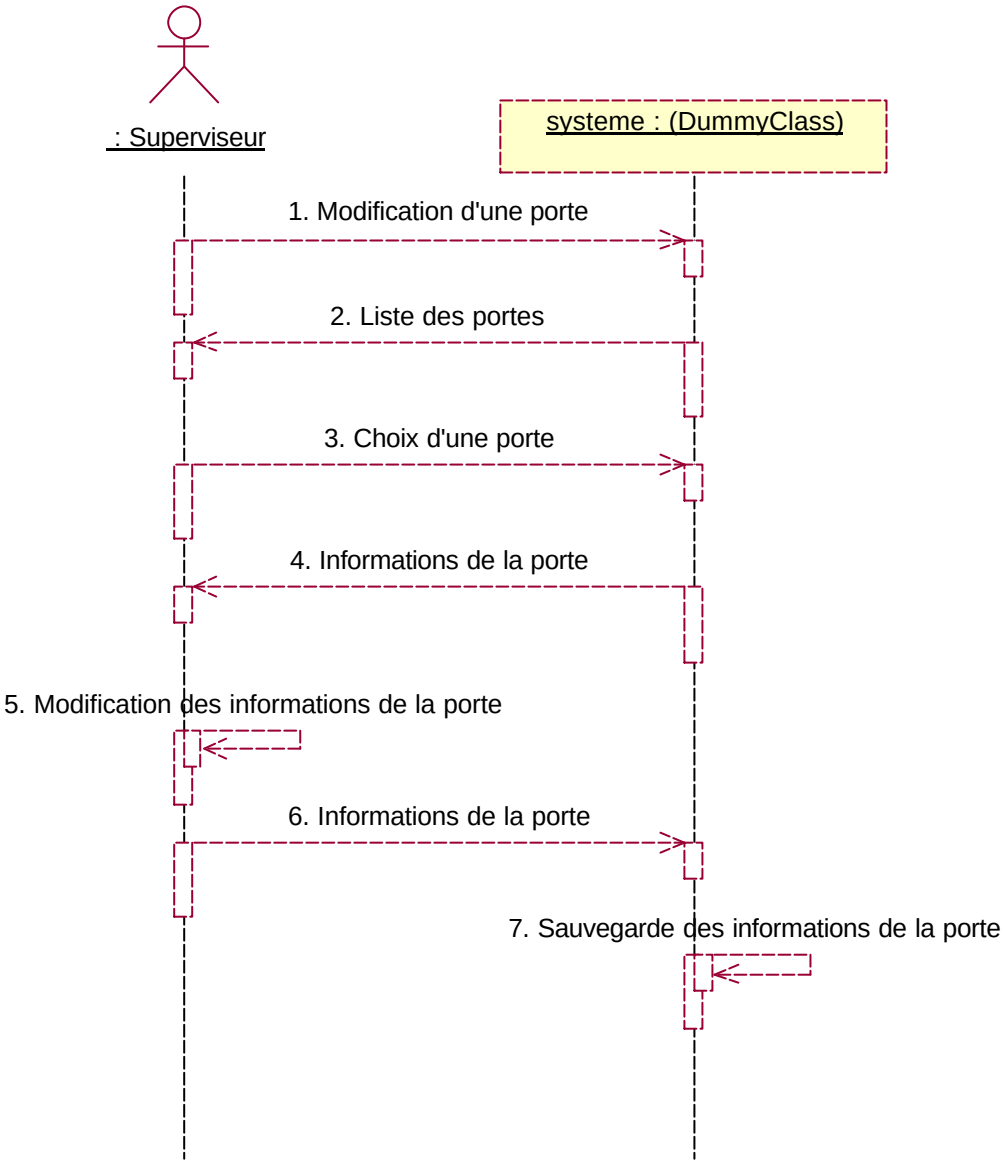


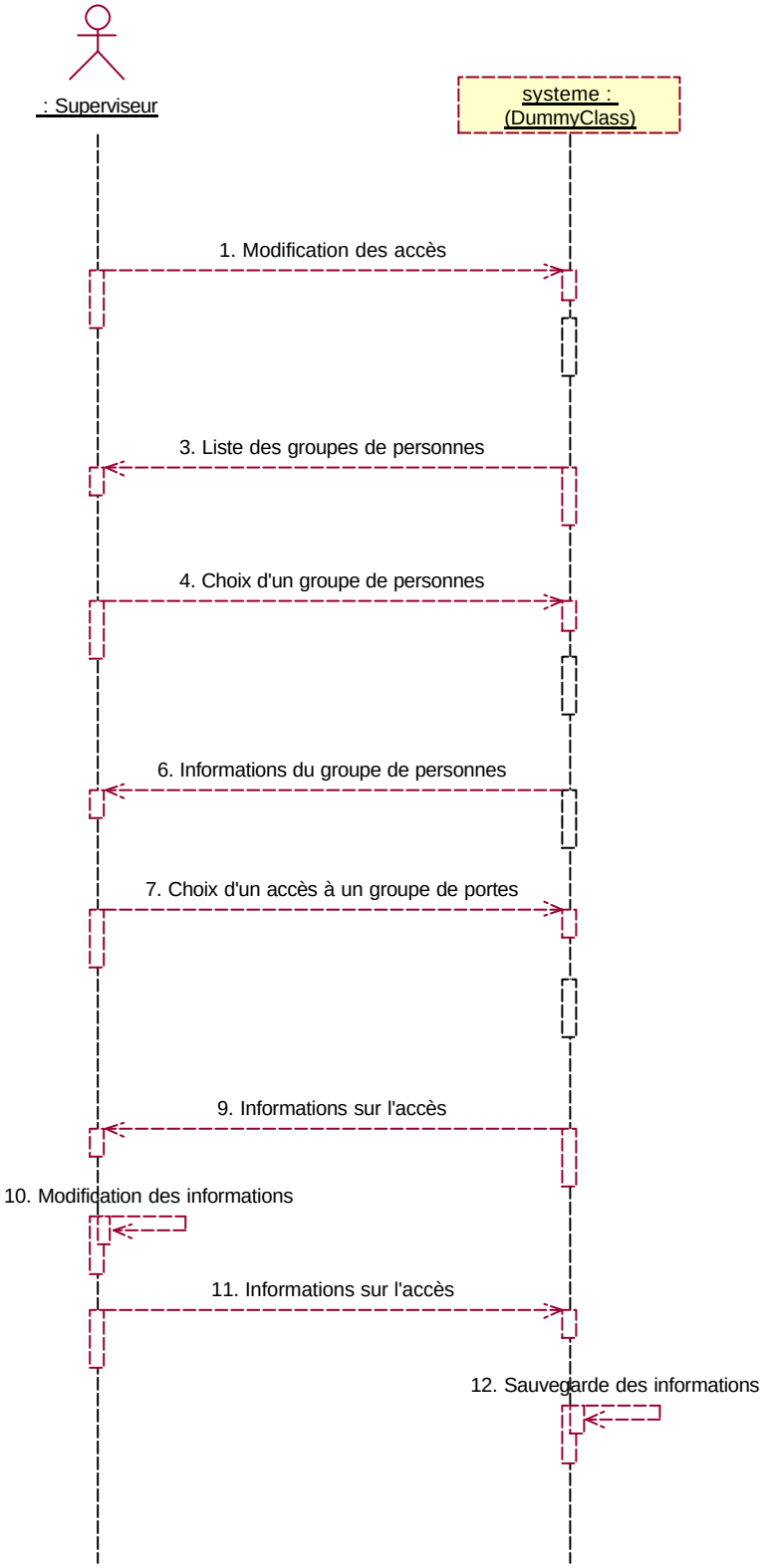
Figure 36 : Diagramme des cas d'utilisation pour le morceau de logiciel à mesurer

Pour la trace d'exécution nous allons focaliser notre attention sur le cas d'utilisation « Configuration », et plus spécifiquement sur les trois (3) diagrammes de séquences ci-après qui sont associés à des scénarios identifiés pour ce cas d'utilisation : « Modification des portes », « Modification des accès d'un groupe de personnes à un groupe de portes » et « Recherche des droits d'accès d'une personne pour une porte donnée ».

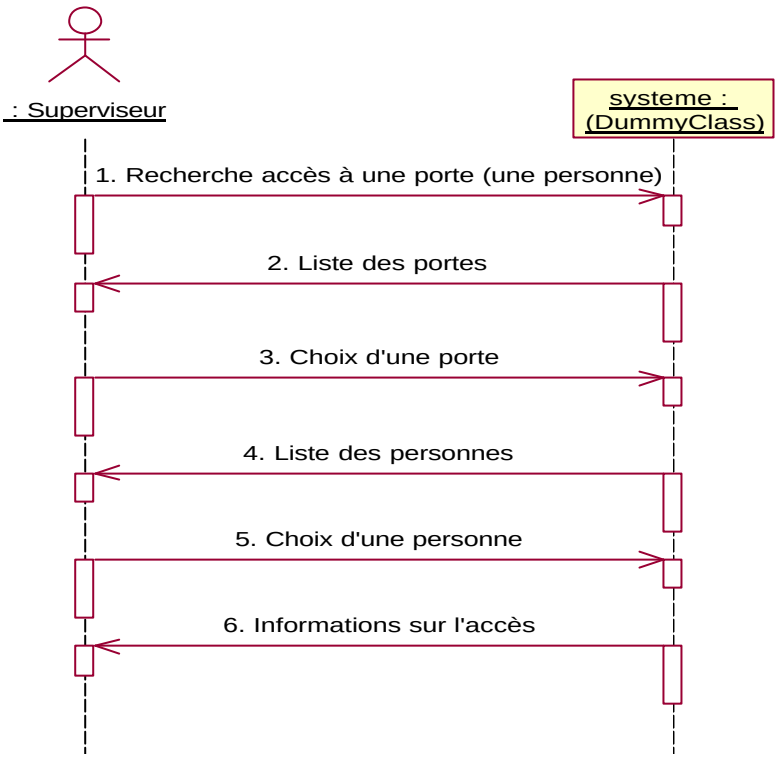
1. Modification des portes



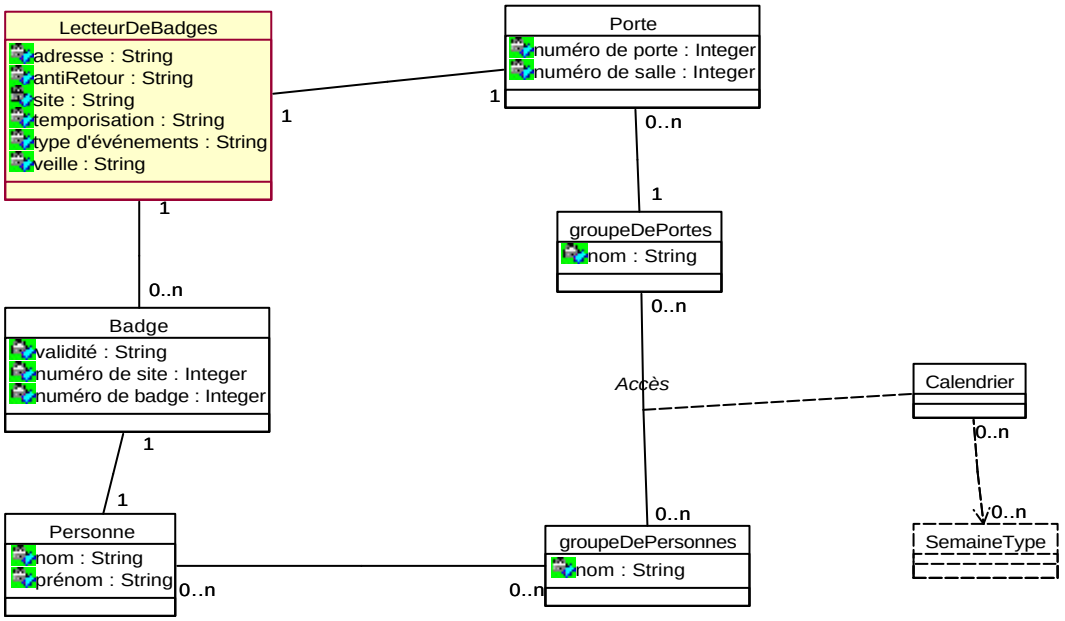
2. Modification des accès d’un groupe de personnes à un groupe de portes



3. Recherche des droits d'accès d'une personne pour une porte donnée



Le diagramme de classes pour le logiciel à mesurer est le suivant :



Ci-dessous nous présentons une trace d'exécution du prototype pour mesurer le morceau de logiciel considéré.

1. Dans le menu Spécifications de l'interface principale de MetricXpert, sélectionner le sous-menu « Add/Import UML Specifications ».

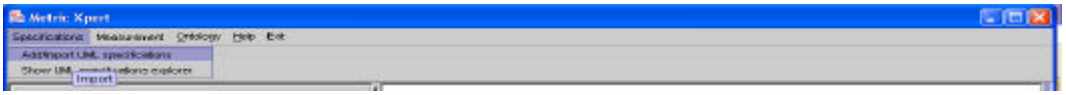


Figure 37 : Menu principal de MetricXpert

2. L’interface d’ajout de morceaux de logiciel UML est affichée.

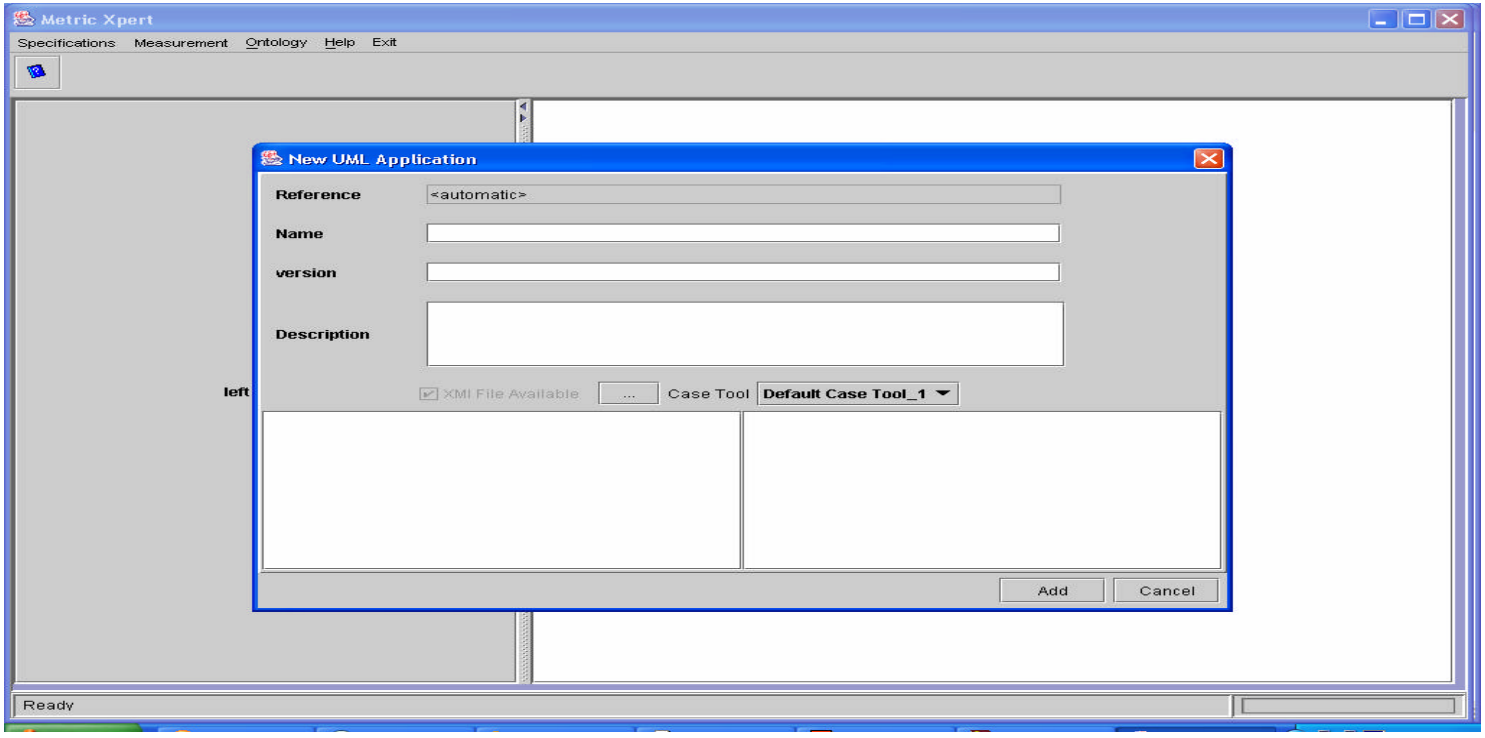


Figure 38 : Interface d’ajout de morceaux de logiciel UML

3. Compléter les champs «Name », « version » et «Description » de l’interface et sélectionner l’outil CASE utilisé pour produire les spécifications. Puis cliquer sur le bouton d’importation des spécifications (« **…** »). Une fenêtre de sélection de fichier apparaît.

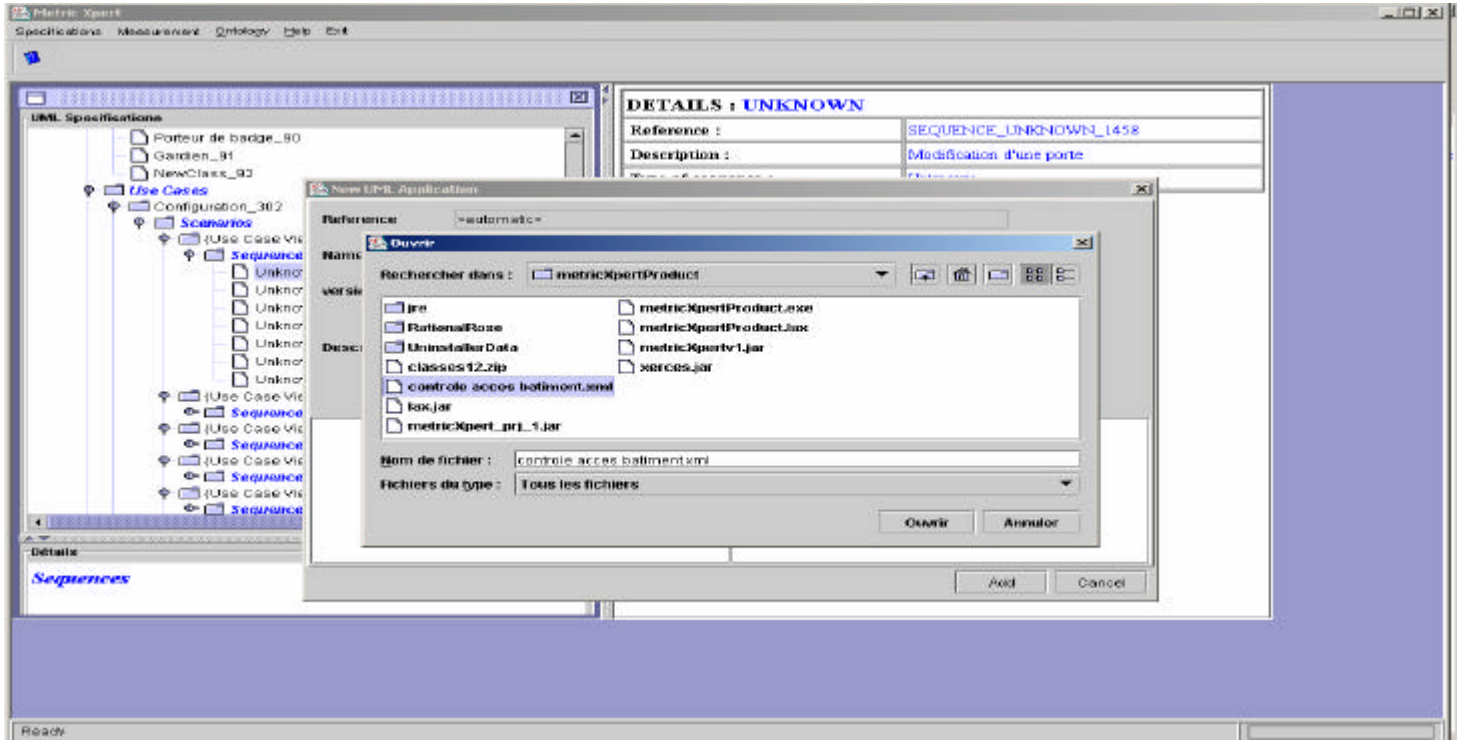


Figure 39 : Fenêtre de selection du fichier .xmi contenant les specifications

4. Sélectionner le fichier contenant les spécifications, puis cliquer sur le bouton <Open>. Le module d'extraction se charge alors d'extraire les concepts de spécification jugés pertinents pour la mesure et les affiche dans l'interface d'ajout de morceaux de logiciel UML

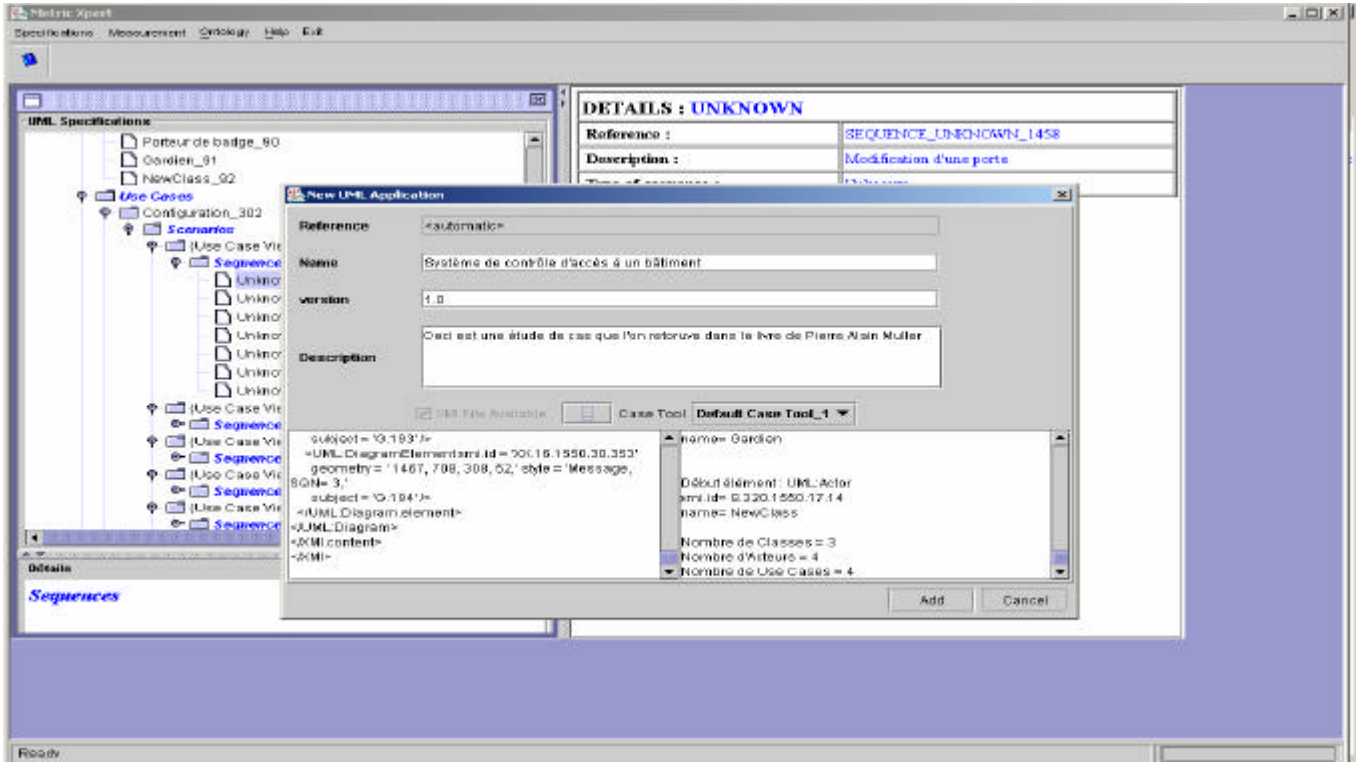
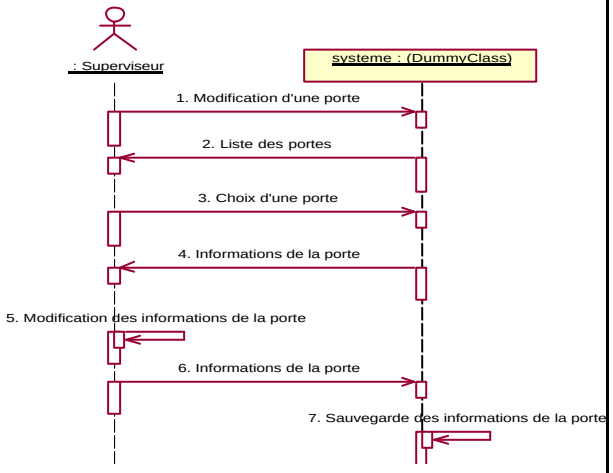
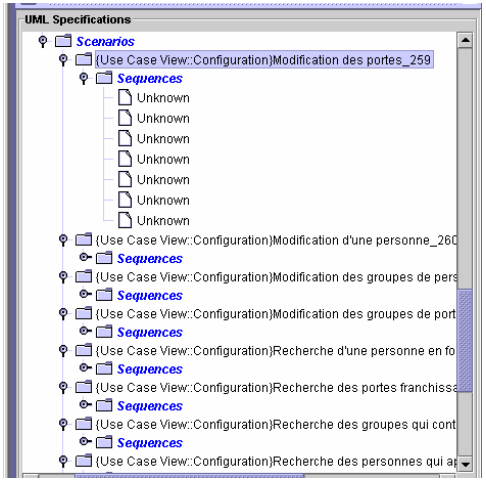
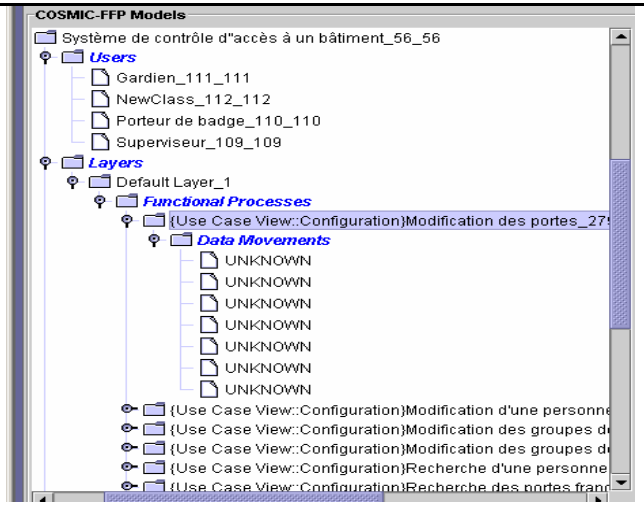


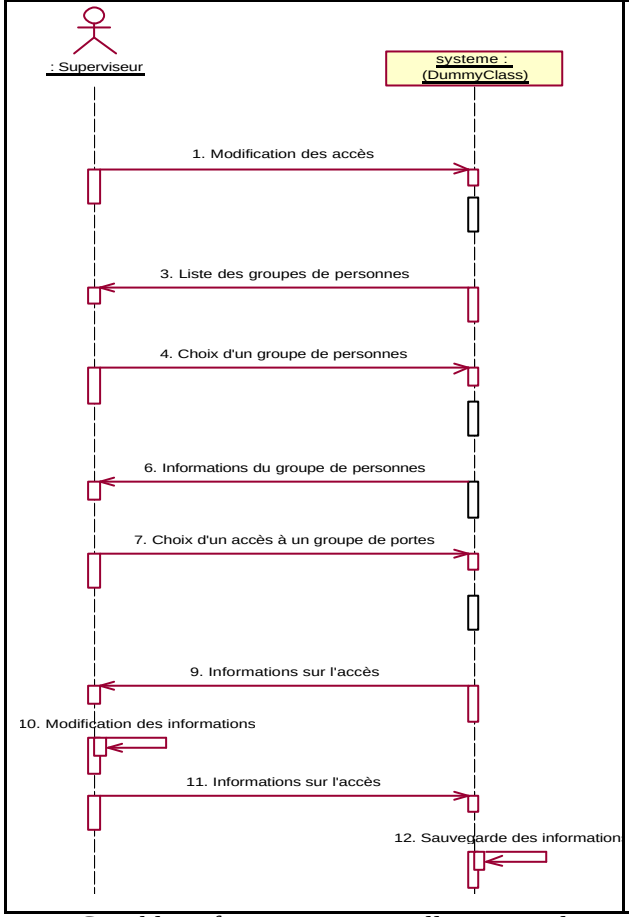
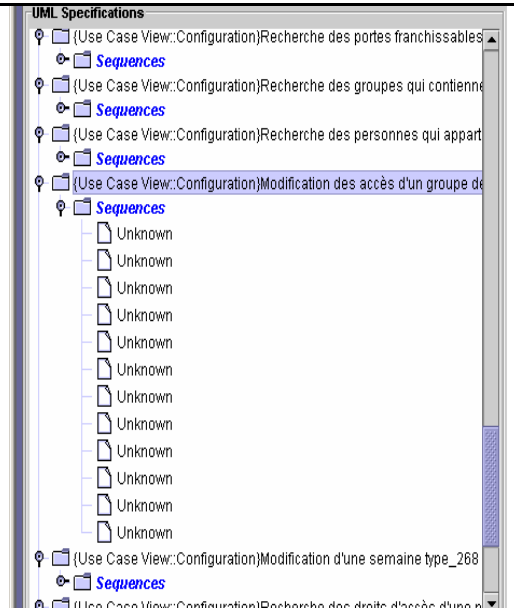
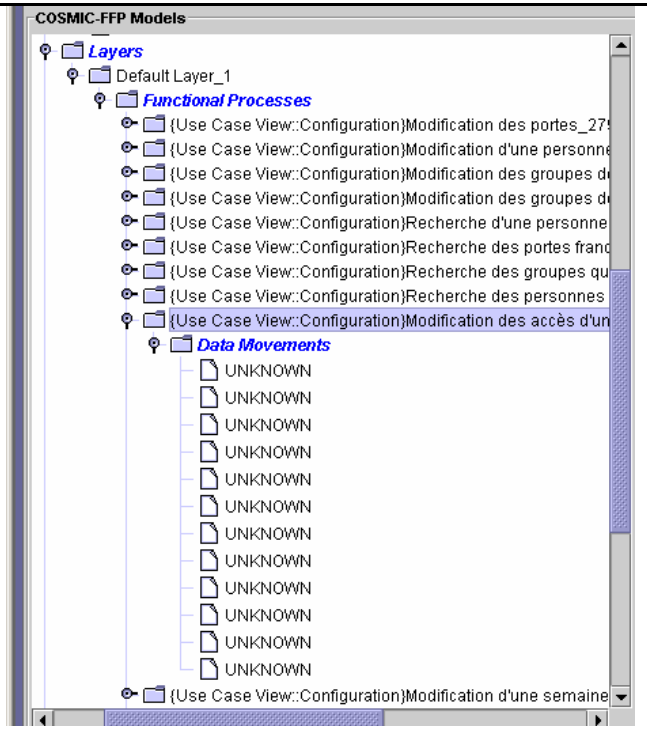
Figure 40 : Concepts de spécifications extraits et affichés dans l'interface d'ajout de morceaux de logiciel UML

5. Cliquer sur le bouton <Add> de l'interface d'ajout de morceaux de logiciel UML pour terminer l'opération. Le module de gestion de la base de connaissances se charge alors de sauvegarder les détails de la nouvelle application ainsi que tous les concepts de spécifications extraits précédemment par le module d'extraction. Au niveau de la base de connaissances, un modèle COSMIC-FFP du morceau de logiciel est automatiquement généré à partir des concepts de spécification extraits et des correspondances entre concepts de mesure et concepts de spécifications. Ces correspondances doivent préalablement avoir été stockés dans la base (résultats de « mapping »). **Nota Benne :** Le prototype déclare *« Unknown »* tout message UML tel qu'il ne peut pas déterminer l'expéditeur (*« Sender »*) et/ou le récepteur (*« Receiver »*). À tout message UML déclaré *« Unknown »* par le prototype, sera associé un mouvement de données COSMIC-FFP déclaré également *« Unknown ».*

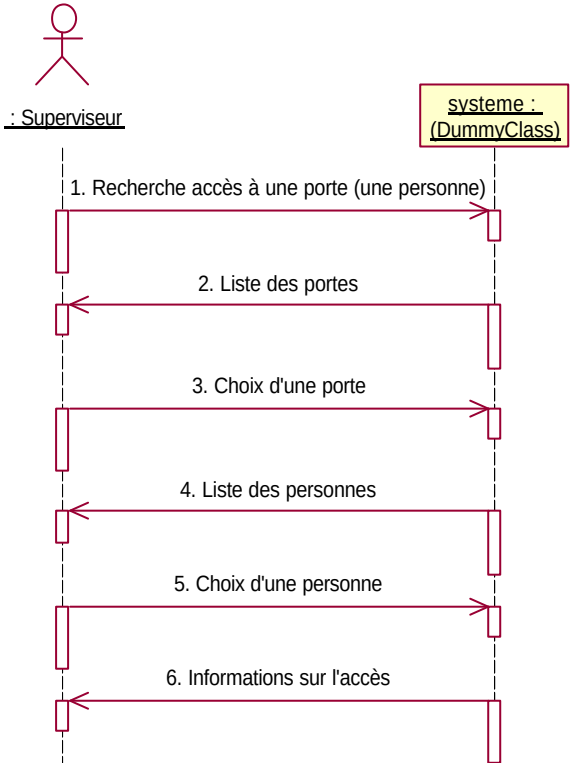
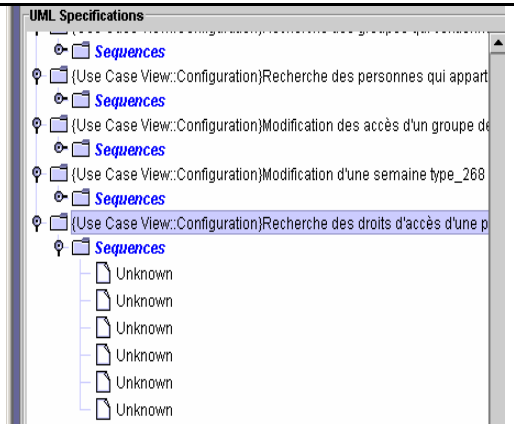
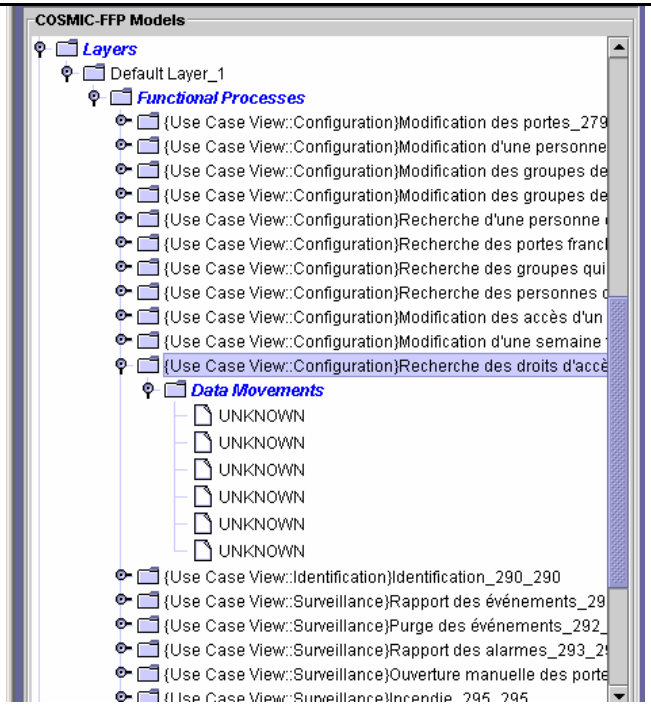
Diagramme de séquence : *Modification d'une porte*	**Concepts UML identifiés par le prototype**	**Concepts COSMIC-FFP identifiés par le prototype**
		

Ce tableau illustre le passage d'un diagramme de séquence à un processus fonctionnel COSMIC-FFP. Pour chaque diagramme de séquence, il sera créé une instance de scénario UML et pour chaque instance de scénario UML, il sera créé une instance de processus fonctionnel COSMIC-FFP. Pour chaque séquence dans le diagramme de séquence, il sera créé une instance de séquence/message UML, et pour chaque instance de séquence/message UML, il devrait être créé une instance de movement de données COSMIC-FFP par classe

impliquée dans le message. Mais pour cela, il faudrait expliciter les diagrammes de collaboration. (Malheureusement pas toujours disponibles et souvent incomplets). Pour le moment, l'outil se limite à un mouvement de données par message.

Diagramme de séquence : *Modification des accès d'un groupe de personnes à un groupe de portes*	**Concepts UML identifiés par le prototype**	**Concepts COSMIC-FFP identifiés par le prototype**
		

Ce tableau fournit une autre illustration du passage d'un diagramme de séquence à un processus fonctionnel COSMIC-FFP.

Diagramme de séquence : *Recherche des droits d'accès d'une personne pour une porte donnée*	**Concepts UML identifiés par le prototype**	**Concepts COSMIC-FFP identifiés par le prototype**
		

Ce tableau fournit une autre illustration du passage d'un diagramme de séquence à un processus fonctionnel COSMIC-FFP.

6. La taille fonctionnelle du morceau de logiciel est également calculée automatiquement et stockée. Dans l'interface principale de MetricXpert, le morceau de logiciel est ajouté sur l'explorateur des applications UML avec tous les concepts extraits. La taille fonctionnelle du morceau de logiciel est aussi affichée

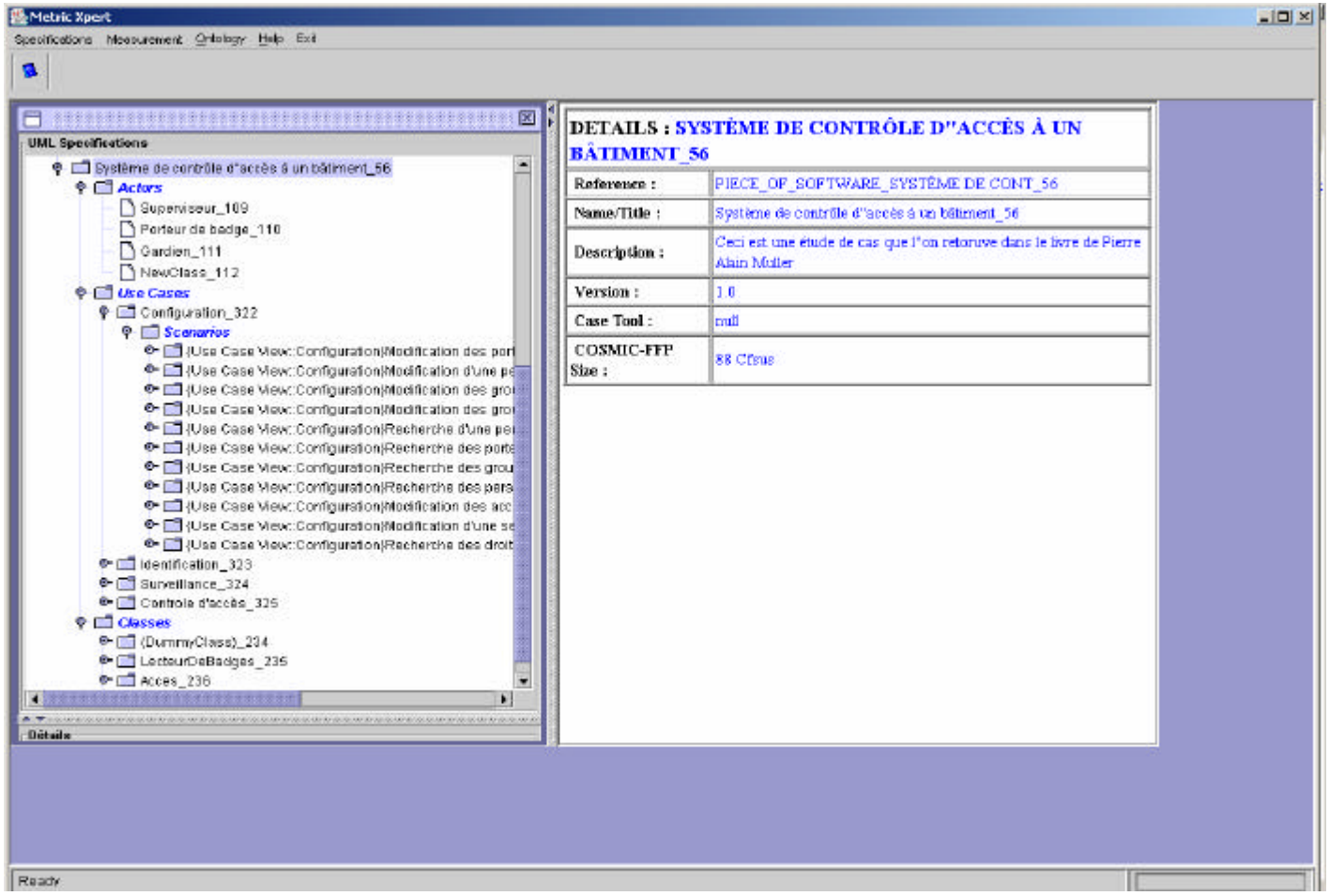


Figure 41 : Détails UML du nouveau morceau de logiciel affiché dans l'interface principale de MetricXpert

7. La taille affichée est une taille brute, susceptible d'être affinée en ajustant manuellement le modèle COSMIC-FFP du morceau de logiciel. Pour avoir le détail du modèle COSMIC-FFP du morceau de logiciel, sélectionner le sous-menu « Show Models » du menu « Measurement »

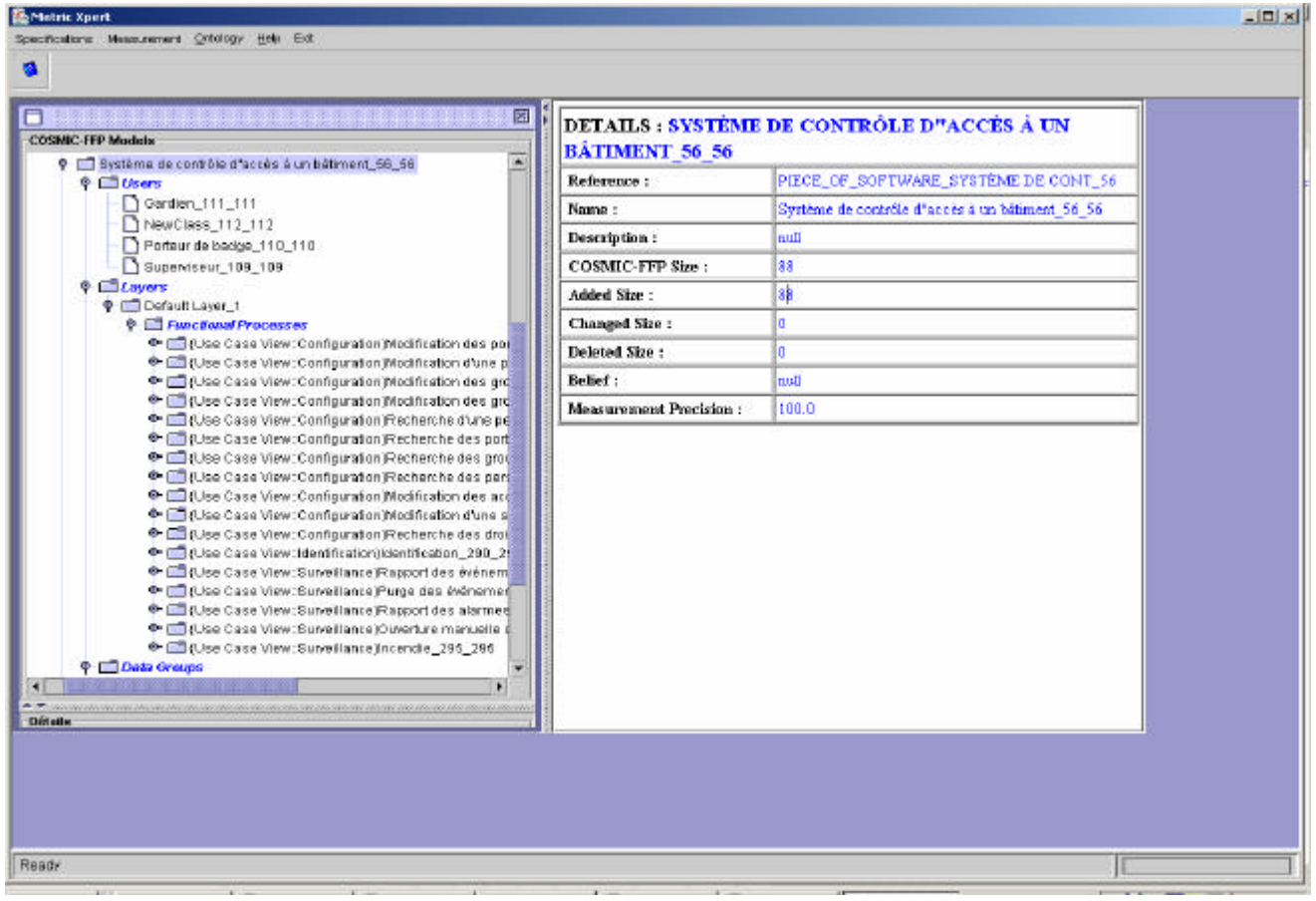


Figure 42 : Détail du modèle COSMIC-FFP du morceau de logiciel

ANNEXE G : PLANIFICATION DETAILLEE DU PROJET

VOLET EXPLORATOIRE

Légende: RL: **Revue de littérature** – RD: **Rédaction** – A: **Analyse** – C: **Conceptualisation** – D: **Développement** – Tu: **Test Unit.** – I: **Intégration** – Tg : **Test Global** – S : **Simulation** – X : **Autres**

Code : IMPR_00

Étape 1: Imprégnation										
Objectif : Revues de littératures										
TÂCHES ASSOCIÉES	**TYPE DE TÂCHE**									**LIVRABLES**
	RL	**RD**	**A**	**C**	**D**	**Tu**	**I**	**Tg**	**S**	**X**	
État de l'art : Mesure de la taille fonctionnelle du logiciels (standard ISO, méthodes de mesure, outils de mesure, automatisation, ...)	√									E_A_Mes_0.1 (rapport de lecture)
État de l'art : La représentation des connaissances	√									E_A_RKces_0.1 (rapport de lecture)
État de l'art : Les ontologies	√									E_A_Onto_0.1 (rapport de lecture)
État de l'art : Catégorisation	√									E_A_Categ_0.1 (rapport de lecture)
Le mapping/la traduction	√									E_A_Mapp_0.1 (rapport de lecture)
État de l'art : La notation UML	√									E_A_UML_0.1 (rapport de lecture)
État de l'art : Synthèse		√	√							E_A_0.1 (rapport de synthèse)

Code : KCES_00

Étape 2 : Analyse & Formalisation ontologique des procédures de mesure											
Objectif : Analyse et formalisation ontologique du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications											
TÂCHES ASSOCIÉES	**TYPE DE TÂCHE**										**LIVRABLES**
	RL	**RD**	**A**	**C**	**D**	**Tu**	**I**	**Tg**	**S**	**X**	
Lien entre la catégorisation/classification et le design/application d'une méthode de mesure de la taille fonctionnelle des logiciels			√	√							Mes_Cat_0.1 (rapport d'analyse)
Catégorisation des connaissances associées au processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels *(→ Connaissances « structurales » ou relationnelles, « procédurales », « factuelles », Prise en compte de l'incertain/l'imprécision [intervalle de confiance d'une mesure])*			√	√							Kces_Appl_Mes_0.1 (rapport d'analyse)

Choix d'un formalisme approprié pour la représentation des connaissances pour chaque catégorie *(Graphes conceptuels + Théorie de la certitude de Stanford)*			√							Kces_Repr_0.1 (rapport d'analyse)
Analyse & Formalisation ontologique du processus d'application de la méthode de mesure COSMIC-FFP *(Ontologie de domaine, ontologie de tâches)*			√	√						Kces_COSMIC_0.1 (modèles documentés)
Analyse & Formalisation ontologique du processus d'application de la méthode de mesure FPA *(Ontologie de domaine, ontologie de tâches)*			√	√						Kces_FPA_0.1 (modèles documentés)
Analyse & Formalisation ontologique du processus d'application de la méthode de mesure MarkII-FPA *(Ontologie de domaine, ontologie de tâches)*			√	√						Kces_MkII-FPA_0.1 (modèles documentés)

Code : AUTO_00

Étape 3 : Automatisation des procédures de mesure

Objectif : Approche orientée ontologie pour l'automatisation d'une bonne partie du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels a partir de spécifications présentées dans un formalisme bien défini

TÂCHES ASSOCIÉES	TYPE DE TÂCHE										LIVRABLES
	RL	RD	A	C	D	Tu	I	Tg	S	X	
Modèle cognitif décrivant une procédure de mesure de la taille fonctionnelle des logiciels à partir des spécifications			√								Modèle_Cognitif_0.1 (Modèle documenté)
Détails de l'approche proposée			√								Approche_Prop_0.1 (Approche documentée)
Architecture générale d'un système d'automatisation d'une procédure de mesure de la taille fonctionnelle des logiciels à partir de spécifications présentées dans un formalisme bien défini			√	√							Auto_Arch_0.1 (modèle documenté)

Code : SYNTH_1

Étape 4 : Synthèse 1

Objectif : Rédaction des trois premiers chapitres de la thèse

TÂCHES ASSOCIÉES	TYPE DE TÂCHE										Livrables
	RL	RD	A	C	D	Tu	I	Tg	S	X	
Chapitre 1 : État de l'art et problématiques	√	√									E_A_1.0 (chapitre 1 de la thèse)
Chapitre 2 : Analyse & formalisation ontologique du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels à partir des spécifications	√	√	√								Modeles_Kces_1.0 (chapitre 2 de la thèse)
Chapitre 3 : Approche orientée ontologie [cadre théorique] pour l'automatisation de *tout ou partie* de la procédure de mesure associée a une méthode de mesure de la taille fonctionnelle des logiciels	√	√	√								Cdre_Theoriq_Auto_1.0 (chapitre 3 de la thèse)

VOLET EXPÉRIMENTAL

Code : SPECIFS_00

Étape 5 : Spécification et conception détaillées du prototype

Objectif : Spécification et conception détaillées du prototype automatisant une bonne partie du processus d'application de la méthode de mesure COSMIC-FFP à partir de spécifications présentées dans le formalisme UML standard

TÂCHES ASSOCIÉES	TYPE DE TÂCHE										LIVRABLES
	RL	RD	A	C	D	Tu	I	Tg	S	X	
Diagramme des cas d'utilisation + scénarios associés à chaque cas d'utilisation + diagrammes de séquences associés a chaque scénario + documentations associées + prototypes d'interfaces usagers			√								Analyse_0.1
Diagrammes d'états/transitions + diagramme de classes + diagramme de composants				√							Design_0.1
Architecture du système (reprendre et adapter celle proposée dans le volet exploratoire du projet)				√							Arch_Syst_0.1

Code : MAPP_00

Étape 6 : Mapping UML & COSMIC-FFP

Objectif : Mise en correspondance des concepts UML et des concepts COSMIC-FFP

TÂCHES ASSOCIÉES	TYPE DE TÂCHE										LIVRABLES
	RL	RD	A	C	D	Tu	I	Tg	S	X	
Recueil de concepts : Concepts UML et COSMIC-FFP qui seront manipulés (métamodèle UML et modèle statique de COSMIC-FFP seront utilisés). *Remarque : Uniquement les concepts UML utilisés pour décrire le logiciel à la phase d'analyse et design, relativement au Processus Unifié de Rational pour le développement de logiciels. Nous nous limitons également à la version standard de UML* ;			√								Cncpt_0.1 (listes de concepts)
Élaboration : Spécification explicite et de façon détaillée de chacun des concepts sélectionnés (les documentations respectives du métamodèle UML et du modèle statique de COSMIC-FFP seront utilisées);			√								Cncpt_Spec_0.1 (listes de concepts documentés)
Transformation (map) et inférences : Proposition de correspondances entre concepts COSMIC-FFP et concepts UML, tout au moins consolidation de celles que nous avons déjà proposées dans des travaux antérieurs [Bevo et al. 99]. Ces correspondances constituent l'essence même de la transformation (map) permettant de passer d'une description UML d'un logiciel à une description COSMIC-FFP du même logiciel;			√	√							Mapp_0.1 (rapport de mise en correspondance)
Justification : Détermination de la validité de la transformation. Pour cela nous pourrons être appelés à réaliser une ou deux études de cas : prendre un ou deux cas simples avec des spécifications selon le formalisme UML et dont on connaît la taille fonctionnelle, déterminer la taille fonctionnelle en passant par la transformation (les études de cas proposées par le groupe COSMIC et disponibles sur le site web du LRGL seront mises à profit)			√								Just_Mapp_0.1 (étude de cas)
Représentation : Trouver une façon de représenter les correspondances obtenues, dans la perspective de l'automatisation de la transformation permettant de passer d'une description UML d'un logiciel à une description COSMIC-FFP du même logiciel			√	√							Mapp_1.0 (Représentation des correspondances)

Code : CTECH_00

Étape 7 : Choix technologiques et outils

Objectif : Mise en place de l'environnement de développement

TÂCHES ASSOCIÉES	TYPE DE TÂCHE	LIVRABLES

	RL	RD	A	C	D	Tu	I	Tg	S	X	
• Plate-forme de développement • Langage de programmation • Outils nécessaires à la construction du prototype (outils programmation, SGBD, ...)	√		√								Tech_0.1 (rapport d'analyse)

Code : CONS_00

Étape 8 : Construction du prototype

Objectif : Construction du prototype automatisant une bonne partie du processus d'application de la méthode de mesure COSMIC-FFP à partir de spécifications présentées dans le formalisme UML standard

TÂCHES ASSOCIÉES	TYPE DE TÂCHE										LIVRABLES
	RL	RD	A	C	D	Tu	I	Tg	S	X	
Construction du module principal + Base de connaissances					√	√					CosmicFFPTool_1.0 (prototype de base)
Construction du module d'import des spécifications *(projet de maîtrise)*					√	√					ImportSpec_0.1 (composant du prototype)
Construction du module de documentation *(projet de maîtrise)*					√	√					Doc_Mes_0.1 (composant du prototype)
Intégration							√	√			CosmicFFPTool_1.1 (prototype de base étendu)

Code : VLD_ANA_00

Étape 8 : Validation & Analyses

Objectif : Validation du prototype construit et analyse des résultats

OBJECTIFS - TÂCHES ASSOCIÉES	TYPE DE TÂCHE										LIVRABLES
	RL	RD	A	C	D	Tu	I	Tg	S	X	
Choix d'une approche de validation [Mendes 96]	√		√								Approche_Vld_0.1
Choix des études de cas			√							√	Etudes_Cas_Vld_0.1
Validation									√	√	Res_Vld_0.1 (résultats de validation)
Analyse des résultats de la validation			√								Anal_Vld_0.1 (Analyse de résultats de validation)

Code : SYNTH_2

Étape 8 : Synthèse 2

Objectif : Rédaction des trois (derniers) chapitres de la thèse

TÂCHES ASSOCIÉES	TYPE DE TÂCHE										LIVRABLES
	RL	RD	A	C	D	Tu	I	Tg	S	X	
Chapitre 4 : Automatisation du processus d'application de la méthode mesure COSMIC-FFP à partir de documents de spécifications présentées dans le formalisme UML standard		√									Tool_0.1 (chapitre 3 de la thèse)
Chapitre 5 : Validation du prototype et analyse des résultats		√									Valid_Anal_0.1
Chapitre 6 : Synthèse des contributions, conclusion et perspectives		√									Concl_Persp_0.1