Approche multi-agents pour la mesure de la taille fonctionnelle des logiciels

Valéry Bévo, Ghislain Lévesque, Alain Abran et Jean-Guy Meunier
valery.bevo@lrgl.uqam.ca, {levesque.ghislain, abran.alain, meunier.jean-guy} @uqam.ca
L.R.G.L. – L.A.N.C.I, UQAM

Résumé

La mesure de la taille fonctionnelle des logiciels demeure une entreprise difficile et ardue. Elle fait appel à une expertise propre pour chacune des méthodes. Un début de généralisation a déjà été entrepris par le Groupe COSMIC avec la méthode de mesure COSMIC-FFP, dite de la deuxième génération des méthodes de mesure de la taille fonctionnelle des logiciels. Cette méthode identifie deux phases dans le processus de mesure : Une phase de mise en correspondance des artefacts du logiciel et une phase de mesure.

À la suite de ces travaux, nous nous proposons d'examiner la possibilité d'automatisation du processus de mesure, suivant une approche basée sur le concept d'agent : Il s'agit dans un premier temps de concevoir une formule assistée, puis éventuellement (dans une itération ultérieure) une formule autonome, pour mesurer la taille fonctionnelle des logiciels, à partir des spécifications desdits logiciels. En effet, les deux phases identifiées dans le processus de mesure peuvent être perçues comme les tâches principales associées au processus. Ces tâches principales pouvant être décomposées en sous-tâches spécialisées (identification des processus fonctionnels, identification des mouvements de données, identification des groupes de données,…). Il en découle une certaine hiérarchisation dans les tâches de mesure, chaque tâche d'un niveau supérieur utilisant les résultats fournis par certaines tâches des niveaux inférieurs. Peu importe le niveau, chacune des tâches pourrait être exécutée par un agent indépendant mais communiquant avec les autres agents impliqués dans le processus de mesure. Pour mener à bien notre travail, nous optons pour une démarche qui visera dans un premier temps à faire une traduction UML (Unified Modeling Language) des *modèles statiques*[1] des méthodes de mesure FPA, Mk II FPA et COSMIC FFP. Puis nous examinerons la possibilité de généralisation de ces modèles dans un méta-modèle. Il est à noter que les tâches et sous- tâches associées au processus de mesure auront un lien étroit avec ce méta-modèle. A partir de ce méta-modèle, nous entreprendrons ultérieurement la conception d'agents chargés selon l'une ou l'autre méthode à appliquer, d'exécuter les différentes tâches associées au processus de mesure. Dans notre exposé, nous mettrons l'accent sur la revue de littérature, les traductions UML des modèles statiques des méthodes de mesure sus mentionnées, et le méta-modèle issu de la généralisation des modèles statiques.

1. Introduction

L'estimation en tout début de projet revêt un caractère très important pour les chefs de projets. Or l'un des paramètres essentiels pour faire de l'estimation est la taille. En général, la taille peut être exprimée en points de fonction ou en lignes de codes. 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. Cependant, le logiciel étant un produit intellectuel, un « objet abstrait » [18], la mesure de sa taille en général et de sa taille fonctionnelle en particulier, n'est pas toujours chose évidente [19] et ne fait pas toujours l'unanimité. Il n'est donc pas surprenant qu'il existe, à ce jour une bonne brochette de méthodes de mesure, notamment pour la taille fonctionnelle des logiciels, même si quelques unes seulement sont émergentes (FPA, COSMIC-FFP, Mk II FPA, NESMA, …) [1, 2, 3]. Ainsi, la mesure de la taille fonctionnelle des logiciels requiert une expertise propre pour chacune des méthodes. Un début de généralisation de la mesure a

[1] Nous entendons ici par *modèle statique* 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. : Dans la littérature, le modèle statique d'une méthode est désigné sous le vocable « méta-modèle » associé à la méthode.

déjà été entrepris par le groupe COSMIC [4] avec les deux premières 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. À la suite de ce travail de généralisation nous estimons qu'il n'est pas utopique de penser à une approche basée sur le concept d'agent pour faire dans un premier temps une évaluation assistée, puis éventuellement (dans une itération ultérieure), une évaluation autonome de la taille fonctionnelle des logiciels, à partir des spécifications desdits logiciels.

Plusieurs questions peuvent alors être soulevées : Quel lien faisons-nous entre la mesure de la taille fonctionnelle des logiciels et le concept d'agent ? En quoi consiste une approche basée sur le concept d'agent pour l'évaluation assistée et éventuellement autonome de la taille fonctionnelle des logiciels à partir des spécifications ? Quelles en sont les principales étapes et quels sont les grands défis à relever ? Quels en sont les principaux avantages ? À la suite du travail de généralisation de la mesure de la taille fonctionnelle des logiciels entrepris par le groupe COSMIC, peut-on généraliser dans un méta-modèle, les modèles du logiciel associés aux principales méthodes de mesure de la taille fonctionnelle des logiciels (COSMIC-FFP, FPA et Mk II FPA) ?

À ces questions et à bien d'autres, nous comptons apporter des éléments de réponses dans le présent document. Nous commençons par un tour d'horizon rapide de la théorie sur les agents à la lumière d'une revue de littérature que nous avons effectuée. Nous jetons également un coup d'œil rapide sur la question d'automatisation du processus de mesure de la taille fonctionnelle des logiciels, à la lumière des travaux de recherche ayant porté sur le sujet. Puis nous examinons le processus général de mesure de la taille fonctionnelle des logiciels, en essayant de dégager le lien que nous faisons entre la mesure de la taille fonctionnelle des logiciels et le concept d'agent. Nous levons ensuite un pan de voile sur l'approche que nous proposons (les principales étapes et les principaux avantages). Après quoi, nous proposons une traduction UML (Unified Modeling Language) [5] des modèles du logiciel associés aux méthodes de mesure FPA, COSMIC-FFP et Mk II FPA. Enfin, nous examinons la possibilité de généralisation de ces modèles dans un méta-modèle dit « générique ».

2. La théorie sur les agents

Depuis les années 80, les agents logiciels et les systèmes multi-agents gagnent peu à peu du terrain dans les activités de recherche et de développement en génie logiciel [6, 7, 8, 9, 10, 11]. Avec le développement du web, ils suscitent de plus en plus d'intérêt, notamment dans la recherche documentaire (développement de moteurs de recherche). À la base des systèmes multi-agents on retrouve le concept d'*agent*. L'idée maîtresse ici consiste à percevoir un système du monde réel comme constitué d'un ensemble d'agents en interaction dans un environnement donné. Dans la littérature, il y a quelques divergences de vues sur la perception d'un agent. Certains auteurs confèrent à un agent des habilités mentales telles la croyance, le désir, l'intention,… Ils estiment alors que la modélisation d'un agent devrait prendre en compte la représentation de ces concepts mentaux [8]. D'autres auteurs [6] par contre se limitent à des propriétés observables et présentent un agent comme un «système » ayant les propriétés suivantes :

- *Autonomie :* Les agents encapsulent un état (non accessible à d'autres agents) et prennent des décisions basées sur cet état, sans intervention directe des humains ou autres systèmes.
- *Réactivité :* Les agents évoluent dans un environnement (monde physique, usagers via une interface graphique, collection d'autres agents, internet ou combinaison de ce qui précède), sont capables de percevoir cet environnement (via des senseurs potentiellement imparfaits) et sont capables de réagir en temps réel aux changements intervenant dans l'environnement.
- *Pro-activité :* Les agents n'agissent pas uniquement en réponse à leur environnement, mais peuvent aussi prendre l'initiative (comportement « goal-directed »).
- *Coopération :* Les agents interagissent avec d'autres agents (et éventuellement des humains) via une sorte de langage de communication agent, et sont capables de s'engager dans des activités « sociales » (coopération, négociation) en vue d'atteindre leurs objectifs.

Malgré ces divergences vues, il est à noter qu'il existe un lien très étroit entre le concept d'agent et celui d'objet [6]. La figure 1 ci-dessous l'illustre d'ailleurs assez bien :

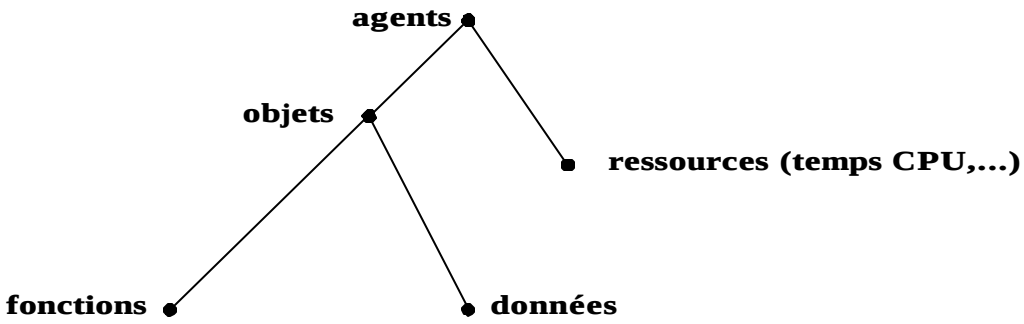


Figure 1 : Agents et objets [8]

Face à la complexité de certains systèmes ou certaines applications du monde réel (commerce électronique, bibliothèques digitales, édition coopérative, conquête spatiale, routage en réseau, gestion des ressources naturelles dans un milieu donné,…), l'approche multi-agents constitue une alternative pour *comprendre, modéliser ou simuler* de tels systèmes ou applications [8, 9]. En général, ce sont des systèmes ou applications à forte dose d'interactions entre composants, certains pouvant être indépendants.

Dans la littérature, plusieurs méthodologies sont proposées pour modéliser et développer des systèmes multi-agents : AAII Methodology (Kinny et al.), Gaia Mehtodology (Wooldridge et al.), Cassiopeia (Collinot et al.), Multi-agent System Engineering (Scott A. DeLoach) [12]. Nous avons opté pour la méthodologie MaSE (Multi-agent System Engineering), que nous trouvons assez simple (quelques étapes simples à suivre), rigoureuse (les liens entre étapes et les résultats attendus pour chaque étape sont assez bien spécifiés) et claire (le détail de chacune des étapes est précisé). Ce sont ces critères qui ont guidé notre choix. La figure 2 en résume les différentes étapes de la méthodologie MaSE.

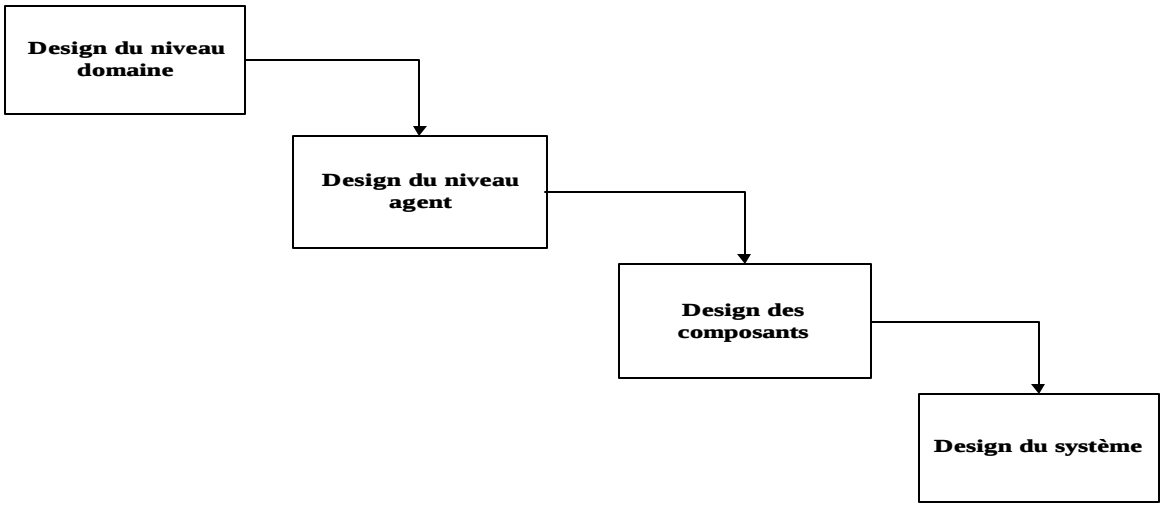


Figure 2 : Méthodologie MaSE (Multi-agent System Engineering) [12]

A la lumière de tout ce qui précède, nous allons dans la suite examiner une façon de tirer profit de la théorie sur les agents aux fins de mesure de taille fonctionnelle des logiciels. En effet, l'approche multi-agents permettant de s'attaquer à la « complexité » des systèmes [8, 9], nous pensons qu'il y a là une piste intéressante si l'on veut automatiser le processus de mesure associé à une méthode de mesure de la taille fonctionnelle des logiciels. Parlant d'automatisation du processus de mesure, nous nous intéressons à cette question dans la section suivante, à la lumière des travaux de recherche ayant porté sur le sujet. Nous abordons uniquement les grandes lignes.

3. La question d'automatisation du processus de mesure de la taille fonctionnelle des logiciels

L'implantation des systèmes de mesure de la taille fonctionnelle des logiciels rencontre bien souvent des difficultés au niveau de l'industrie. Plusieurs raisons sont avancées parmi lesquelles : La difficulté à appliquer les méthodes de mesure (ce qui rend fastidieuse la tâche du mesureur et nécessite bien souvent l'intervention d'un ou plusieurs experts malheureusement pas toujours disponibles), le manque d'outils d'assistance pour l'application des méthodes de mesure. Par rapport à ce dernier point, Sue Black & David Wigg [13] notent 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* ».

Ainsi, l'automatisation de *tout ou partie* du processus d'application d'une méthode de mesure serait à notre avis un pas 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 [14]. 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 ») [15].

Deux principales pistes de recherche existent jusqu'à ce jour, en matière d'automatisation du processus de 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 [16, 24, 25, 27], contrairement à la seconde approche pour laquelle très peu de travaux ont été consacrés [23, 29]. La grande majorité des travaux sur l'automatisation du processus de mesure de la taille fonctionnelle des logiciels pendant les vingt dernières années, ont porté sur la méthode de mesure FPA. Malheureusement toutes les tentatives d'automatisation complète du processus de mesure associé à la méthode ont été des échecs quasi complets au niveau de l'industrie, et partiels pour ce qui est des recherches académiques [27]. C'est d'ailleurs l'une des raisons (sinon la principale) ayant amené le groupe COSMIC à entreprendre le re-design des méthodes mesures 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.

À la suite des travaux du groupe COSMIC, est-il possible à ce jour d'automatiser *tout ou partie* du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels ? Pour avoir une meilleure idée des défis à relever, il convient d'examiner en détail ce processus.

4. Le processus de mesure de la taille fonctionnelle des logiciels : les tâches de mesure

L'application d'une méthode de mesure de la taille fonctionnelle du logiciel suppose deux choses essentielles : Une méthode de mesure de la taille fonctionnelle des logiciels (*modèle statique* & règles de mesure) et la documentation du logiciel à mesurer (documents de spécification, de design,…)

La tâche première du « mesureur » lors de l'application d'une méthode de mesure de la taille fonctionnelle du logiciel consiste en ce qui suit : A l'aide du *modèle statique* de la méthode de mesure, et à partir des documents de spécification du logiciel à mesurer, bâtir le modèle dudit logiciel relativement à la méthode de mesure (description du logiciel suivant un formalisme, un langage propre à la méthode de mesure : Seuls les éléments du logiciel qui sont pertinents par rapport à la mesure sont retenus).

Cette première tâche sera d'autant plus aisée pour le « mesureur » qu'il existe une correspondance (« mapping ») entre les concepts du langage associé à la méthode de mesure et les concepts du langage dans lequel est décrit le logiciel (dans les documents de spécifications). Malheureusement, cette mise en correspondance est rarement disponible, ce qui n'est pas pour faciliter le travail du « mesureur ». Ce dernier doit alors analyser les documents de spécification du logiciel, identifier ce qui est pertinent du point de vue de la mesure et ce qui ne l'est pas, en se basant sur le *modèle statique* de la méthode de mesure considérée. Le *modèle statique* de la méthode de mesure lui sert ici de « *gabarit* ». De ce travail d'analyse, va résulter une instance du *modèle statique* de la méthode, c'est-à-dire une représentation, une description du logiciel à mesurer, dans un formalisme propre à la méthode de mesure.

Dans le cas de l'automatisation du processus de mesure, il revient à un outil logiciel de faire ce travail d'analyse et de dériver une représentation, une

description du logiciel à mesurer, dans un formalisme propre à la méthode de mesure. L'outil doit alors disposer du « *gabarit* » mentionné plus haut (le *modèle statique* de la méthode de mesure). Quel est le mode de représentation le plus adéquat pour ce «gabarit » ? D'autre part l'outil doit aussi disposer d'un « mapping » (sinon il doit le réaliser) entre formalisme associé à la méthode de mesure et le formalisme dans lequel est décrit le logiciel (dans les documents de spécifications). Comment réaliser ce « mapping » ? (*Question épistémologique du « mapping », de la traduction*). Comment modéliser, représenter les résultats du « mapping » (à savoir les correspondances entre concepts) de manière à les rendre utilisables par l'outil ? (*Questions de représentation, de modélisation*).

Enfin, l'instanciation du *modèle statique* de la méthode de mesure doit tenir compte d'un certain nombre de règles (règles de mesure ou protocoles de mesure) dont certaines peuvent varier suivant l'organisation, le type de logiciel, etc. (protocoles de mesure dans un contexte spécifique). Comment tenir compte de ces règles ? Comment les représenter ? (*Question de représentation, de modélisation*).

La seconde et dernière tâche du « mesureur », lors de l'application d'une méthode de mesure de la taille fonctionnelle du logiciel, consiste en l'application des règles d'assignation numérique à l'instance du *modèle statique* de la méthode de mesure préalablement bâtie (représentation, description du logiciel à mesurer, dans un formalisme propre à la méthode de mesure), en vue de déterminer la taille fonctionnelle du logiciel à mesurer. Dans le cas de l'automatisation du processus de mesure, il revient à un outil logiciel de calculer la taille fonctionnelle du logiciel à mesurer, à partir de l'instance du *modèle statique* de la méthode de mesure préalablement bâtie et des règles d'assignation numérique proposées par la méthode de mesure. Pour ce faire, la machine doit disposer d'un modèle pour représenter les règles d'assignation numérique. Quel modèle adopter ? (*Question de représentation, de modélisation*).

Ainsi, la taille fonctionnelle d'un logiciel, 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. La particularité de ce processus est qu'il part de quelque chose d'abstrait, de conceptuel, savoir le logiciel. Ce processus est nourri par un certain nombre de « connaissances ». Ces « connaissances » doivent être spécifiées (représentées, décrites) et mises en relation. Elles sont en général relatives à la description d'un logiciel (objet de la conceptualisation, objet de la mesure). On retrouve ici des connaissances dites « déclaratives » (définitions de concepts) et des connaissances dites « procédurales » (règles d'identification des concepts et règles de mesure). Le processus de conceptualisation lui aussi doit être spécifié clairement (décrit, représenté, modélisé). En effet, le processus lui-même constitue une « connaissance » essentielle pour la mesure. 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 outil logiciel. Ces tâches faisant appel une certaine expertise (en termes de connaissances relatives à la méthode de mesure et au processus d'application de la méthode), nous pensons à la conception d'agents dits « intelligents » pour l'accomplissement de ces tâches, dans le cas de l'automatisation. À chaque agent (qui simulerait donc certaines actions du « mesureur ») serait alors affecté une tâche bien spécifique. L'environnement d'un agent serait constitué des autres agents et du contexte dans lequel la mesure est effectuée (protocoles de mesure dans un contexte spécifique). Le détail de l'approche que nous proposons ainsi que les principales étapes de sa mise en œuvre sont présentés dans la section suivante.

5. Approche multi-agents pour la mesure de la taille fonctionnelle des logiciels

L'approche que nous proposons dans le cadre de l'automatisation de tout ou partie du processus d'application d'une méthode de mesure, prend appui sur le concept d'agent dit « intelligent », soit capable d'interagir avec d'autres agents (et éventuellement des humains), capable d'apprendre, doté d'un ensemble de connaissances et capable d'inférer sur ces connaissances. Lesdites connaissances sont comme nous l'avons mentionné plus haut, relatives à la description d'un logiciel (objet de la mesure) et au processus de mesure. La première étape du processus consistera donc à décrire ces connaissances (Nous insisterons sur celles qui sont relatives à la description d'un logiciel). Les connaissances relatives à la description d'un logiciel se trouve en fait synthétisées dans ce que nous avons appelé le *modèle statique* de chaque méthode de mesure. Nous nous limiterons aux *modèles statiques* des méthodes COSMIC-FFP, FPA et MK II FPA. Nous pensons qu'il est possible de généraliser ces modèles statiques dans un méta-modèle, qui regrouperait les concepts communs et tiendrait compte des particularités de chaque modèle). Pour la description, nous ferons appel à une approche basée sur les réseaux sémantiques (un diagramme de classes en est un cas particulier) et à une notation que

nous estimons assez expressive (UML). Cette première partie du travail sera essentielle pour la définition d'un cadre pour l'automatisation de tout ou partie du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels. La figure 3 présente les principales étapes de la mise en œuvre de l'approche.

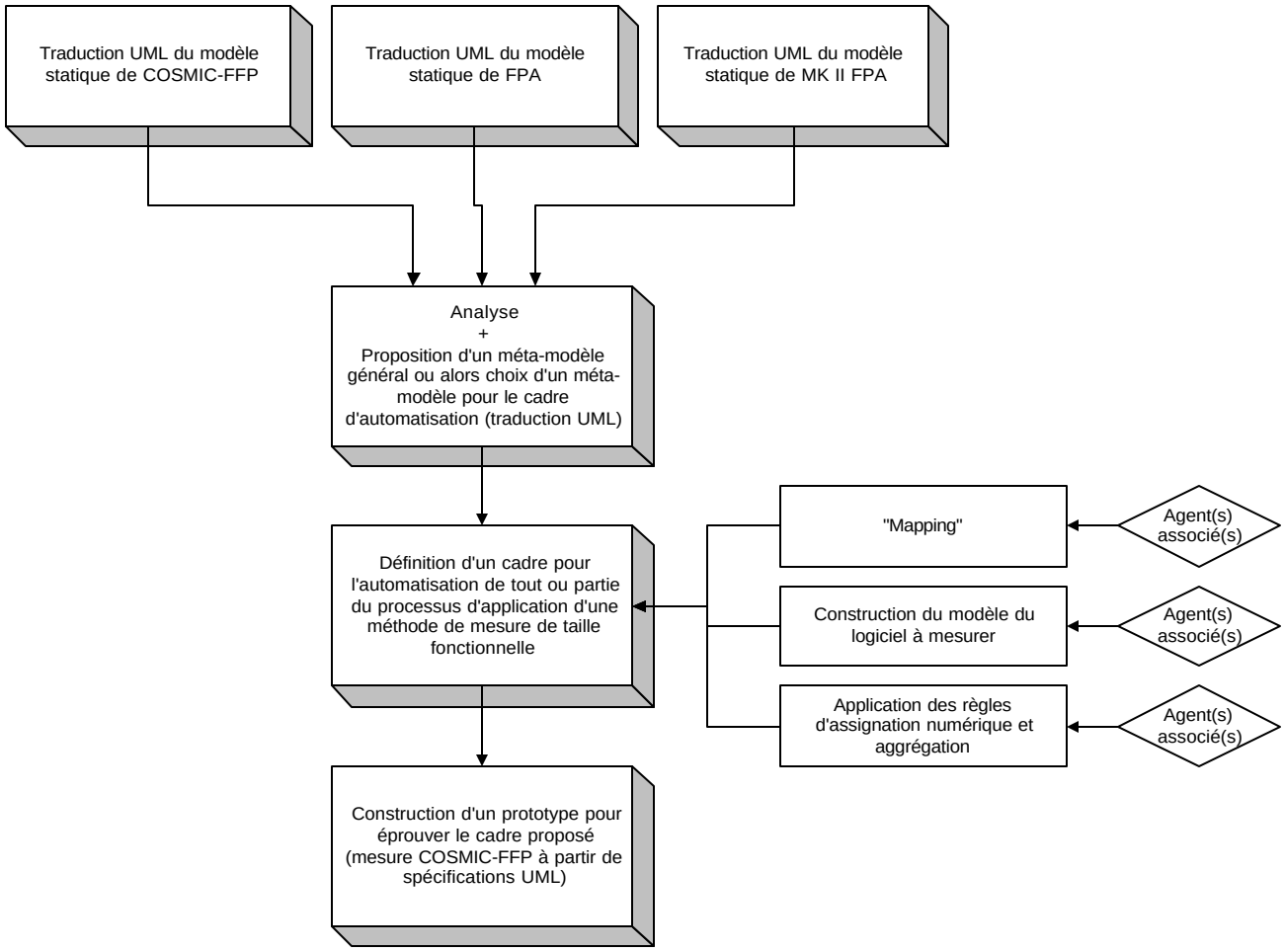


Figure 3 : Les principales étapes de la mise en œuvre de l'approche

Comme l'indique la figure 3, la définition d'un cadre pour l'automatisation de tout ou partie du processus d'application d'une méthode de mesure implique trois (3) niveaux d'abstraction: Le niveau « mapping », le niveau construction du modèle du logiciel à mesurer et le niveau application des règles d'assignation numérique et agrégation. Nous examinerons la question d'automatisation en relation avec chacun de ces niveaux. À chaque niveau, nous associerons un certain nombre d'agents « intelligents » chargés d'accomplir les tâches spécifiques au niveau. Par rapport au niveau « construction du modèle du logiciel à mesurer » par exemple, nous pensons à un agent spécialisé pour l'identification de certains concepts présents dans le modèle statique général que nous allons dans la suite (par exemple, un agent spécialisé dans l'identification des processus fonctionnels, à un agent spécialisé dans l'identification des groupes de données, à un agent spécialisé dans l'identification de mouvements de données,…). Tous les agents de ce niveau utiliseraient les résultats du « mapping » fournis par le(s) agent(s) du niveau « mapping » ainsi que les protocoles de mesure dans un contexte spécifique, et pourraient être dotés d'un *outil d'analyse textuel* pour l'identification de concepts dans les spécifications (à examiner). Pour cette partie du travail, nous nous inspirerons en partie de la méthodologie MaSE (Multi-agent System Engineering) [12], mentionnée plus haut.

Notre objectif à terme n'est pas nécessairement 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. D'où l'idée de proposer un cadre, une approche pour l'évaluation assistée, et éventuellement autonome, de la taille fonctionnelle des logiciels, à partir des spécifications.

Pour éprouver le cadre proposé, nous allons examiner le cas de l'automatisation du processus d'application de la méthode mesure COSMIC-FFP à partir de spécifications dans le formalisme UML, avec proposition d'un prototype.

5.1 Test du cadre d'automatisation

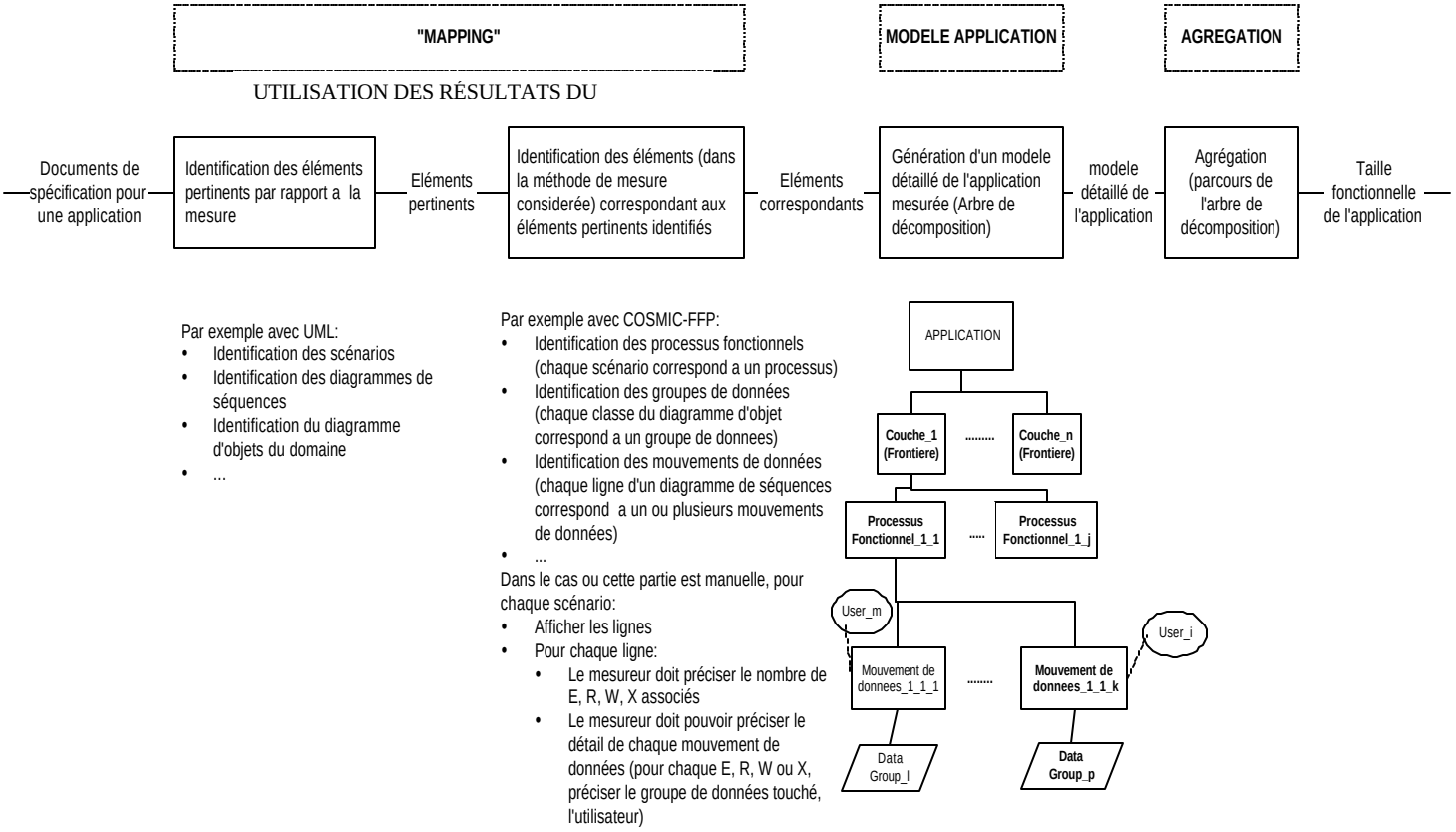


Figure 4 : Test du cadre d'automatisation proposé

Parmi les principaux avantages de cette approche, nous pouvons citer sa modularité, sa flexibilité.

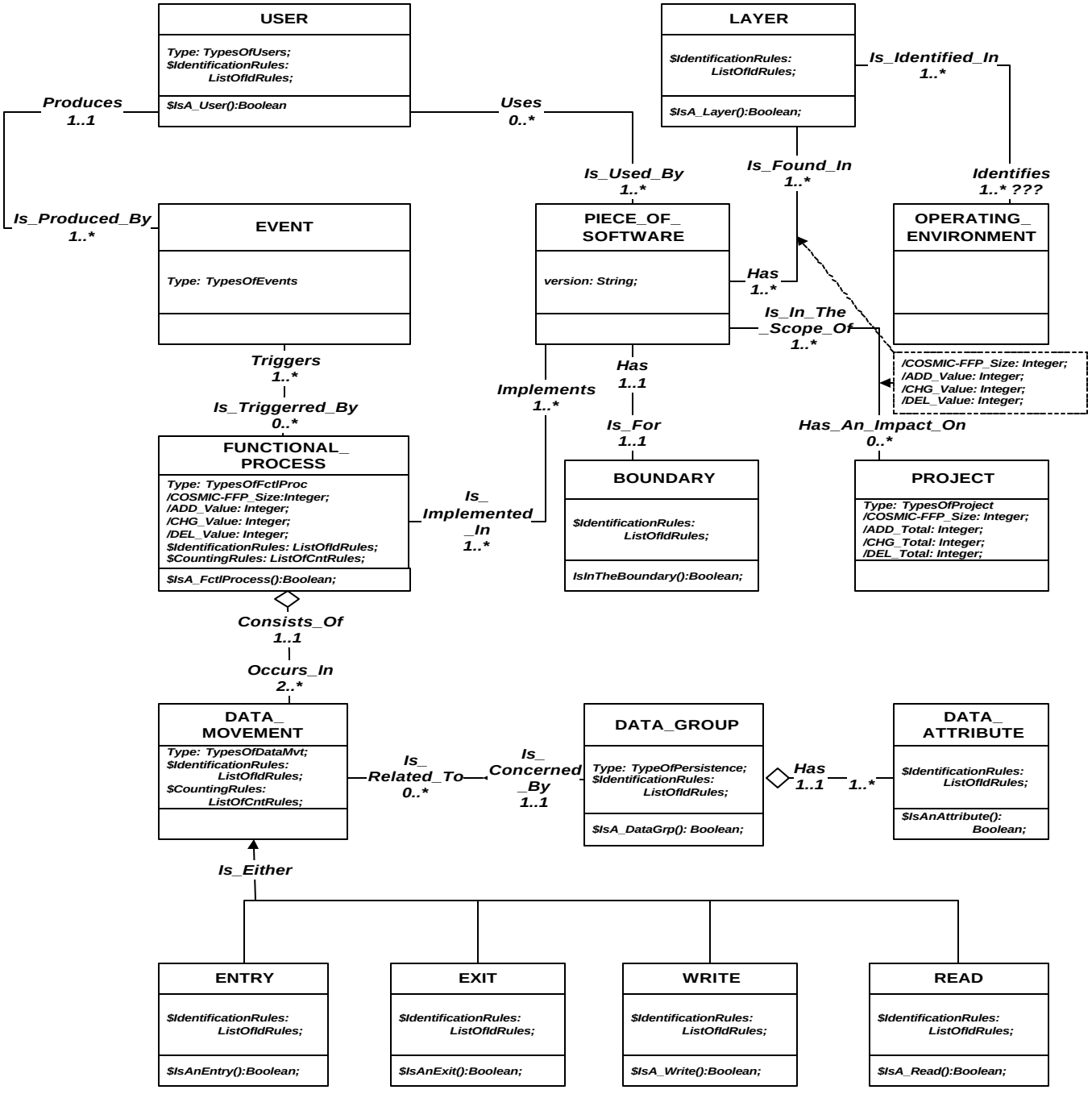
5.2 Pourquoi UML dans notre travail ?

UML (Unified Modeling Language) [5] *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 logiciel seront basés sur cette notation. UML fournit un vocabulaire et des règles assez claires, pour représenter les différents modèles permettant de comprendre (de visualiser) un système. Cette notation dont les bases ont été jetées au sein de *Rational Corporation* (1994-1996) [20], avec comme principaux précurseurs Ivar Jacobson, Grady Booch et James Rumbaugh, 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 pour les technologies de l'information. L'expressivité qui caractérise UML fait de cette notation un bon outil de communication, accessible au spécialiste tout comme au novice.

Dans la section suivante, nous vous présentons les traductions UML des *modèles statiques* des méthodes de mesure COSMIC-FFP, FPA et MK II FPA, ainsi que le modèle général dérivé de ces modèles statiques. Ceci constitue l'étape préliminaire de l'approche que nous avons présentée plus haut.

6. Traduction UML des modèles statiques des méthodes de mesure COSMIC-FFP, FPA et MARK II

6.1 Traduction UML du modèle statique de la méthode de mesure COSMIC-FFP



* TypesOfUsers = {'HumanBeing', 'PieceOfSoftware', 'EngineeredDevice'}
* TypesOfEvents = {'Timing', 'Clock', 'GraphicalInterfaceEvent', 'KeyboardEvent'}
* TypesOfDataMvt = {'added', 'deleted', 'updated'}
* TypesOfPersistence = {'Indefinite', 'Transient', 'Short'}
* TypesOfProject = {'Development', Enhancement''}
* TypesOfFctlProc = {'added', 'deleted', 'updated'}

Figure 5 : Traduction UML du modèle statique de COSMIC-FFP

6.1.1 Lecture du schéma

6.1.1.1 Les classes

Les classes identifiées sur le schéma représentent les différents concepts manipulés lors de l'application de la méthode de mesure COSMIC-FFP. Bon nombre de ces concepts doivent pouvoir être identifiés dans les documents de spécifications de logiciels lors de la première phase du processus de mesure. Le processus d'identification suit un certain nombre de règles que nous avons regroupé dans les attributs de classes nommés *$IdentificationRules*. Ces règles dépendent du formalisme dans lequel les documents de spécifications sont présentés et aussi de l'environnement dans lequel la mesure est effectuée. À certains de ces concepts sont associés un ensemble de règles, relativement à la mesure elle-même. Ces règles indiquent comment prendre en compte lesdits concepts lors de la deuxième phase du processus de mesure. Nous les avons regroupés dans les attributs de classes nommés *$CountingRules*.

6.1.1.2 Les associations

LAYER – OPERATING_ENVIRONMENT (Couche – Environnement Opérationnel Logiciel) : Dans un environnement logiciel on peut identifier au moins une couche.

LAYER – PIECE_OF_SOFTWARE (Couche – Morceau de logiciel) : Dans un morceau de logiciel (application, module, composante…) on peut identifier au moins une couche. Une même couche peut être identifiée dans plusieurs morceaux de logiciel. Il est possible de déterminer la contribution de chaque couche à la taille fonctionnelle d'un morceau de logiciel, grâce aux attributs dérivés */COSMIC-FFP_Size, /ADD_Value, /CHG_Value et /DEL_Value.*

PIECE_OF_SOFTWARE – USER (Morceau de logiciel - Utilisateur) : Un utilisateur utilise au moins un morceau de logiciel (version). Un morceau de logiciel peut être utilisé par plusieurs utilisateurs.

USER - EVENT (Utilisateur - Événement) : Un utilisateur est la source des événements qui se produisent au niveau d'un morceau de logiciel.

PIECE_OF_SOFTWARE – PROJECT (Morceau de logiciel - Projet) : Un projet a un impact sur au moins un morceau de logiciel (version). Un morceau de logiciel peut être concerné par plusieurs projets. Il est possible de déterminer la contribution de chaque morceau de logiciel à la taille fonctionnelle d'un projet, grâce aux attributs dérivés */COSMIC-FFP_Size, /ADD_Value, /CHG_Value et /DEL_Value.*

PIECE_OF_SOFTWARE – BOUNDARY (Morceau de logiciel - Frontière) : Un morceau de logiciel (version) est délimité par une frontière unique qui le sépare de façon conceptuelle des autres morceaux de logiciel de l'environnement dans lequel il opère.

PIECE_OF_SOFTWARE – FUNCTIONAL_PROCESS (Morceau de logiciel – Processus fonctionnel) : Un morceau de logiciel (version) implante au moins un processus fonctionnel. Un même processus fonctionnel peut être implanté par plusieurs morceaux de logiciels (versions).

FUNCTIONAL_PROCESS – EVENT (Processus fonctionnel - Événement) : Un processus fonctionnel est déclenché par au moins un événement (« *triggering event* »). Un même événement peut déclencher plusieurs processus fonctionnels.

FUNCTIONAL_PROCESS – DATA_MOVEMENT (Processus fonctionnel – Mouvement de données) : Un processus fonctionnel est un ensemble unique d'au moins deux mouvements de données (une entrée et une sortie).

DATA_MOVEMENT – ENTRY – EXIT – WRITE - READ (Mouvement de données – Entrée – Sortie – Écriture - Lecture) : Un mouvement de données est soit une entrée, soit une sortie, soit une écriture, soit une lecture.

DATA_MOVEMENT – DATA_GROUP (Mouvement de données – Groupe de données) : Un mouvement de données concerne un seul groupe de données. Un groupe de données peut faire l'objet de plusieurs mouvements de données.

DATA_GROUP – DATA_ATTRIBUTE (Groupe de données – Attribut de données) : Un groupe de données est un regroupement d'attributs de données (au moins un)

6.2 Traduction UML du modèle statique de la méthode de mesure FPA

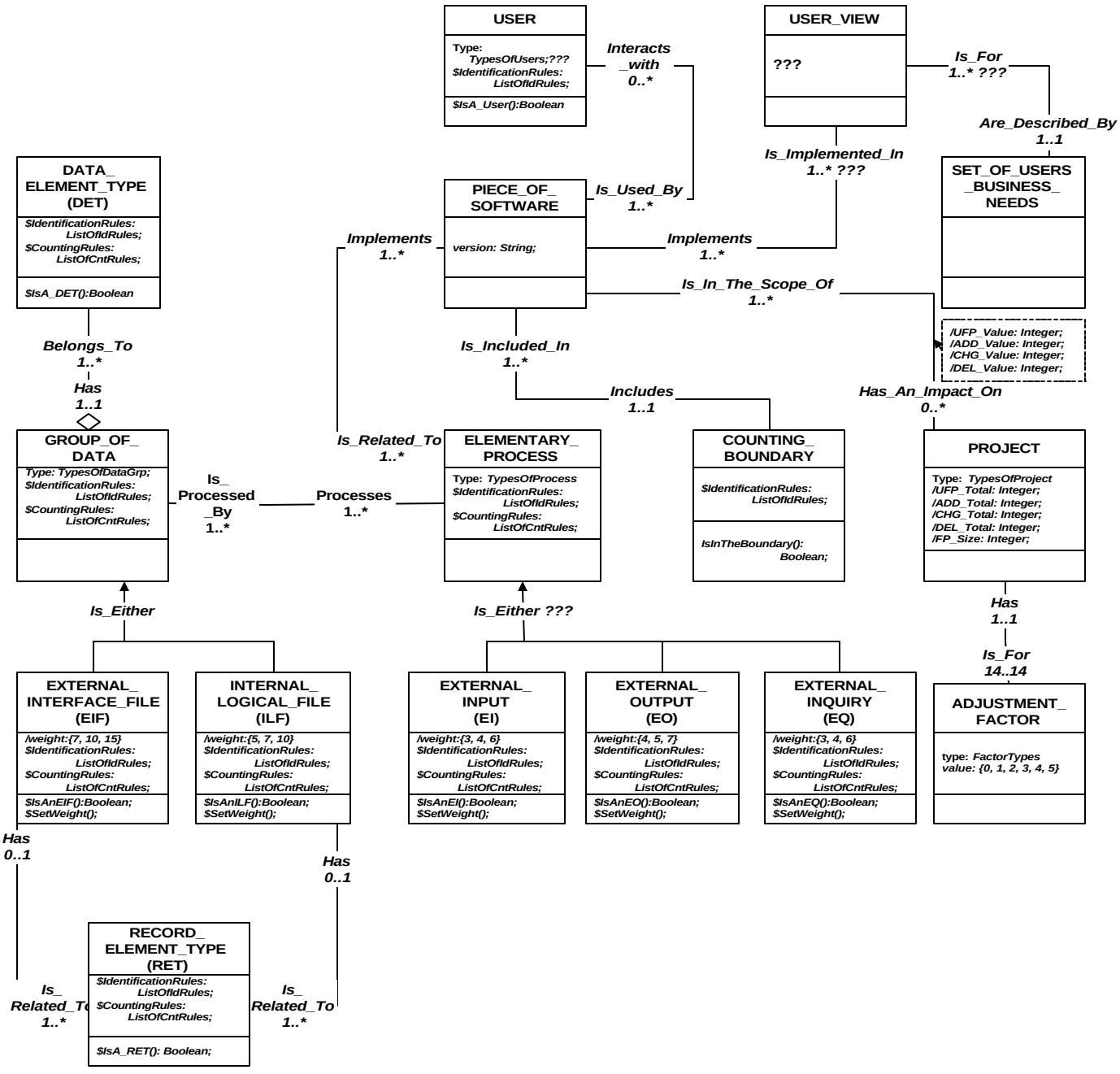


Figure 6 : Traduction UML du modèle statique de FPA

6.2.1 Lecture du schéma

Les points d'interrogations sur le modèle indiquent qu'il y a matière à apporter des clarifications au niveau de la méthode de mesure sur les concepts concernés, sans quoi il serait difficile voir impossible de les intégrer au modèle.

6.2.1.1 Les classes

Les classes identifiées sur le schéma représentent les différents concepts manipulés lors de l'application de la méthode de mesure FPA. Bon nombre de ces concepts doivent pouvoir être identifiés dans les documents de spécifications de logiciels lors de la première phase du processus de mesure. Le processus d'identification suit un certain nombre de règles que nous avons regroupé dans les attributs de classes nommés *$IdentificationRules.* Ces règles dépendent du formalisme dans lequel les documents de spécifications sont présentés et aussi de l'environnement dans lequel la mesure est effectuée. À certains de ces concepts sont associés un ensemble de règles, relativement à la mesure elle-même. Ces règles indiquent comment prendre en compte lesdits concepts lors de la deuxième phase du processus de mesure. Nous les avons regroupés dans les attributs de classes nommés *$CountingRules*.

6.2.1.2 Les associations

SET_OF_USERS_BUSINESS_NEEDS – USER_VIEW (Ensemble de besoins utilisateur – USER_VIEW) : Un 'USER_VIEW' décrit un ensemble unique de besoins utilisateur. Cependant, pour un même ensemble de besoins utilisateur, peut-on avoir plusieurs 'USER_VIEWs' ? Si oui qu'est-ce qui va faire la différence entre ces 'USER_VIEWs' ?

USER_VIEW - PIECE_OF_SOFTWARE (USER_VIEW - Morceau de logiciel) : Un morceau de logiciel (version) implante-t-il plusieurs 'USER_VIEW' (au moins un) ? Un même 'USER_VIEW' peut-il être implanté par plusieurs morceaux de logiciels ?

PIECE_OF_SOFTWARE – USER (Morceau de logiciel - Utilisateur) : Un utilisateur interagit avec au moins un morceau de logiciel (version). Un morceau de logiciel peut être en interaction avec plusieurs utilisateurs.

PIECE_OF_SOFTWARE – PROJECT (Morceau de logiciel - Projet) : Un projet a un impact sur au moins un morceau de logiciel (version). Un morceau de logiciel peut être concerné par plusieurs projets. Il est possible de déterminer la contribution de chaque morceau de logiciel à la taille fonctionnelle d'un projet, grâce aux attributs dérivés */UFP_Value, /ADD_Value, /CHG_Value et /DEL_Value.*

PROJECT - ADJUSTMENT_FACTOR (Projet – Facteur d'ajustement) : Pour chaque projet, il faut déterminer les valeurs de quatorze (14) facteurs d'ajustements définis par la méthode FPA.

PIECE_OF_SOFTWARE – CONTING BOUNDARY (Morceau de logiciel – Frontière de comptage) : La frontière de comptage inclut au moins un morceau de logiciel.

PIECE_OF_SOFTWARE – ELEMENTARY_PROCESS (Morceau de logiciel – Processus élémentaire) : Un morceau de logiciel (version) implante au moins un processus élémentaire. Un processus élémentaire est implanté par un ou plusieurs morceaux de logiciel (versions).

ELEMENTARY_PROCESS – GROUP_OF_DATA (Processus élémentaire – Groupe de données) : Un processus élémentaire manipule au moins un groupe de données. Un même groupe de données peut être manipulé par plusieurs processus élémentaires.

ELEMENTARY_PROCESS – EI – EO – EQ (Processus élémentaire – Entrée – Sortie – Interrogation) : Un mouvement de données est-il soit une entrée, soit une sortie, soit une interrogation de donnée ?

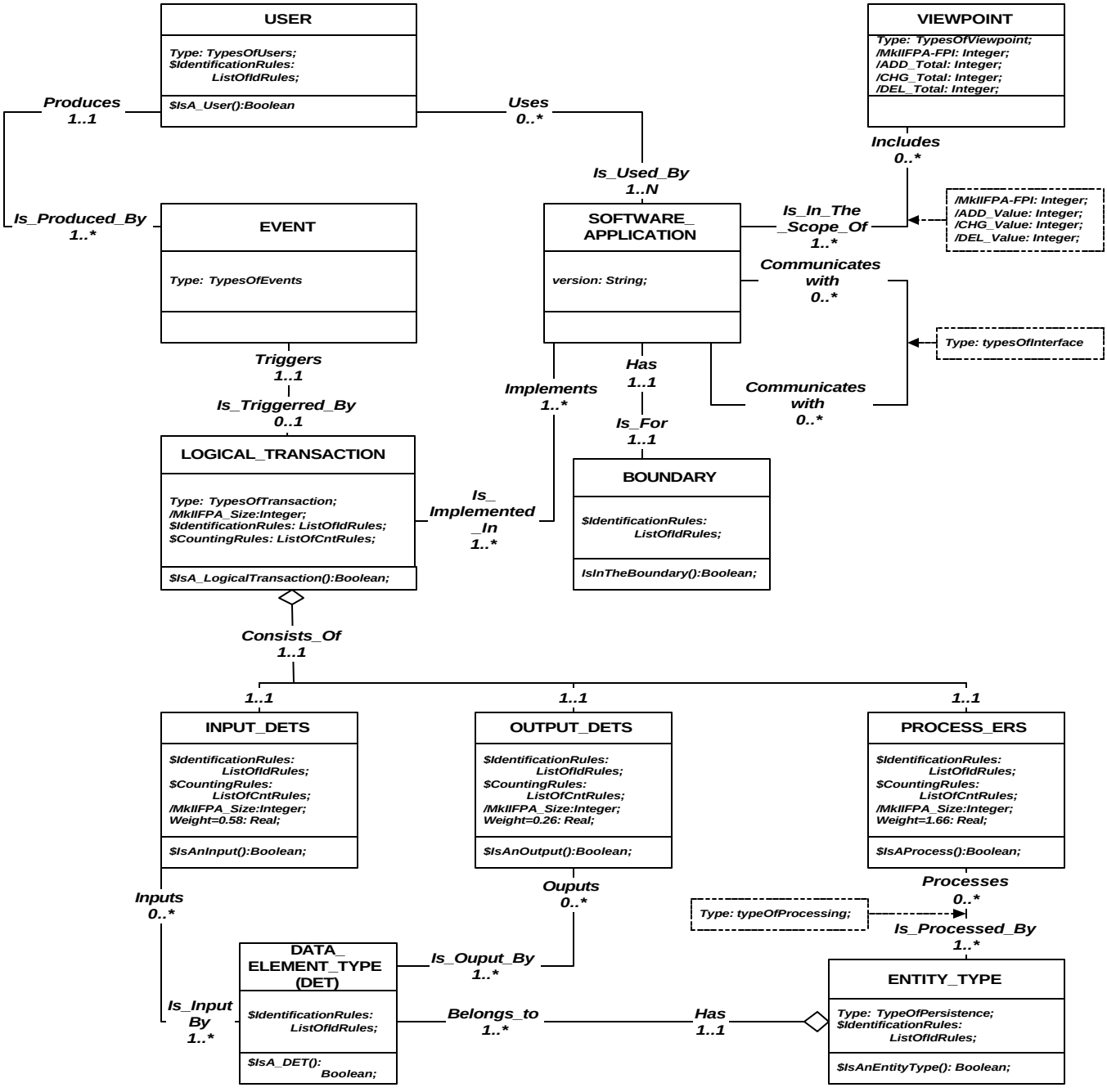
GROUP_OF_DATA – DET (Groupe de données – Élément de données) : Un groupe de données est un regroupement d'éléments de données (au moins un).

GROUP_OF_DATA – EIF - ILF (Groupe de données – EIF - ILF) : Un groupe de données est soit un EIF, soit un ILF.

EIF – RET : Un EIF peut être associé à plusieurs RET.

ILF – RET : Un ILF peut être associé à plusieurs RET.

6.3 Traduction UML du modèle statique de la méthode de mesure MK II FPA



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

Figure 7 : Traduction UML du modèle statique de Mark II FPA

6.3.1 Lecture du schéma

6.3.1.1 Les classes

Les classes identifiées sur le schéma représentent les différents concepts manipulés lors de l'application de la méthode de mesure Mk II FPA. Bon nombre de ces concepts doivent pouvoir être identifiés dans les documents de spécifications de logiciels lors de la première phase du processus de mesure. Le processus d'identification suit un certain nombre de règles que nous avons regroupé dans les attributs de classes nommés *$IdentificationRules*. Ces règles dépendent du formalisme dans lequel les documents de spécifications sont présentés et aussi de l'environnement dans lequel la mesure est effectuée. À certains de ces concepts sont associés un ensemble de règles, relativement à la mesure elle-même. Ces règles indiquent comment prendre en compte lesdits concepts lors de la deuxième phase du processus de mesure. Nous les avons regroupés dans les attributs de classes nommés *$CountingRules*. Nous nous limitons uniquement à la partie de la méthode qui traite de la mesure de la taille fonctionnelle. Par conséquent nous n'abordons pas l'aspect mesure de productivité.

6.3.1.2 Les associations

VIEWPOINT – SOFTWARE_APPLICATION (Perspective – Application Logicielle) : Une perspective (projet, gestion d'applications, entreprise) inclut au moins une application logicielle (version). Une même application logicielle peut être incluse dans plusieurs perspectives. Il est possible de déterminer la contribution de chaque application logicielle à la taille fonctionnelle d'une perspective, grâce aux attributs dérivés */MKIIFPA-FPI_Size, /ADD_Value, /CHG_Value et /DEL_Value.*

SOFTWARE_APPLICATION - SOFTWARE_APPLICATION (Application Logicielle – Application Logicielle) : Une application logicielle peut être « interfacée » avec d'autres applications logicielles. Le type d'interface correspondant est stocké dans l'attribut d'association nommé *Type.*

SOFTWARE_APPLICATION - BOUNDARY (Application Logicielle – Frontière) : Une application logicielle (version) est délimitée par une frontière unique qui le sépare de façon conceptuelle des autres applications de l'environnement dans lequel il opère.

SOFTWARE_APPLICATION - USER (Application Logicielle – Utilisateur) : Un utilisateur interagit avec au moins une application logicielle (une version). Une application logicielle (une version) peut être en interaction avec plusieurs utilisateurs.

USER – EVENT (Utilisateur - Événement) : Un utilisateur est la source des événements qui se produisent au niveau d'une application logicielle.

SOFTWARE_APPLICATION – LOGICAL_TRANSACTION (Application Logicielle – Transaction logique) : Une application logicielle (une version) implante au moins une transaction logique. Une même transaction logique peut être implantée par plusieurs applications logicielles (plusieurs versions d'une application en réalité).

LOGICAL_TRANSACTION - EVENT (Transaction logique - Événement) : Une transaction logique est déclenchée par un seul événement.

LOGICAL_TRANSACTION – INPUT_DETS – OUTPUT_DETS – PROCESS_ERS (Transaction logique - INPUT_DETS – OUTPUT_DETS – PROCESS_ERS) : Une transaction logique consiste en une entrée d'éléments de données (INPUT_DETS), un traitement sur des entités (PROCESS_ERS) et une sortie d'éléments de données (OUTPUT_DETS).

INPUT_DETS – DET (INPUT_DETS – DET) : Une entrée d'éléments de données (INPUT_DETS) fait rentrer des éléments de données à l'intérieur de la frontière d'une application. Un même élément de données peut être concerné par plusieurs INPUT_DETS.

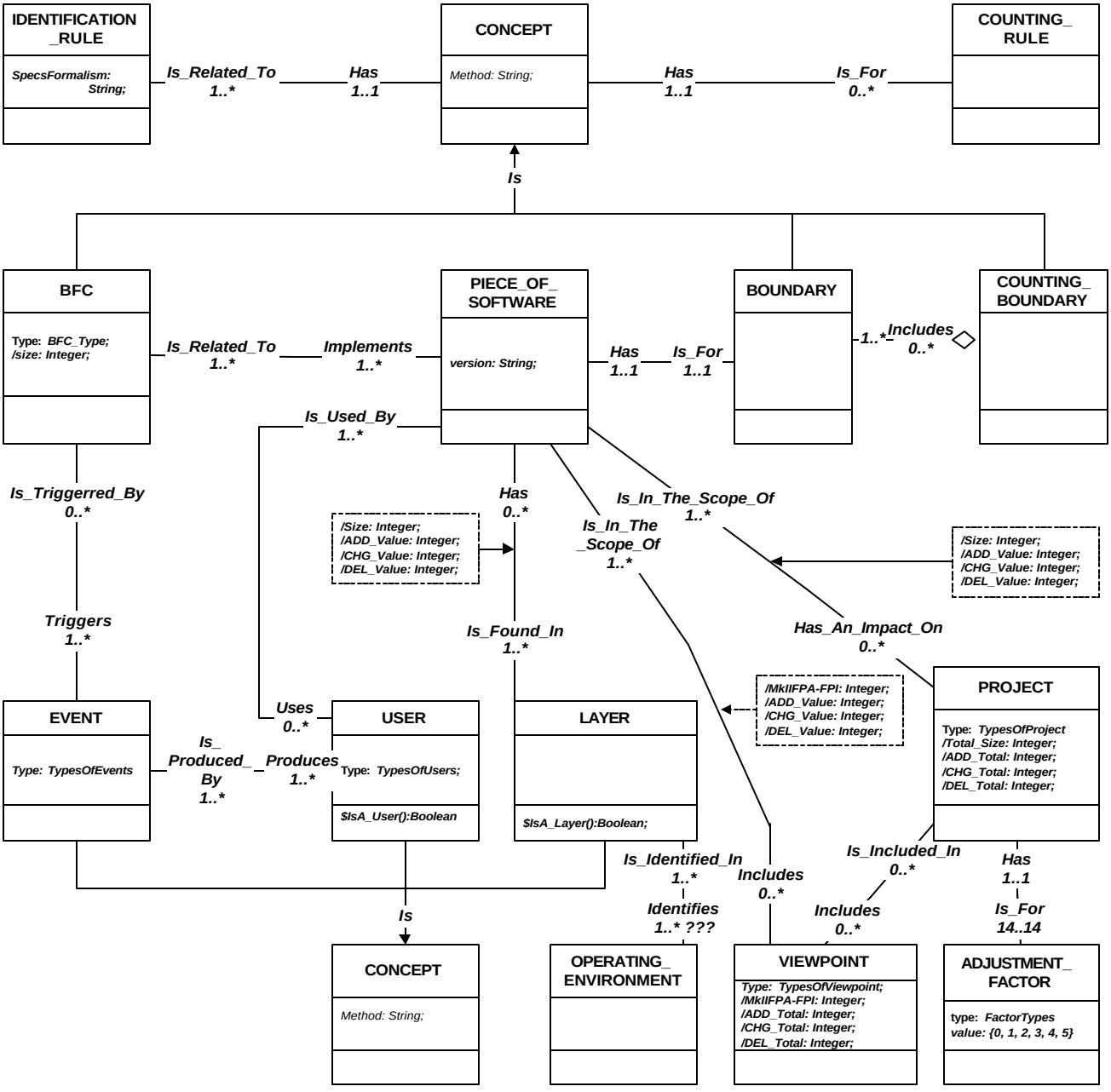
OUPUT_DETS – DET (OUPUT_DETS – DET) : Une sortie d'éléments de données (OUPUT_DETS) envoie des éléments de données hors de la frontière d'une application. Un même élément de données peut être concerné par plusieurs OUPUT_DETS.

PROCESS_ERS – ENTITY_TYPE (PROCESS_ERS –Type d'entité) : Un PROCESS_ERS effectue des traitements (création, mise-à-jour, suppression, lecture) sur des entités à l'intérieur de la frontière d'une application. Un même type d'entité peut être concerné par plusieurs PROCESS_ERS.

ENTITY_TYPE – DET (Type d'entité – DET) : Un type est un regroupement de DETs (au moins un)

Dans la section suivante, nous vous présentons une possibilité de généralisation des différents modèles examinés plus haut dans un méta-modèle. Nous nous inspirons entre autres du travail de T. Fetcke [17]. Nous faisons des hypothèses par rapport aux questions soulevées plus haut sur les différents modèles présentés.

6.4 Idée de généralisation des modèles statiques



*** TypesOfProject = {'Development', Enhancement''}**
*** TypesOfUsers = {'HumanBeing', 'PieceOfSoftware', 'EngineeredDevice'}**
*** FactorTypes = {'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'}**
*** TypesOfEvents = {'Timing', 'Clock', 'GraphicalInterfaceEvent', 'KeyboardEvent'}**

Figure 8: Traduction UML du modèle statique général (1ère partie)

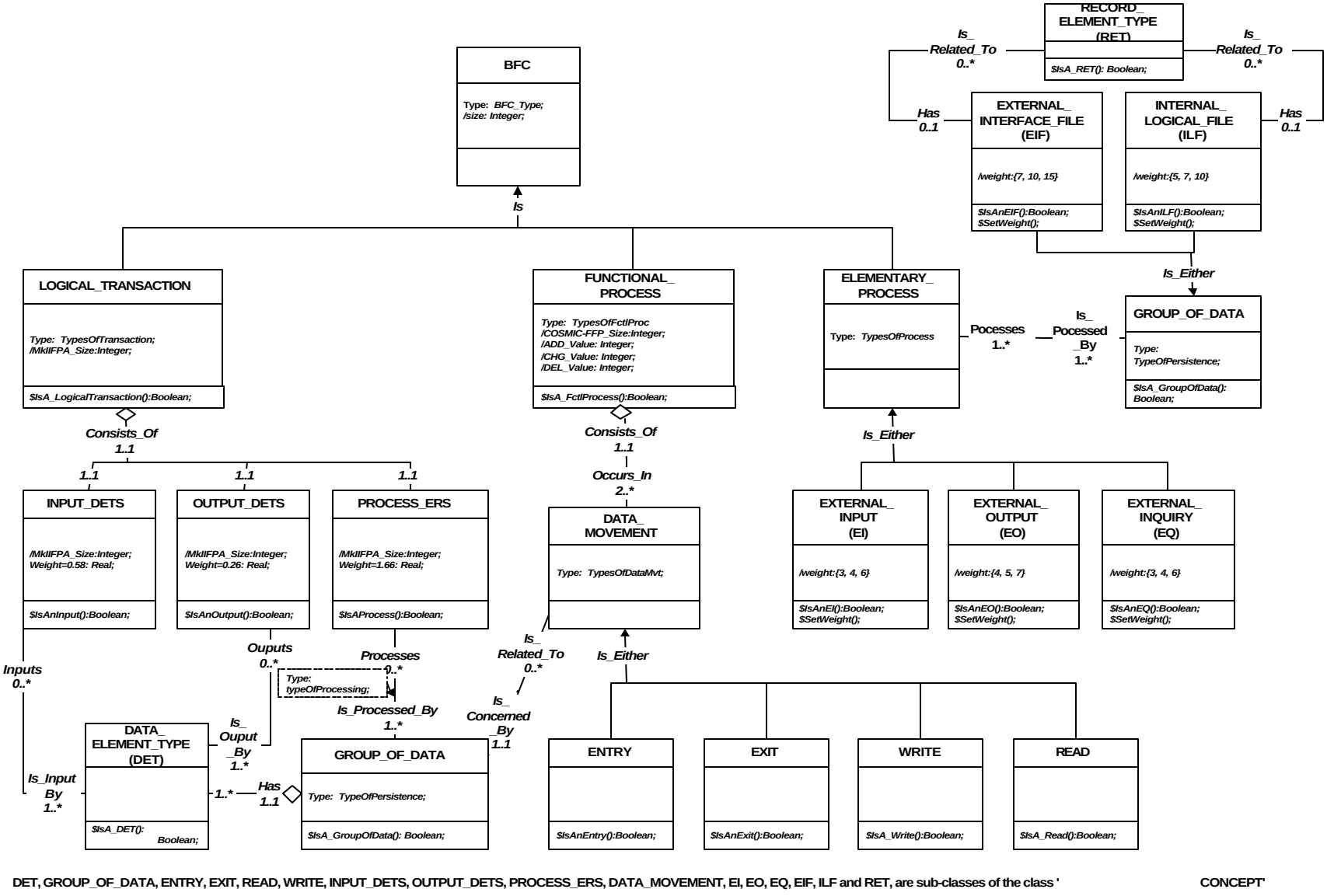


Figure 9 : Traduction UML du modèle statique général (2ème partie)

6.4.1.1 Les classes

Les classes identifiées sur les schémas représentent les différents concepts manipulés lors de l'application d'une méthode de mesure de la taille fonctionnelle des logiciels (FPA, Mk II FPA ou COSMIC-FFP). Chacun de ces concepts doit pouvoir être identifié dans les documents de spécifications de logiciels lors de la première phase du processus de mesure. Le processus d'identification suit un certain nombre de règles que nous avons regroupé dans la classe nommée *IdentificationRule.* Ces règles dépendent du formalisme dans lequel les documents de spécifications sont présentés et aussi de l'environnement dans lequel la mesure est effectuée. À certains de ces concepts sont associés un ensemble de règles, relativement à la mesure elle-même. Ces règles indiquent comment prendre en compte lesdits concepts lors de la deuxième phase du processus de mesure. Nous les avons regroupés dans la classe nommée *CountingRule*. Certains concepts sont communs à toutes les méthodes, mêmes s'ils apparaissent sous des noms différents dans les méthodes (groupe de données, attribut de données, utilisateur, …), tandis que d'autres sont propres à une seule méthode.

6.4.1.2 Les associations

CONCEPT – IDENTIFICATION_RULE (Concept – Règle d'identification) : À chaque concept manipulé lors de l'application d'une méthode de mesure de la taille fonctionnelle des logiciels, correspond au moins une règle d'identification (ces règles dépendent du formalisme dans lequel les documents de spécifications sont présentés).

CONCEPT – COUNTING_RULE (Concept – Règle de mesure) : À certains concepts manipulés lors de l'application d'une méthode de mesure de la taille fonctionnelle des logiciels, correspondent des règles de mesure (indiquent comment prendre en compte lesdits concepts lors de la deuxième phase du processus de mesure).

CONCEPT – VIEWPOINT, LAYER, BOUNDARY, COUNTING_BOUNDARY, USER, ADJUSTMENT_FACTOR, EVENT, BFC, LOGICAL_TRANSACTION, FUNCTIONAL_PROCESS, ELEMENTARY_PROCESS, INPUT_DETS, OUTPUT_DETS, PROCESS_ERS, DATA_MOVEMENT, ENTRY, EXIT, WRITE, READ, EI, EO, EQ, GROUP_OF_DATA, DET, RET, EIF, ILF : Chacun des éléments suivant est un concept propre à la mesure : un « VIEWPOINT », une couche, une frontière, une frontière de comptage, un utilisateur, un facteur d'ajustement, un événement, un « BFC », une transaction logique, un processus fonctionnel, un processus élémentaire, un « INPUT_DETS », un « OUTPUT_DETS », un « PROCESS_ERS », un mouvement de données, une entrée, une sortie, une écriture, une lecture, un « EI », un « EO », un « EQ », un groupe de données, un « DET », un « RET », un « EIF », un « ILF ».

BFC – EVENT (BFC - Événement) : Un BFC est déclenché par au moins un événement (« *triggering event* »). Un même événement peut déclencher plusieurs BFCs.

BFC – PIECE_OF_SOFTWARE (BFC – Morceau de logiciel) : Un BFC (Base Functional Component) est implanté dans au moins un morceau de logiciel. Un morceau de logiciel implante au moins un BFC.

PIECE_OF_SOFTWARE – BOUNDARY (Morceau de logiciel - Frontière) : Un morceau de logiciel (version) est délimité par une frontière unique qui le sépare de façon conceptuelle des autres morceaux de logiciel de l'environnement dans lequel il opère.

BOUNDARY – COUNTING_BOUNDARY (Frontière – Frontière de comptage) : La frontière de comptage est l'union des frontières des morceaux d'application impliqués dans le comptage.

PIECE_OF_SOFTWARE – USER (Morceau de logiciel - Utilisateur) : Un utilisateur utilise au moins un morceau de logiciel (version). Un morceau de logiciel peut être utilisé par plusieurs utilisateurs.

PIECE_OF_SOFTWARE – LAYER (Morceau de logiciel - Couche) : Dans un morceau de logiciel (application, module, composante…) on peut identifier au moins une couche. Une même couche peut être identifiée dans plusieurs morceaux de logiciel. Il est possible de déterminer la contribution de chaque couche à la taille fonctionnelle d'un morceau de logiciel, grâce aux attributs dérivés */ Size, /ADD_Value, /CHG_Value et /DEL_Value.*

LAYER – OPERATING_ENVIRONMENT (Couche – Environnement Opérationnel Logiciel) : Dans un environnement logiciel on peut identifier au moins une couche.

PIECE_OF_SOFTWARE – VIEWPOINT (Morceau de logiciel – Perspective) : Une perspective (projet, gestion d'applications, entreprise) inclut au moins une morceau de logiciel (version). Un même morceau de logiciel peut être inclus dans plusieurs perspectives. Il est possible de déterminer la contribution de chaque morceau de logiciel à la taille fonctionnelle d'une perspective, grâce aux attributs dérivés */MKIIFPA-FPI_Size, /ADD_Value, /CHG_Value et /DEL_Value.*

PIECE_OF_SOFTWARE – PROJECT (Morceau de logiciel - Projet) : Un projet a un impact sur au moins un morceau de logiciel (version). Un morceau de logiciel peut être concerné par plusieurs projets. Il est

possible de déterminer la contribution de chaque morceau de logiciel à la taille fonctionnelle d'un projet, grâce aux attributs dérivés */ Size, /ADD_Value, /CHG_Value et /DEL_Value.*

PROJECT – VIEWPOINT (Projet - Perspective) : Une perspective peut englober un ensemble de projets. Un même projet peut faire partie de plusieurs perspectives.

PROJECT - ADJUSTMENT_FACTOR (Projet – Facteur d'ajustement) : Pour chaque projet, il faut déterminer les valeurs de quatorze (14) facteurs d'ajustements définis par la méthode FPA.

BFC – LOGICAL_TRANSACTION, FUNCTIONAL_PROCESS, ELEMENTARY_PROCESS (BFC – Transaction logique, processus fonctionnel, processus élémentaire) : Un BFC (Base Functional Component) est une transaction logique, un processus fonctionnel ou un processus élémentaire.

LOGICAL_TRANSACTION – INPUT_DETS – OUTPUT_DETS – PROCESS_ERS (Transaction logique - INPUT_DETS – OUTPUT_DETS – PROCESS_ERS) : Une transaction logique consiste en une entrée d'éléments de données (INPUT_DETS), un traitement sur des entités (PROCESS_ERS) et une sortie d'éléments de données (OUTPUT_DETS).

INPUT_DETS – DET (INPUT_DETS – DET) : Une entrée d'éléments de données (INPUT_DETS) fait rentrer des éléments de données à l'intérieur de la frontière d'une application. Un même élément de données peut être concerné par plusieurs INPUT_DETS.

OUPUT_DETS – DET (OUPUT_DETS – DET) : Une sortie d'éléments de données (OUPUT_DETS) envoie des éléments de données hors de la frontière d'une application. Un même élément de données peut être concerné par plusieurs OUPUT_DETS.

PROCESS_ERS – ENTITY_TYPE (PROCESS_ERS –Type d'entité) : Un PROCESS_ERS effectue des traitements (création, mise-à-jour, suppression, lecture) sur des entités à l'intérieur de la frontière d'une application. Un même type d'entité peut être concerné par plusieurs PROCESS_ERS.

FUNCTIONAL_PROCESS – DATA_MOVEMENT (Processus fonctionnel – Mouvement de données) : Un processus fonctionnel est un ensemble unique d'au moins deux mouvements de données (une entrée et une sortie).

DATA_MOVEMENT – ENTRY – EXIT – WRITE - READ (Mouvement de données – Entrée – Sortie – Écriture - Lecture) : Un mouvement de données est soit une entrée, soit une sortie, soit une écriture, soit une lecture.

DATA_MOVEMENT – GROUP_OF_DATA (Mouvement de données – Groupe de données) : Un mouvement de données concerne un seul groupe de données. Un groupe de données peut faire l'objet de plusieurs mouvements de données.

ELEMENTARY_PROCESS – GROUP_OF_DATA (Processus élémentaire – Groupe de données) : Un processus élémentaire manipule au moins un groupe de données. Un même groupe de données peut être manipulé par plusieurs processus élémentaires.

ELEMENTARY_PROCESS – EI – EO – EQ (Processus élémentaire – Entrée – Sortie – Interrogation) : Un mouvement de données est soit une entrée, soit une sortie, soit une interrogation sur une donnée.

GROUP_OF_DATA – DET (Groupe de données – Élément de données) : Un groupe de données est un regroupement d'éléments de données (au moins un).

GROUP_OF_DATA – EIF - ILF (Groupe de données – EIF - ILF) : Un groupe de données est soit un EIF, soit un ILF.

EIF – RET : Un EIF peut être associé à plusieurs RET.

ILF – RET : Un ILF peut être associé à plusieurs RET.

7. Conclusion et prochaines étapes

Pour les prochaines étapes du travail, nous prévoyons :

- La validation des modèles présentés, question de les rendre aussi fidèles que possible à l'esprit des différentes méthodes de mesures associées.
- La définition d'un cadre général (« *general framework* ») pour l'automatisation de tout ou partie du processus d'application d'une méthode de mesure de la taille fonctionnelle des logiciels (cadre pour le « mapping », cadre pour la construction du modèle du logiciel à mesurer, cadre pour l'application des règles d'assignation numérique et la déduction de la taille)
- Le test du cadre général proposé avec la construction d'un prototype permettant de mesurer à l'aide de la méthode COSMIC-FFP, la taille fonctionnelle de logiciels dont les spécifications sont faites dans le formalisme UML.

8. Références

[1] IFPUG, Function Point Analysis Counting Practices Manual, release 4.1, Mequon, Wisconsin, 2000.

[2] UKSMA Metrics Practices Committee., MK II Function Point Analysis Counting Practices Manual., v.1.3.1, UK, September 2000.

[3] Abran, A., Desharnais, J.-M., Oligny, S., St-Pierre, D., Symons, C., COSMIC FFP - Manuel de mesures, version 2.1, Montréal, Mai, 2001. http://www.lrgl.uqam.ca/cosmic-ffp

[4] Abran, A., « FFP Release 2.0 : An implementation of COSMIC Functional Size Measurement Concepts », FESMA99, Amsterdam, October 4-7,1999.

[5] Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide, 1999.

[6] Wooldridge, M.; Ciancarini, P., « Agent-Oriented Software Engineering : The state of the Art », In *Agent Oriented software engineering : first international workshop; revised papers/AOSE 2000*, Limerick, Ireland, June 10, 2000

[7] Huhns, M., N., « Interaction-Oriented Programming », In *Agent Oriented software engineering : first international workshop; revised papers/AOSE 2000*, Limerick, Ireland, June 10, 2000

[8] Lind, J., «Issues in Agent-Oriented Software Engineering », In *Agent Oriented software engineering : first international workshop; revised papers/AOSE 2000*, Limerick, Ireland, June 10, 2000

[9] Petrie, C., « Agent-Based Software Engineering », In *Agent Oriented software engineering : first international workshop; revised papers/AOSE 2000*, Limerick, Ireland, June 10, 2000

[10] Shehory, O., « Software Architecture Attributes of Multi-agent Systems », In *Agent Oriented software engineering : first international workshop; revised papers/AOSE 2000*, Limerick, Ireland, June 10, 2000

[11] Bauer, B., Muller, P., Odell, J., « Agent UML : A Formalism for specifying Multi-agent Software Systems », In *Agent Oriented software engineering : first international workshop; revised papers/AOSE 2000*, Limerick, Ireland, June 10, 2000

[12] DeLoach, S., A., « Multiagent Systems Engineering : A Methodology and Language for Designing Agent Systems », 1999

[13] Black, S. and Wigg, D., « X-Ray : A Multi-Language, Industrial Strengh Tool », IWSM'99, Lac Supérieur, Canada, p.39, Sept.ember 8-10, 1999

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

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

[16] Ho, T.V. and Abran, A., « A Framework for Automatic Function point Counting From Source Code », IWSM'99, Lac Supérieur, Canada, p.248, Sept.ember 8-10, 1999

[17] Fetcke, T., « A Generalized Structure for Function Point Analysis », IWSM'99, Lac Supérieur, Canada, Sept.ember 8-10, 1999

[18] Naur, P. et Randel, B., « Software Engineering : A Report on a Conference sponsored by the NATO Science Committee, NATO, 1969

[19] Lawrence H. Putnam & Ware Myers, Measures for excellence: Reliable software on time, within budget, *Yourdon Press, Prentice Hall Building, Englewood Cliffs, New Jersey 07632, p.61,64, 1992*

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

[21] 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.

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

[23] Diab, H., Frappier, M., St-Denis, R., « A Formal Definition of COSMIC-FFP for Automated Measurement of Room Specifications, in FESMA 2001 ».

[24] 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.

[25] Edge, N.; Finnie, G.; 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.

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

[27] 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.

[28] Edge, N., Finnie, G., Wittig, G., Automating Function Point Approximation Counts For COBOL Legacy, in ACOSM 97, Australian Software Metrics Association, 1997.

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