UNIVERSITÉ DU QUÉBEC À MONTRÉAL

INTÉGRATION DU WEB SÉMANTIQUE DANS UN SYSTÈME D'AIDE À LA DÉCISION POUR LE GÉNIE LOGICIEL

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

SANDRINE MOUNGANG KOUAMO

OCTOBRE 2014

REMERCIEMENTS

J'adresse mes remerciements aux personnes qui m'ont aidée dans la réalisation de ce mémoire.

En particulier professeur Hakim Lounis, qui en tant que directeur de mémoire m'a guidée dans mon travail et m'a soutenue financièrement. Je remercie également professeur Petko Valtchev pour son apport et sa disponibilité.

DÉDICACE

À mes très chers parents Mr Moungang David et Mme Moungang Esther,
Mes frères et sœur Ghislain, Patrick et Raïssa,
3D, Mister B et F. Mogo Nem.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

Tableau Page

RÉSUMÉ

Avoir à sa disposition des données et des connaissances et savoir à quoi elles servent c'est bien, savoir s'en servir c'est encore mieux. La qualité est un critère recherché dans tous les domaines. Dans des domaines qui font allusion aux objets matériels, il est facile de définir, d'observer et de savoir comment obtenir un produit fini de bonne qualité. Dans le domaine du génie logiciel, il est bien plus difficile de définir et d'observer la qualité d'un produit. On fait appel aux métriques, aux normes de qualité, aux modèles de qualité, etc., pour pouvoir déterminer, évaluer et améliorer la qualité d'un logiciel. Les résultats des études empiriques et les connaissances des experts à ce sujetne sont malheureusement pas partagés avec tous les acteurs du domaine, ce qui entraine des interprétations différentes, la répétition des études ou l'ignorance de certains faits importants pour produire un logiciel de qualité. Dans le soucis de permettre et faciliter le partage des connaisances et des données sur la qualité logicielle, nous avons exploité ce que le web sémantique offre (RDF, RDFS, OWL-DL). Nous avons aussi tiré avantage du web sémantique pour encourager la communauté du génie logiciel à unir leur savoir afin d'avoir la même compréhension et interprétation des données et connaissances sur la qualité logicielle. Nous avons réalisé une ontologie qui regroupe ces connaissances et données (modèles de qualité, attributs de qualité, métriques, etc) indépendamment de leurs formats de sauvegarde. Nous avons mis cette ontologie à la disposition de tous et tout acteur avec le droit d'écriture peut apporter sa contribution à cet effort de centralisation, d'uniformisation et partage des connaissances. En faisant partie intégrante d'un système d'aide à la décision, cette ontologie est destinée à contribuer, dans les phases de conception, d'implémentation et de maintenance.

MOTS_CLÉS : modèles de qualité, métriques, web sémantique, ontologie.

INTRODUCTION

Un système d'aide à la décision pour la conception logicielle a pour objectif principal d'assister son utilisateur dans ses choix afin d'obtenir à la fin de la phase de conception et d'implémentation des livrables de qualité. Pour y arriver, notre système doit suivre un processus de raisonnement et de décision qui doit se faire à partir de données heuristiques, de données provenant d'experts et de connaissances accumulées. Dans son ensemble, le développement logiciel utilise de plus en plus des outils pour supporter l'humain dans les différentes étapes de développement. Parmi ces outils de développement, il y en a qui commence à calculer des métriques, sans expliquer les résultats des calculs, ni le rôle de ces métriques. C'est pourtant les connaissances liées à ces métriques et leurs valeurs qui sont utiles aux développeurs, afin que ceux-ci produisent des logiciels de qualité. Un autre aspect important est de pouvoir savoir comment et quand est-ce que les observations d'un projet de développement peuvent être utilisées dans un autre projet. Quelques travaux [1], [2], [3], [4], [31] ont ont été réalisés pour produire des modèles de qualité afin que ceux-ci servent de modèles pour développer des logiciels de qualité. De ces travaux, on conclut que très souvent les modèles de qualité produits par certains ne sont pas utilisés par les autres. On peut expliquer cette conclusion par les formalismes de représentation utilisés par ces modèles de qualité. Ces formalismes ne sont pas toujours compris par tous les outils. Pour permettre donc l'utilisation des connaissances dans différents contextes on doit les formaliser de façon à faciliter leur compréhension.

Avec la diversité des langages de programmation orientés-objets et les différents formats de documents à traiter, il y a le problème de compatibilité et d'interopérabilité qui se pose entre les données que l'on peut extraire des codes sources de différents projets. La construction

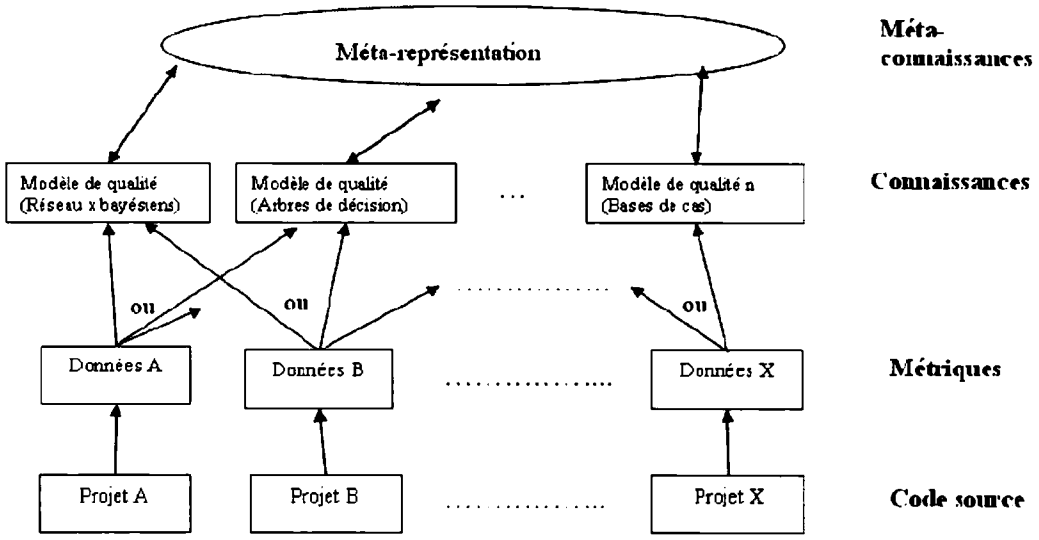
d'un pont entre la richesse des connaissances générées par les activités empiriques du génie logiciel et le besoin grandissant d'utiliser ces connaissances de manière efficace pendant le développement logiciel, requiert une approche où les chercheurs et les développeurs partagent leurs connaissances sans les contraintes imposées par les structures des outils actuels. Nous nous concentrerons sur le problème d'interopérabilité, car il aborde des aspects importants tels que le partage des connaissances, la communication matérielle et humaine. Pour pouvoir partager des connaissances représentées et enregistrées sous différents formats, il faudrait trouver une façon de transformer ces dernières de manière à ce qu'elles soient indépendantes du format d'enregistrement, de l'outil utilisé pour extraire et transformer les connaissances, et du langage.

Le web sémantique se trouve être la solution à ce problème, car il permet l'interopérabilité entre les données en créant un vaste espace distribué où les utilisateurs peuvent publier et accéder aux informations de différentes sources. Il a fait ses preuves dans divers domaines comme le domaine biomédical, l'internet (le partage et la fouille d'informations), etc. Selon les règles du web sémantique, on doit donc créer une nouvelle ontologie ou adapter une ontologie déjà existante à notre besoin. Notre besoin est de formuler des informations sur les connaissances que nous disposons actuellement. Ces méta-connaissances pourront ainsi être exploitées sans contraintes de langages, de formats ou d'outils. L'objectif principal de notre travail est donc de contribuer à la réduction du fossé entre la production des connaissances et leur utilisation efficace dans le domaine du développement logiciel. Notre contribution est de permettre l'interopérabilité entre les entités capables d'interpréter des termes, des concepts et des ressources.

Pour cela, nous allons publier les connaissances de façons structurées et fournir un cadre de travail qui permet de partager et d'utiliser ces connaissances structurées dans un format interprétable par la machine. Nous allons aussi produire un inventaire de tous les ingrédients logiciels qui permettront à une entité de comprendre les connaissances produites par une

autre entité (vocabulaires sémantiques, méta-données, etc.). La figure suivante résume sommairement notre objectif:

Figure 0.1 Présentation du problème

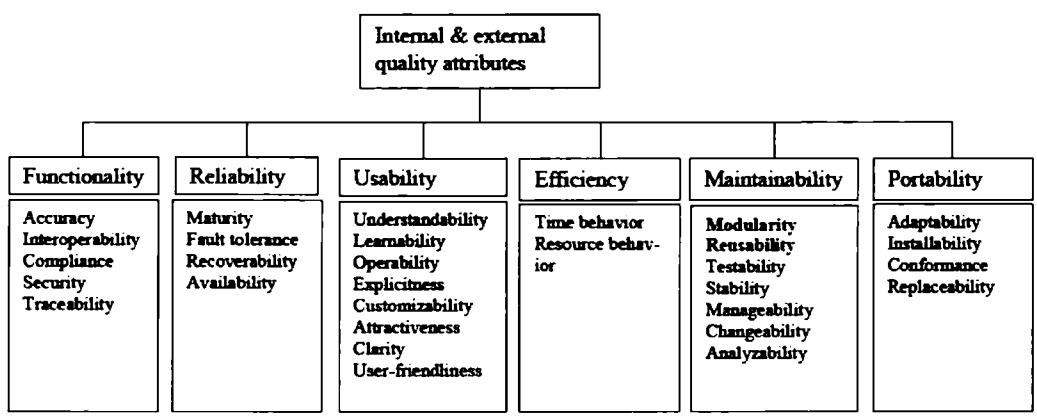


Si dans le cadre d'un projet A on veut, en plus des connaissances obtenues du projet A, exploiter les connaissances du projet B, on doit pouvoir transformer la méta-représentation des connaissances du projet B sous la forme de la représentation1. On suppose que la représentation1 (représentaion2) a été utilisée sur les données A (données B) du projet A (projet B), avec représentation1 = représentation2. La représentation1 peut être une représentation sous forme de règles et représentaion2 peut être une représentation sous forme d'un réseau de neurones.

Notre travail se concentrera sur la partie supérieure de cette figure à savoir la méta-représentation des connaissances ou méta-connaissances. D'après ISO 9126, les attributs internes et externes de la qualité logicielle peuvent être regroupés comme le montre la figure

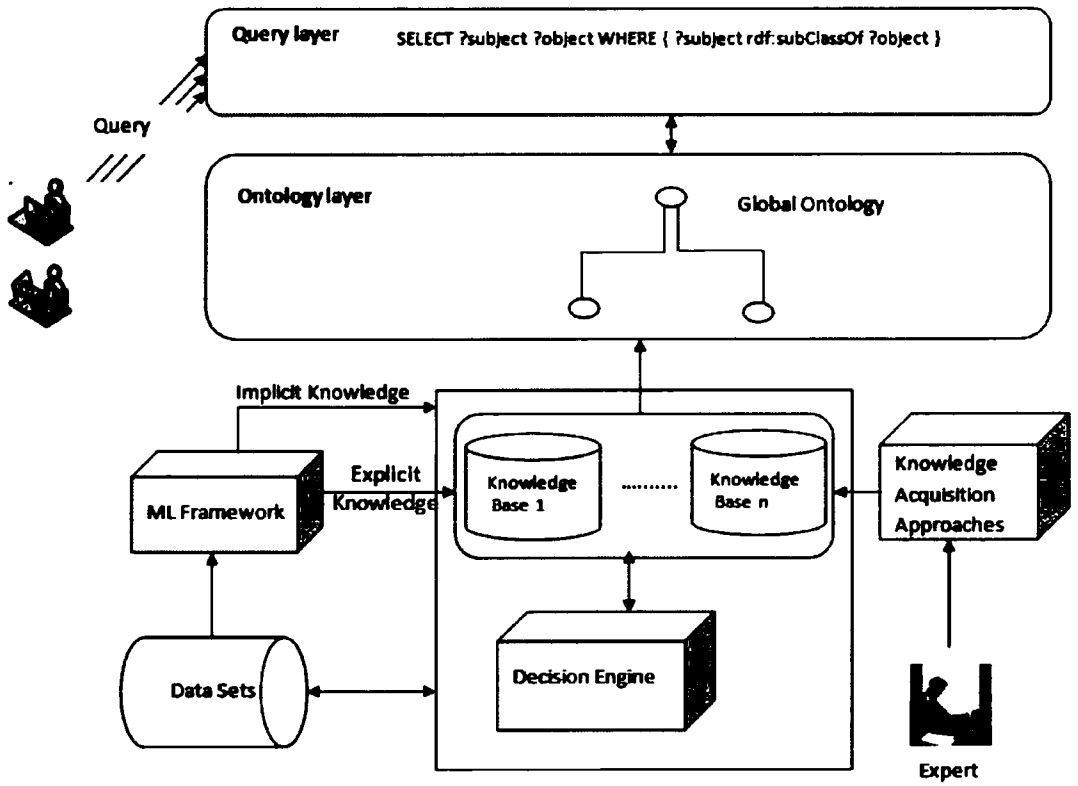
0.2. De quoi sont composés les attributs de qualité qui nous intéressent? Comment les évaluer et les mesurer ? Quels sont les modèles de qualité communément utilisés pour évaluer la qualité logicielle ? etc. Ce sont autant de questions auxquelles il faut répondre afin de pouvoir qualifier de bonne, la qualité d'un logiciel en service et garantir la bonne qualité d'un logiciel en développement. Afin de faciliter les réponses à ces questions et de permettre à tous les acteurs du domaine du génie logiciel d'avoir des réponses claires et non contradictoires, il faudrait que ces acteurs aient la même compréhension des termes, expressions, formules et notions qui concourent aux réponses à ces questions. L'attribut d'un logiciel qui a une grande influence sur sa qualité est la maintenabilité, car un logiciel passe la majeure partie de sa vie en phase de maintenance. Plus il est difficile à maintenir, plus il est coûteux [46]. Pour cette raison, nous nous focaliserons sur les connaissances et données qui se rapportent à la maintenabilité d'un logiciel et plus précisément à deux de ses sous-attributs, que sont la modularité et la réutilisabilité. Il faut toutefois garder à l'esprit que la maintenabilité a une influence sur les choix de conception et d'implémentation. En se focalisant donc sur les connaissances et données se rapportant à la maintenabilité, on se focalise aussi sur les connaissances et données qui portent sur les phases de conception et d'implémentation.

Figure 0.2 Attributs internes et externes de qualité



Pour faciliter le partage des connaissances entre les experts en génie logiciel ou entre développeurs, il faut donc permettre l'uniformisation et la centralisation des connaissances sur la qualité d'un logiciel (développement orienté-objets). Pour cela, on utilisera les ontologies, car elles permettent de définir un vocabulaire propre à un domaine précis. Ce vocabulaire ainsi défini garantit une compréhension unique et claire du domaine concerné. Notre problématique se résume à comment faciliter le partage des connaissances entre les différents niveaux d'utilisateurs du génie logiciel? Notre solution se résume à l'architecture présentée à la figure 0.3 qui ajoute à l'architecture de départ deux couches (la couche ontologique et la couche de requêtes). Les connaissances contenues dans les bases de connaissances proviennent des experts du domaine et de l'apprentissage automatique. À partir d'ensembles de données, des méthodes d'apprentissage automatique permettent d'enrichir les bases de connaissances obtenues explicitement et implicitement. Le contenu des bases de connaissances va être transformé en une ontologie globale qui sera interrogée au besoin par les utilisateurs du système.

Figure 0.3 Système d'aide à la décision pour le génie logiciel.



La suite de notre document est répartie comme suit : le chapitre 1 présentera les modèles de qualité en génie logiciel. Ces modèles de qualité prédictifs permettent d'obtenir et de réutiliser les connaissances nécessaires pour le développement de logiciels de bonne qualité. Le chapitre 2 portera sur la notion d'ontologie et les processus de développement d'une ontologie. Dans ce chapitre, nous présenterons les différents processus existants et nous expliquerons le choix du processus utilisé pour développer notre ontologie. Dans le chapitre 3, nous parlerons des différentes étapes de création de notre ontologie et nous présenterons notre ontologie ainsi que les « provenance méta-data » qui permettent d'enrichir notre ontologie de références et donnent ainsi la possibilité de vérifier l'exactitude des instances et des concepts de notre ontologie. Le chapitre 4 quant à lui se focalisera sur l'implémentation,

l'évaluation et l'interrogation locale et publique de notre ontologie: comment interroger une ontologie localement et sur le web? Existe-t-il déjà des moyens facilitant la publication et l'interrogation d'ontologies sur le web? Quels sont les besoins sur le plan technique ? Nous terminerons par une conclusion et en annexe la documentation complémentaire de notre ontologie.

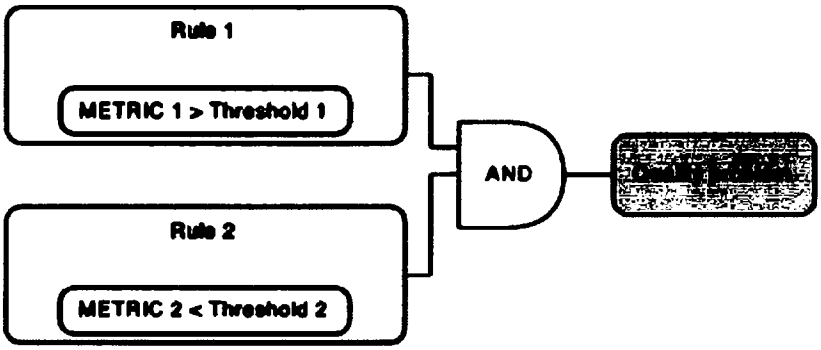
CHAPITRE I

MODÈLES DE QUALITÉ EN GÉNIE LOGICIEL

Au cours de la planification d'un projet logiciel, la prise en compte de la qualité est tout aussi importante que la planification des différentes activites et ressources. Cependant, planifier la qualité d'un produit logiciel sous-entend pouvoir estimer sa propension aux fautes, la facilité à le maintenir, sa capacité à être réutilisé, sa capacité à pouvoir interagir avec d'autres logiciels au besoin et l'effort que tout cela requiert. Pour pouvoir prédire tous ces facteurs, il faut collecter des métriques et des données pertinentes sur lesquelles appliquer ces métriques. Plus les codes sources sont volumineux et font parti de systèmes complexes en utilisation, plus ils sont pertinents comme données. Il en est de même pour les modèles conceptuels. C'est dans ce souci de prise en compte de la qualité d'un produit logiciel que des experts du domaine ont réalisé des expériences et observé les relations et dépendances entre les métriques et les différents facteurs importants pour la qualité logicielle. De ces travaux, ont été adoptés des modèles de prédiction de la qualité encore appelés modèles de qualité. Les résultats de chaque groupe d'experts sont ainsi sauvegardés dans le modèle de qualité de leur choix. Ils pourront ainsi utiliser et réutiliser ces résultats pour prédire et obtenir un système de bonne qualité. Ces modèles de qualité sont représentés sous la forme de réseaux de neurones, réseaux bayésiens, arbres de décision, règles, bases de cas, modèles paramètres (COQUALMO, COCOMO), etc. Nous allons, dans la suite de cette section, présenter les modèles de qualité les plus utilisés, à savoir : les

arbres de décisions, les règles et les réseaux bayésiens. La figure 1.1 nous montre comment est-ce que les règles sont utilisées pour détecter un problème de qualité d'un logiciel. Cette stratégie est la même pour les deux autres modèles de qualité.

Figure 1.1 Stratégie de détection de la mauvaise qualité d'un logiciel [39].



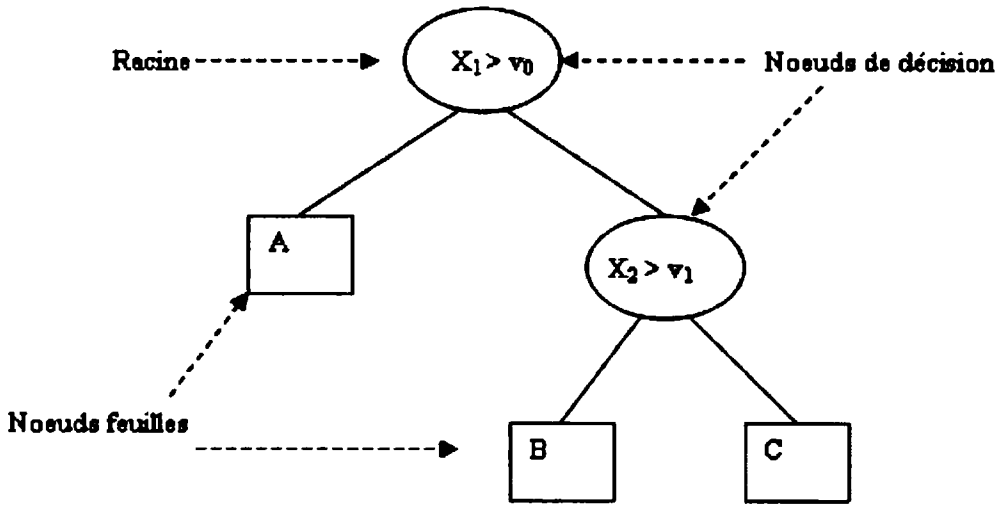
1.1 Les arbres de décision

1.1.1 Concepts

Un arbre de décision est un modèle hiérarchique où les régions locales sont identifiées à la suite d'une séquence récursive de division. Ce modèle utilise la stratégie diviser et régner et est utilisé pour l'apprentissage supervisé. Un arbre de décision est composé de nœuds de décision et de nœuds feuilles. Chaque nœud de décision *m* exécute une fonction de test $f_m(x)$ dont les résultats discrets représentent les noms des branches de l'arbre [7]. Après l'exécution de la fonction de test par un nœud de décision, le chemin emprunté est celui dont le nom ou le label de la branche sortante correspond au résultat de la fonction de test. Ce processus se répète pour le nœud suivant jusqu'à atteindre un nœud feuille. La valeur contenue dans la feuille est

le résultat de l'arbre de décision. La figure 1.2 illustre un exemple d'arbre de décision.

Figure 1.2 Exemple d'arbre de décision.



Les arbres de décision sont utilisés beaucoup plus pour la classification que pour la régression. Dans le cas de la classification, un nœud feuille détermine parmi les valeurs d'entrée un intervalle pour lequel les instances ont le même label. Dans le cas de la régression, un nœud feuille détermine parmi les valeurs d'entrée (données d'apprentissage) l'intervalle pour lequel les instances ont des valeurs de sortie numériques très similaires. L'arbre de décision doit son succès sans doute à sa facilité d'interprétation, sa transparence (les étapes qui permettent d'aboutir à un résultat sont claires), sa facilité à pouvoir être transformé en règles et son exactitude dans plusieurs domaines [45]. On assiste cependant à une dégradation de sa performance lorsqu'on a un grand nombre de classes.

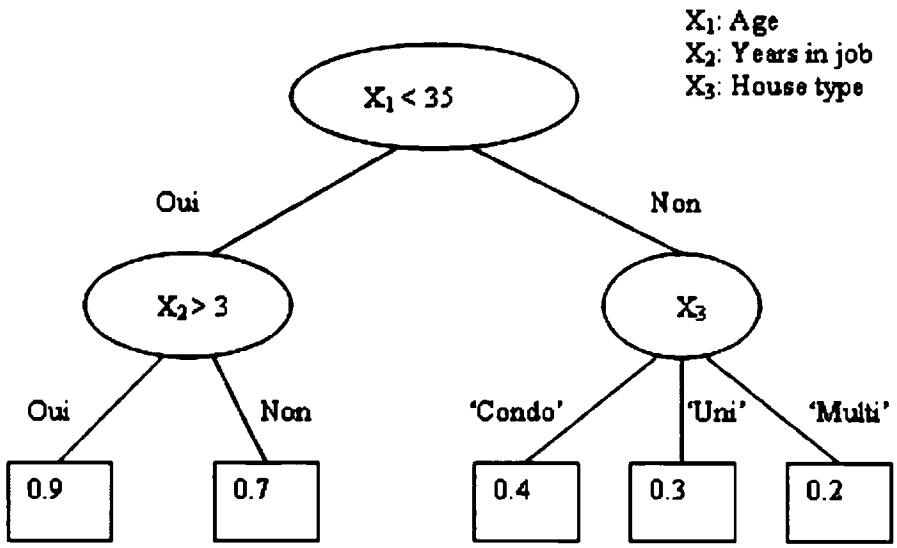
1.1.2 Types d'arbres de décision

En fonction du nombre de variables utilisées par la fonction de test de chaque nœud de décision, on distingue trois types d'arbres de décision. Si la variable est numérique, on a deux branches sortantes par nœud. Si la variable est discrète, on a n branches sortantes; n étant le nombre de valeurs que la variable peut avoir. Par exemple, si la variable est couleur avec $couleur \in \{bleu, blanc, rouge\}$, $n = 3$.

Les trois types d'arbres de décision sont :

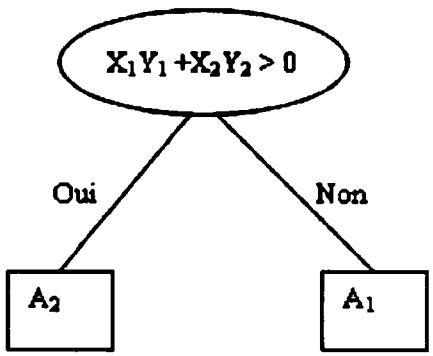
a. Les arbres univariants. Dans un arbre univariant, la fonction de test de chaque nœud de décision utilise une seule variable. On dit que l'arbre est composé de nœuds de décision univariants.

Figure 1.3 Exemple d'arbre univariant.



b. Les arbres multivariants. Dans un arbre multivariant, la fonction de test de chaque nœud de décision utilise plusieurs variables. On dit que l'arbre est composé de nœuds de décision multivariants.

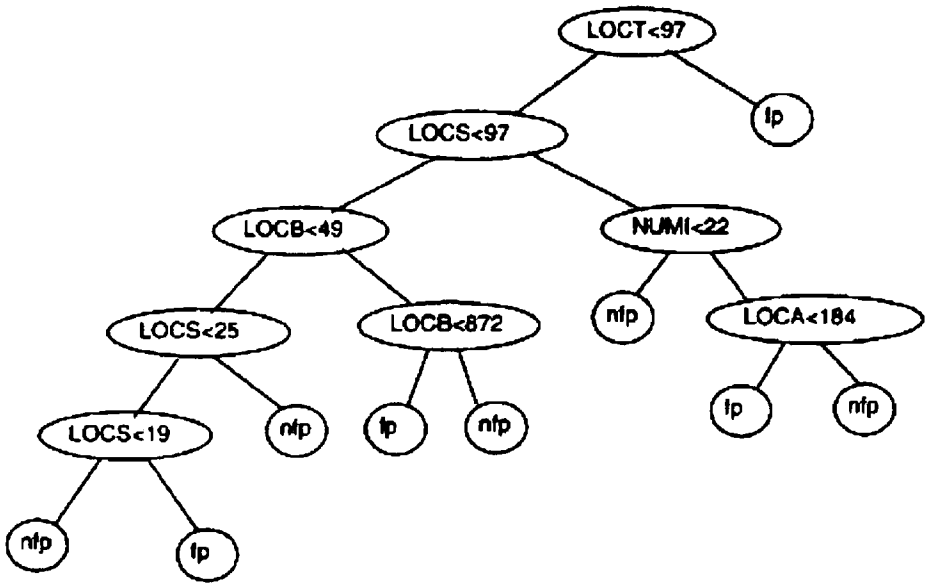
Figure 1.4 Exemple d'arbre multivariant.



c. Les arbres omnivariants ont une architecture hybride avec des nœuds univariants et multivariants.

Dans le cadre de notre problématique, notre modèle se base sur des métriques pour prédire la qualité d'un système par rapport à un attribut de qualité non mesurable comme la propension aux fautes, l'impact du changement, etc. Il existe des algorithmes (C4.5, J48, ID3, CART, SPRINT, etc.) qui implémentent ces modèles de qualité.

Figure 1.5 Arbre de décision basé sur le nombre de fautes [48].



La figure 1.5 est un exemple d'arbre de décision sur la qualité logicielle. Comme nœuds de décision, on a les métriques : nombre de fois que le fichier source a été inspecté avant la phase de test du système (NUMI), nombre de lignes de code du fichier source avant la phase de test du système (LOCT), nombre de lignes de commentaires du fichier source avant la phase de test du système (LOCS), nombre de lignes de code du fichier source avant la phase de codage (LOCB) et nombre de

lignes de commentaires du fichier source avant la phase de codage (LOCA). Comme feuilles, on a les attributs propension aux fautes (fp) et non propension aux fautes (nfp).

1.2 Les réseaux bayésiens

1.2.1 Concepts

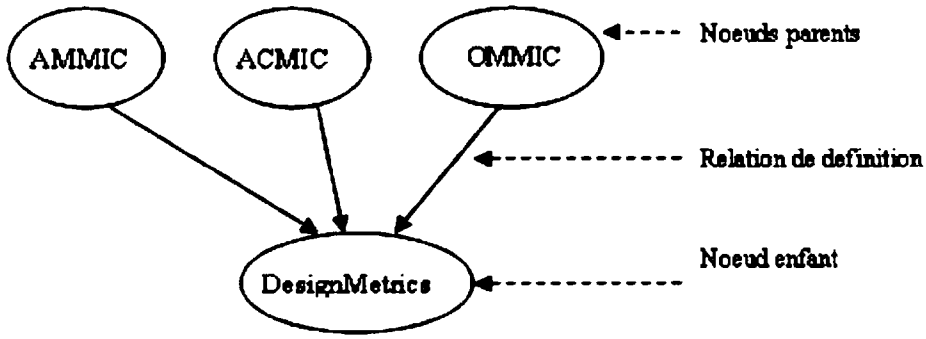
Un réseau bayésien est composé d'un graphe (aussi appelé structure) et de plusieurs tables de probabilités, avec une table de probabilité par nœud. Les réseaux bayésiens sont le résultat de la fusion entre la théorie des graphes et la théorie des probabilités. Les tables de probabilités sont basées sur le théorème de Bayes qui stipule que si A et B sont deux évènements et si on connait la probabilité de A, de B et de B connaissant A, alors on peut déterminer la probabilité de A connaissant B :

$$\textbf{Formule : } P(A/B) = \frac{P(B/A) \cdot P(A)}{P(B)}$$

Les valeurs comprises dans la table de probabilités d'un nœud parent influencent ou définissent les valeurs de la table de probabilités de ses nœuds enfants. Chaque nœud parent à une relation de causalité ou de définition avec son nœud enfant. On parle de relation de définition entre deux nœuds lorsque le nœud enfant indique à quelle catégorie ou à quel type appartient les variables représentées par ses nœuds parents. Dans ce cas, les valeurs de la table de probabilité du nœud parent définissent les valeurs de la table du nœud enfant.

Exemple (relation de définition, nœud parent/enfant) :

Figure 1.6 Exemple de structure d'un réseau bayésien.



On parle de relation de causalité lorsqu'il y a une relation de cause à effet entre le nœud parent et son enfant. Dans ce cas, les valeurs de la table de probabilité du nœud parent influencent les valeurs de la table du nœud enfant. Les nœuds du graphe d'un réseau bayésien sont regroupés en nœuds d'entrée, nœuds intermédiaires et nœuds de sortie ou feuilles. Les valeurs des tables de probabilité représentent les différents états que peuvent avoir les variables représentées par les nœuds ainsi que la probabilité de ces états. En respect avec la théorie des probabilités, la somme des probabilités est une distribution de probabilité. Par exemple, un nœud peut représenter la propension aux fautes d'un logiciel, cette propension peut être faible, moyenne ou forte. Faible, moyenne et forte sont donc les différentes valeurs ou états que la variable propension aux fautes peut avoir. Les données de la table de probabilités des nœuds d'entrée sont fournies par celui qui construit le réseau. Ces données permettront de construire automatiquement les tables de probabilités des nœuds intermédiaires et de sortie en appliquant le théorème de Bayes.

1.2.2 Types de réseaux bayésiens

On distingue deux catégories de réseaux bayésiens :

1. Ceux qui prennent en entrée des données (des résultats des observations empiriques par exemple) et construisent un réseau de causes à effets;

2. Ceux à qui on fournit les résultats désirés et en retour le réseau fournit les entrées nécessaires pour atteindre ces résultats désirés.

La première catégorie est utilisée pour aider à prendre une décision et la seconde pour l'analyse de compromis. Un réseau bayésien même simple et petit peut simultanément gérer un assez grand nombre de valeurs possibles d'un élément à prédire. Les réseaux bayésiens peuvent être étendus à des réseaux bayésiens dynamiques en ajoutant l'aspect temporel à celui-ci. Un réseau bayésien dynamique (RBD) est un modèle temporel qui représente un système dynamique, c'est-à-dire un système qui change d'état par rapport au temps [32]. Un RBD est donc composé d'une séquence de réseaux bayésiens identiques à un moment t, $t = 1, 2, ..., T$.

Au vu de quelques travaux [33], [34], [35], [36], [37], [38] en génie logiciel qui utilisent les réseaux bayésiens, on constate que ce modèle est un modèle de prédilection, sans doute entre autres à cause du fait de :

- Pouvoir considérer les données incertaines lors de la construction du modèle. Cette caractéristique permet d'obtenir des résultats plus proches de la réalité;

- Pouvoir apprendre à partir d'un petit jeu de données;

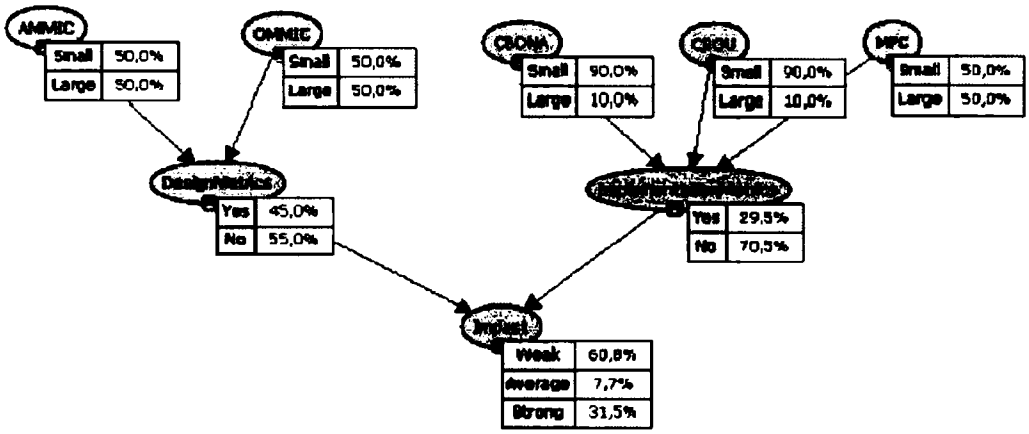
- Pouvoir facilement faire évoluer le modèle, car les données nécessaires pour la structure et les tables de probabilité peuvent être ajoutées de façon incrémentale.

1.2.3 Exemples

Dans le cas de notre problématique, il est question d'utiliser le résultat des métriques des précédents projets ([5], [30], [42], [43], [44], [50]) pour obtenir les connaissances nécessaires qui pourront être utilisées plus tard pour prédire la qualité d'un système en plein développement ou en maintenance. Prédire la qualité d'un système consiste ici à pouvoir attribuer des valeurs aux attributs internes du système comme le couplage, la cohésion, la complexité du code, etc., qui sont pris en considération pour déterminer le niveau de qualité d'un système. Il est donc question de trouver les dépendances entre ce qui est mesurable (les métriques) et ce qui n'est pas mesurable (les attributs internes du système).

Étant donné que les métriques sont des variables quantitatives, il est possible d'avoir un nombre infini de valeurs. Construire un réseau bayésien avec ce type de variables peut donner un réseau énorme et très difficile à interpréter. Pour cette raison, il faut convertir ces valeurs en valeurs discrètes ou ordinales pour réduire le champ de valeurs possibles et le réseau par la même occasion.

Figure 1.7 Exemple de réseau bayésien basé sur des métriques [30].



Dans le réseau bayésien de la figure 1.7, les données d'entrée sont des métriques avec leurs tables de probabilités. Les valeurs des métriques ont été transformées en valeurs discrètes « Small » et « Large ». À chacune de ces valeurs on attribue un pourcentage qui nous dit si le résultat de la métrique fourni par l'utilisateur est bas ou élevé. Les nœuds intermédiaires représentent des métriques de conception et d'implémentation avec leurs tables de probabilités respectives. Ces nœuds intermédiaires permettent de regrouper les nœuds d'entrée par catégorie. Il existe donc une relation de définition entre les nœuds d'entrée et les nœuds intermédiaires. Pour terminer, le nœud de sortie qui est un attribut de qualité non mesurable (impact du changement) et les différentes probabilités associées à ses valeurs possibles (faible, moyen, fort). La relation entre ce nœud et ses nœuds parents est une relation de causalité. Ce réseau bayésien permet donc, à partir des métriques sélectionnées ainsi que leurs valeurs associées, de prédire le degré d'impact qu'une modification faite sur un module du système peut avoir sur le reste du système. En modifiant les valeurs associées aux métriques choisies, on obtient plusieurs réseaux bayésiens et plusieurs observations sont faites. Après

vérification et validation de ces observations, celles-ci sont sauvegardées et publiées. Pour l'instant, seuls les résultats de [31] ont été introduits dans notre ontologie.

1.3 Les règles

Les règles sont un modèle facile à interpréter et à créer, car très proche du langage naturel. Elles sont explicites, mais plus coûteuses en ressource temps, comparées aux arbres de décision. D'autres modèles de qualité comme les arbres de décision, les modèles basés sur les cas peuvent être transformés en règles. Une règle à la base, a la structure « si *condition* alors *conséquence* ». En génie logiciel, lorsque les acteurs du domaine sauvegardent leurs observations dans des règles, un élément appelé *pertinence* est ajouté à ces règles. On a donc une structure «si *condition* alors *conséquence* (*pertinence*) ». L'élément pertinence est très important, car il aide l'utilisateur dans son choix de règles et l'oriente ainsi dans sa prise de décision.

La condition d'une règle peut être une seule comparaison ou plusieurs. Dans le cas de plusieurs comparaisons, il est conseillé de n'avoir que les conjonctions comme connecteur, car en fait une disjonction dans la partie condition veut dire qu'on a regroupé deux règles en une. La condition d'une règle est certes de type Boolean, mais dans notre cadre de travail, cette partie doit être exprimée dans un format bien précis qui est : *<métrique signe-de-comparaison limite>* ou une combinaison de ce format relié implicitement par une conjonction (règle conjonctive). Lorsque nos règles proviennent d'arbres de décision, la pertinence de chaque règle renseigne sur le nombre de cas correctement classés par l'algorithme utilisé (cas réussis) et le nombre de cas mal classés par l'algorithme utilisé, on parle de cas non réussis ou échoués. Les règles provenant d'arbres de décision ont donc la structure « si *condition* alors *conséquence* (*cas réussi/cas non réussi*) ».

Exemple de règle issue des travaux de [30] :

MPC $\leq$ 1 OMMIC $\leq$ 4 AMMIC ϵ]0, 3] ➔ impact : very-weak (52.0/3.0). Cette règle se résume à dire que si dans une application orientée-objets, la métrique Message Passing Coupling (MPC) est inférieure ou égale à 1, la métrique Others Method-Method Import Coupling (OMMIC) est inférieure ou égale à 4 et la métrique Ancestors Method-Method Import Coupling (AMMIC) a une valeur comprise entre 0 exclu et 3, alors l'impact des changements est très faible, avec 52 cas réussis contre 3 non réussis.

Lorsque les règles proviennent des réseaux bayésiens, la pertinence de chaque règle est la probabilité pour que la conséquence de la règle soit vraie en fonction de la condition.

Autre exemple de règle issue des travaux de [30] :

CBONA $\leq$ 3.5 CBOU $\leq$ 0.5 ➔ impact : very-weak (0.46). Cette règle est traduite comme suit : si la métrique Coupling Between Object No Ancestors (CBONA) est inférieure ou égale à 3.5 et la métrique Coupling Between Object Using (CBOU) est inférieure ou égale à 0.5, l'impact des changements est très faible.

Dans ce chapitre nous avons présenté les modèles de qualité les plus utilisés. Nous avons aussi montré comment ces modèles sont utilisés dans le génie logiciel pour détecter la mauvaise qualité d'un logiciel. Dans le chapitre suivant, nous allons parler des aspects du web sémantique intéressants pour notre problématique, à savoir la notion d'ontologie, les langages du web sémantique, les outils de développemnt d'ontologies, etc.

CHAPITRE II

LA NOTION D'ONTOLOGIE

Le terme « ontologie » provient du domaine de la philosophie et est défini comme la théorie de l'être ou des concepts généraux de l'être. Bien que cette notion trouve ses origines dans le domaine de la philosophie, elle a aussi été adoptée dans les domaines de l'intelligence artificielle, du web sémantique, du génie logiciel, des mathématiques, etc. En informatique, ontologie est un terme technique qui dénote un artefact conçu dans le but de pouvoir modéliser des connaissances d'un domaine réel ou imaginaire [8]. L'un des premiers objectifs du développement d'une ontologie est de partager une même compréhension de la structure d'une information entre des personnes [10][11]. On distingue aussi, entre autres, les raisons suivantes pour lesquelles il est nécessaire de développer une ontologie [12][18] :

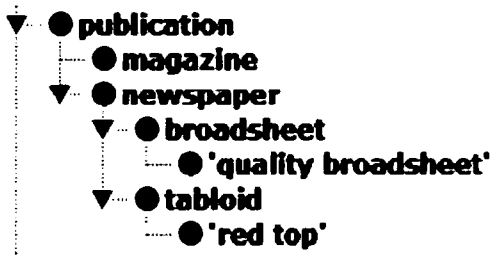
- Permettre la réutilisation des connaissances d'un domaine;
- Rendre explicites les hypothèses d'un domaine;
- Séparer les connaissances du domaine des connaissances opérationnelles;
- Analyser les connaissances d'un domaine;
- Faciliter l'interopérabilité entre deux systèmes;

- Assurer la fiabilité des connaissances;
- Faciliter la communication entre utilisateurs.

2.1 Caractéristiques d'une ontologie

Une ontologie est caractérisée par ses différents ensembles de classes, propriétés et individus. Les classes sont communément appelées *concepts*. On distingue deux types de propriétés : les *propriétés de données* ou attributs des classes et les *propriétés objets* qui définissent les relations qui existent entre deux classes. La classe à gauche d'une relation est le *domaine* et la classe à droite d'une relation est le *co-domaine* (voir exemple ci-dessous). Les individus d'un concept, par analogie au développement orienté objet, sont des instances d'une classe. Des restrictions peuvent s'appliquer sur les propriétés. Les concepts sont organisés dans un graphe et sont reliés, entre–eux, par des relations sémantiques et de subsomption (inclusion). Tous les concepts d'une ontologie sont inclus dans le concept *THING*. C'est aussi la classe de tous les individus. Le concept le plus spécifique est le concept *NOTHING* qui est le sous-concept de tous les concepts d'une ontologie. *NOTHING* est le concept qui n'a aucune instance. Les exemples suivants proviennent de l'ontologie people.owl[1].

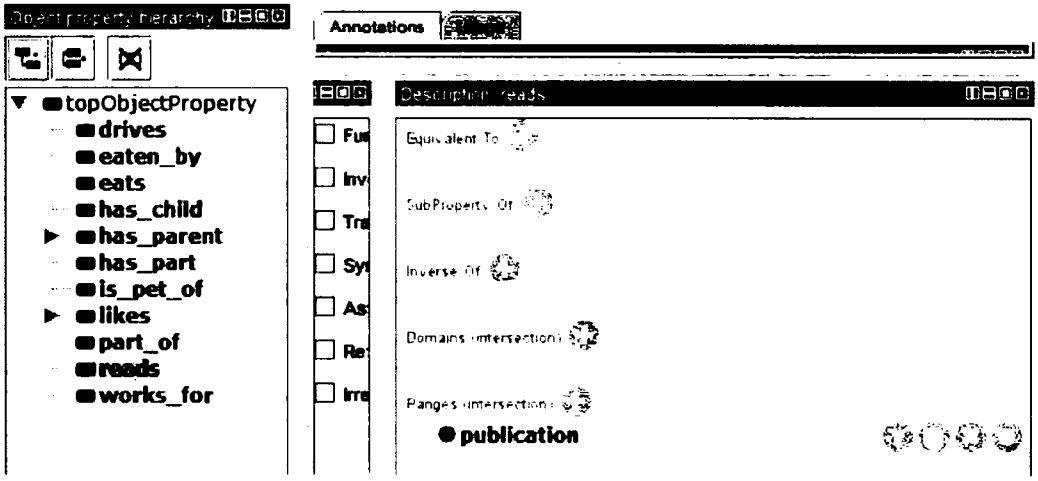
Exemple de taxonomie de classes (concepts) :



[1] http://owl.man.ac.uk/2006/07/sssw/people.owl

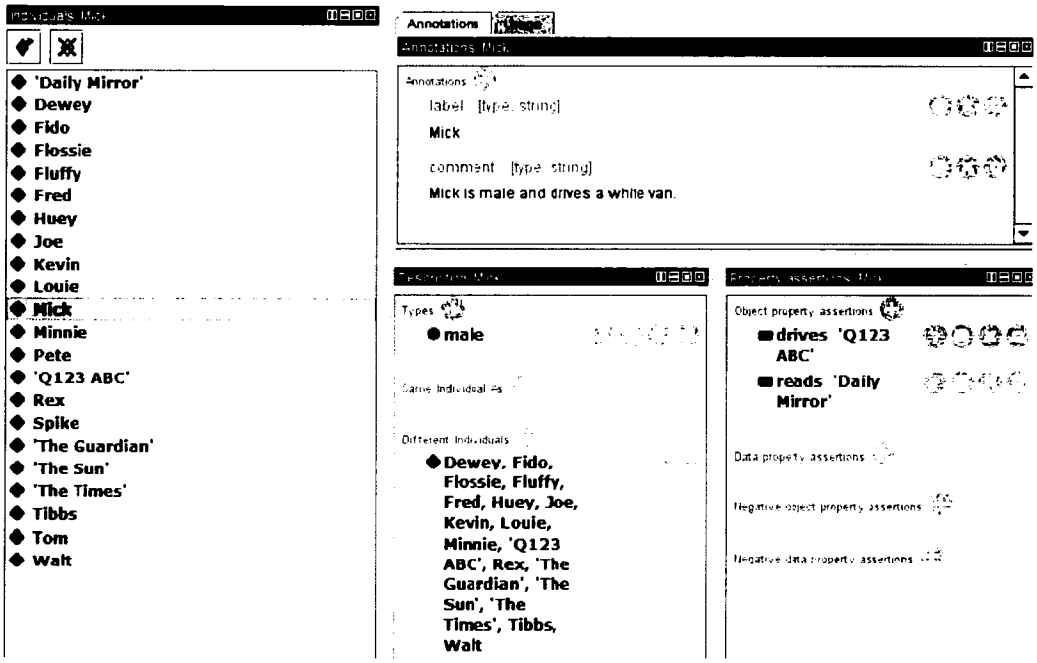
La classe *publication* subsume les classes *magazine* et *newspaper*. La classe *newspaper* subsume la classe *broadsheet* et *tabloid*. La classe *broadsheet* subsume la classe *quality broadsheet* et la classe *tabloid* subsume la classe *red top*.

Figure 2.1 Exemple de propriétés objets.



On observe la hiérarchie des propriétés objets à gauche de la figure 2.1. La relation « reads » qui est sélectionnée a pour co-domaine (« range » en anglais) le concept « publication ». Dans la partie « Domains », aucun concept n'a été ajouté cela veut dire que la propriété reads a pour domaine le concept THING. Donc, tout concept de l'ontologie peut lire une publication. White man van reads only tabloid : Un concept « white man van » qui est un sous-concept de « man » lit uniquement les tabloids.

Figure 2.2 Exemple d'individus.



Dans la partie gauche de la figure 2.2, on a la liste des individus de l'ontologie. L'individu *Mick* sélectionné est une instance de la classe *male* qui conduit (« drives ») une *Q123 ABC* et lit (« reads ») le *Daily Mirror*. Les aspects importants d'une ontologie sont [9] :

- La clarté : toute définition d'un concept doit communiquer le sens voulu du terme, être indépendante du contexte et complète.

- La cohérence : il ne doit pas y avoir de contradiction entre tout ce qui est inféré et les définitions des concepts.

- L'extensibilité : on doit pouvoir spécialiser ou personnaliser une ontologie sans avoir à modifier les fondations de l'ontologie.

- Un engagement ontologique minimal : une ontologie doit se limiter à définir un vocabulaire pour décrire un domaine de manière complète si possible. Une ontologie ne doit pas raisonner pour répondre à une question posée par exemple.

- Une déformation d'encodage minimale : cela a lieu lorsque des choix de représentation sont faits par rapport à une notation précise. La conception doit être spécifiée au niveau des connaissances sans dépendre de l'implémentation.

2.2 Types d'ontologies

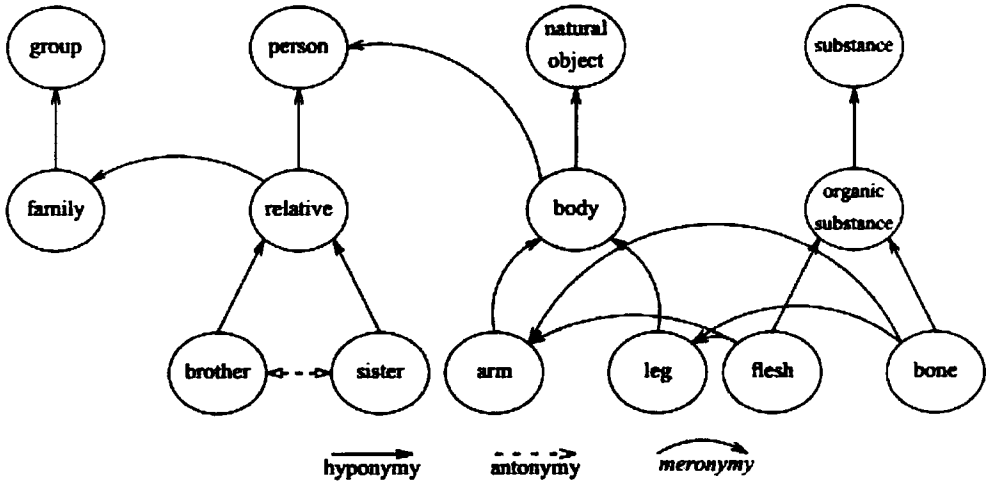
Les ontologies peuvent être classées différemment en fonction des critères de classification. Les critères les plus fréquemment utilisés sont le degré d'abstraction et le domaine d'application. Sur la base de ces critères, on obtient la classification suivante [13] :

1. Les ontologies de haut niveau : les concepts sont très généraux et indépendants d'un problème. On peut aussi les appeler méta-ontologies. On peut citer DOLCE et Wordnet comme exemples d'ontologies de haut niveau. DOLCE[2] (Descriptive Ontology for Linguistic and Cognitive Engineering) est une ontologie qui a pour but de capturer les différentes catégories ontologiques du langage naturel et du bon sens humain. Ces catégories sont considérées comme des artéfacts cognitifs qui dépendent de la perception humaine, des empreintes culturelles et des conventions sociales.

[2] http ://www.loa-cnr.it/DOLCE.html

WordNet[3] est une large base de données lexicale pour la langue anglaise. Les mots anglais sont groupés en ensembles de synonymes appelés synsets. Ces synsets sont reliés entre eux par rapport aux relations sémantiques existantes. WordNet est indiqué pour la linguistique computationnelle et le traitement du langage naturel.

Figure 2.3 Représentation de trois relations sémantiques entre une variété de concepts lexicaux [6].



2. Les ontologies de domaine : les concepts sont spécifiques à un domaine. Comme exemples, on peut citer :

   UMLS[4] (Unified Medical Language System) intègre et distribue la terminologie, la classification et les standards, ainsi que les ressources associées pour permettre la création et l'interopérabilité des systèmes d'information et services biomédicaux. L'une des principales utilisations

[3] Princeton University "About WordNet." WordNet. Princeton University. 2010. http ://wordnet.princeton.edu/

[4] http ://www.nlm.nih.gov/research/umls/

d'UMLS est la connexion des termes médicaux, des informations médicales, des noms de médicaments, des codes de facturation entre différents systèmes.

Dublin Core[5] est une ontologie qui intègre les termes qui spécifient différents types de ressources (papier, digitales, culturelles, scientifiques, etc.). Ces termes sont des données sur les données (méta-données).

3. Les ontologies de de tâches : les concepts sont spécifiques aux tâches d'une application ou d'un processus. Comme exemple, on a l'ontologie MyGrid[6] qui est composée d'une ontologie de domaine et d'une ontologie de services (tâches) qui décrivent le domaine de la recherche bio-informatique et en détaillent l'ensemble des services (tâches) liés à ce domaine.

4. Les ontologies d'application : les concepts sont propres à une tâche particulière d'une application donnée. Comme exemple, on a l'ontologie d'application B31.1 qui est l'une des ontologies du projet MEMORAe[7]. Cette ontologie porte sur « le module B31.1 de mathématiques appliquées suivi par les étudiants de l'IUP (Institut Universitaire Professionnalise) MIAGE (Méthodes Informatiques Appliquées à la Gestion des Entreprises) de l'Université de Picardie Jules Verne d'Amiens » [14]. Cette ontologie décrit les termes et notions des mathématiques appliquées.

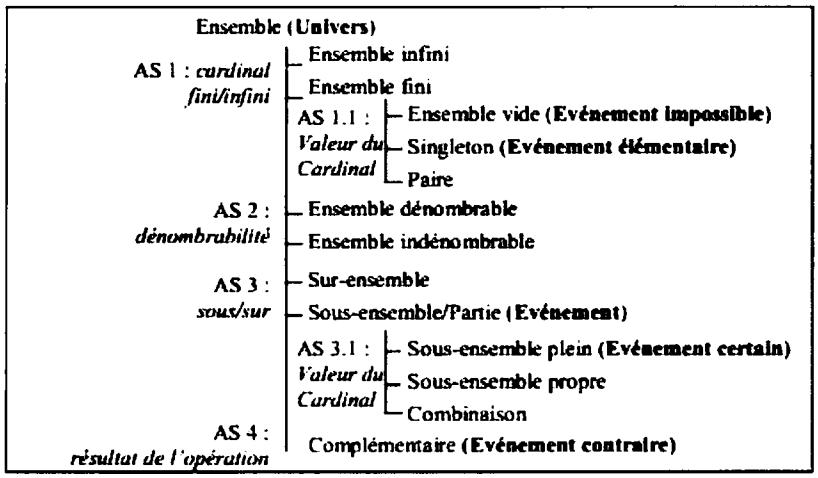
---

[5] http ://dublincore.org

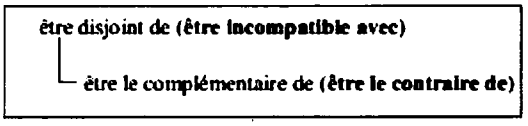
[6] http ://www.mygrid.org.uk/tools/service-management/mygrid-ontology/

[7] http ://www.hds.utc.fr/memorae/

Exemple de hiérarchie de concepts (1) et de relations (2) de l'ontologie B31.1. [14] :



1

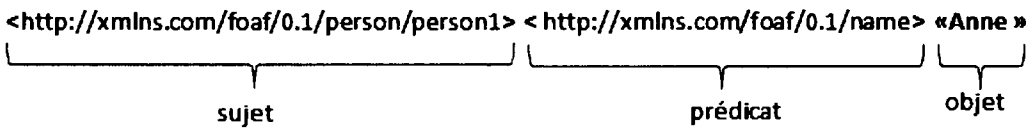


2

2.3 Les langages et outils pour les ontologies

La plupart des langages utilisés pour créer une ontologie sont proches de la logique du premier ordre, dans ce sens que les connaissances sont représentées sous la forme sujet – prédicat – objet.

Exemple :



Le groupe W3C a publié le langage OWL (Web Ontology Language) qui est devenu le standard pour modéliser les ontologies publiables et échangeables sur internet. Le langage OWL s'inspire de ses prédécesseurs DAML + OIL et de la logique de description. OWL est fondé sur RDF (Resource Description Framework) et est plus expressif que RDFS (Resource Description Framework Schema). En effet, en se basant sur le facteur expressivité, on distingue OWL-Lite, OWL-DL et OWL-Full. OWL-Full étant le langage le plus expressif. Plus le pouvoir expressif d'un langage est grand, plus coûteux devient le raisonnement.

OWL-Full permet des combinaisons arbitraires entre les primitives d'OWL et celles de RDF et RDFS. OWL-Full a une compatibilité entière et ascendante avec RDF, syntaxiquement et sémantiquement. Cela veut dire que tout ce qui est valable syntaxiquement et sémantiquement en RDF l'est aussi en OWL-Full. OWL-Full est si puissant qu'il est indécidable, car il n'y a aucun support pour un raisonnement complet.

OWL-DL (Description Logic) est un sous-langage d'OWL-Full avec une compatibilité partielle avec RDF : tout document valide en OWL-DL l'est aussi en RDF, mais tout document RDF n'est pas forcement valide en OWL-DL. OWL-DL est décidable, car il admet un support efficace au raisonnement. OWL-Lite est un sous-langage de OWL-DL qui se limite à un sous-ensemble des constructeurs d'OWL-DL. OWL-Lite est plus facile à appréhender pour les utilisateurs et plus facile à implémenter pour les constructeurs d'outils. Par contre OWL-Lite a une expressivité très réduite.[16]

Il existe plusieurs syntaxes pour représenter les ontologies. On peut citer les syntaxes RDF/XML, Turtle, JSON et N3 comme exemples. Les trois dernières syntaxes étant plus faciles à lire et à écrire pour l'utilisateur. Il existe de nombreux outils pour les

ontologies qui supportent le langage OWL et des syntaxes de représentation d'une ontologie. On peut diviser les éditeurs d'ontologies en deux groupes. Les éditeurs gratuits et accessibles sur internet et les éditeurs payants. Qu'un éditeur soit gratuit ou pas, la majorité des caractéristiques recherchées chez un éditeur d'ontologies sont entre autres [15] : robustesse et prêt à l'utilisation, support des formats RDF, RDFS et OWL, un environnement collaboratif, facilité d'utilisation, taxonomie graphique des classes et propriétés, support de la croissance de grandes ontologies, vérification de la consistance, un environnement pour plusieurs ontologies, possibilité de formuler des requêtes, possibilité de formuler des axiomes, facilité à naviguer entre les concepts, vue graphique, possibilité d'interopérabilité et raisonneur.

Il existe de nombreux éditeurs gratuits tels que : Protégé[8], SWOOP[9], Ontolingua[10], KMgen[11], IsaViz[12], DOE[13] (Differential Ontology Editor), etc. Dans cette catégorie d'éditeurs, le plus connu et le plus utilisé est *Protégé*. C'est un outil open source développé par l'université de Stanford. Depuis sa première version, il a beaucoup évolué et continue d'évoluer rapidement. Au cours de la réalisation de notre ontologie, dans un intervalle de temps de quelques mois (environ 6 à 7 mois), deux versions ont été créées. Ajouté à cela le fait que chaque version a plusieurs sous-versions. La communauté de Protégé est très grande et très active et il est facile de l'utiliser et de trouver de l'aide sur internet. Il intègre le langage OWL-DL. En plus des fonctions nécessaires pour créer une ontologie, il a des composants tels que :

---

[8] http ://protege.stanford.edu

[9] http ://code.google.com/p/swoop/

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

[11] http ://www.algo.be/dev-logiciels.htm#kmed

[12] http ://www.w3.org/2001/11/IsaViz/

[13] http ://www.eurecom.fr/~troncy/DOE/

raisonneurs (pellet, HermiT, FACT++, etc.), interfaces graphiques, comparateurs d'ontologies, générateur de code java, documentation HTML de l'ontologie, etc.

Parmi les éditeurs commerciaux, on peut citer : SemanticWorks[14], TopBraid Composer[15], KAD-Office[16], etc. L'éditeur KAD-Office est spécialisé sur la représentation graphique des connaissances industrielles d'ingénierie à l'aide de système d'exploitation. On distingue aussi parmi ces outils ceux qui sont semi-automatiques, car ils aident à construire une ontologie en parcourant un corpus textuel, à la recherche des termes fournis par l'utilisateur et les relations qui existent entre ces termes. Les raisonneurs des éditeurs d'ontologies permettent de vérifier la consistance du contenu de l'ontologie. Ils ne raisonnent pas pour répondre à une question ou un problème posé. Cependant, les ontologies peuvent être utilisées dans le processus d'aide à la décision, car il fournit des informations importantes à prendre en considération lors de la prise de décision, ces informations étant :

- Le vocabulaire à utiliser et à respecter aussi bien dans la requête que dans la réponse/solution.
- Les individus ou instances autorisé(e)s dans la décision.
- Les restrictions importantes, nécessaires, voire indispensables à ne pas ignorer.

En résumé, on peut conclure que les ontologies servent de délimiteur et de régulateur. Elles empêchent de se disperser et de se perdre dans un domaine, une application ou

[14] http ://www.altova.com/products_semanticworks.html

[15] http ://www.topquadrant.com/products/TB_Composer.html

[16] http ://www.iknova.com/produits.htm

une tâche. Elles permettent de s'assurer que tous les termes qu'elles contiennent ont le même sens pour tous et que chaque terme est utilisé comme il se doit.

2.4 Processus de développement d'une ontologie

Les ontologies peuvent être développées de plusieurs façons. Étant donné qu'il n'existe pas de processus standard de développement, nous avons exploré les différentes méthodes proposées dans la littérature afin de déterminer celle appropriée à notre cas. Nous allons présenter les méthodes explorées dans la suite de cette section et faire un choix pour le développement de notre ontologie en nous basant sur certains critères de comparaison. Avant de commencer, il est important de mentionner que le développement d'une ontologie est un processus itératif.

2.4.1 La méthodologie de Noy et McGuinness

Dans [12], Noy et McGuinness nous présentent une méthodologie de l'ingénierie des connaissances assez simple pour développer des ontologies. Cette méthodologie se résume en sept étapes :

1. Déterminer le domaine et la portée de l'ontologie. Ils suggèrent de commencer par déterminer le domaine auquel se rapportera l'ontologie en répondant à des questions simples, telles que :
    - Quel domaine sera couvert par l'ontologie ?
    - Dans quel but sera utilisée l'ontologie ?
    - Pour quels types de questions les informations contenues dans l'ontologie doivent-elles fournir des réponses ?

- Qui utilisera et maintiendra l'ontologie ?

Une façon pratique de déterminer la portée de l'ontologie est de dresser une liste de questions (competency questions) pour lesquelles un système à base de connaissances est capable de répondre en se basant sur cette ontologie.

2. Considérer la réutilisation d'ontologies existantes

Avant d'aller plus loin avec le développement d'une ontologie, il est important de savoir s'il n'existe pas des ontologies qui peuvent faciliter notre tâche de développement. En prenant en compte les ontologies qui existent déjà, on peut trouver une ontologie (ou plusieurs) qui nécessite d'être raffiné ou agrandi pour répondre à nos attentes. Il y a des cas où les ontologies à réutiliser sont claires dès le départ. Il s'agit des cas où l'on doit développer une ontologie pour un système qui doit interagir avec d'autres systèmes qui utilisent leurs propres ontologies ou des vocabulaires bien spécifiques.

3. Énumérer les termes importants de l'ontologie

Faire une liste des termes qu'on aimerait expliquer, les termes dont on veut parler, leurs caractéristiques. Il faut lister tous les termes qui sont importants pour la compréhension de notre domaine d'application, sans chercher à savoir à quelles catégories (classes, attributs) appartiennent ces termes.

4. Définir les classes et la hiérarchie des classes

Parmi les termes de la liste obtenue à l'étape précédente, il faut définir et hiérarchiser les termes qui auront été identifiés comme étant des classes. Ce sont ces termes-là qui décrivent les objets ayant une existence indépendante et non les termes qui décrivent ces objets. Pour réaliser cela, il faut choisir

entre les trois approches possibles, celle qui s'adapte le plus à son cas. On distingue donc les approches **top-down**, **bottom-up** et la **combinaison** de ces deux approches.

- L'approche top-down consiste à commencer par définir les termes les plus généraux du domaine ou de l'application et par la suite définir les termes plus spécifiques propres à chaque terme général. Répéter ce processus pour chacun des sous-termes des termes généraux.

- L'approche bottom-up consiste à commencer par la définition des termes les plus spécifiques de la hiérarchie et les grouper ensuite sous des termes un peu plus généraux. Il faut procéder ainsi jusqu'à ce qu'il ne soit plus possible de les grouper. À ce moment-là, les termes les plus généraux ont été atteints.

- La combinaison des approches top-down et bottom-up consiste tout d'abord à définir les termes qui attirent le plus l'attention (les termes saillants). Ensuite il faut correctement les généraliser (approche bottom-up) et les rendre plus spécifiques (top-down).

5. Définir les propriétés des classes (slots)

Après avoir éliminé les classes de la liste obtenue à l'étape 3, il est plus probable que tous les termes restants soient les propriétés des classes. Pour chacune de ces propriétés, il faut donc déterminer quelle classe est ce qu'elle décrit. En général, ces propriétés peuvent être différenciées comme suit :

- Les propriétés intrinsèques

- Les propriétés extrinsèques

- Des parties de l'objet qu'elle décrit, dans le cas où l'objet est structuré.

- Des relations avec d'autres individus

Chaque propriété doit être rattachée à la classe la plus générale qui peut avoir cette propriété. Ainsi, toutes les sous-classes hériteront automatiquement des propriétés de leurs classes parents.

6. Définir les facettes (cardinalités, types des valeurs, etc.) des slots

   Les slots peuvent avoir différentes facettes qui peuvent être le type accepté par le slot, les valeurs acceptées par le slot, la cardinalité, etc. Il faut aussi déterminer les domaines et co-domaines des slots.

7. Créer les instances

   Il faut créer les instances de notre ontologie en choisissant les classes qui doivent être peuplées d'individus. Pour chacune de ces classes, il faut créer ses individus et compléter les slots des valeurs appropriées.

2.4.2 La méthodologie METHONTOLOGY

De la méthodologie METHONTOLOGY est né le système METHONTOLOGY qui permet de construire des ontologies du point de vue des connaissances et inclus l'identification du processus de développement d'ontologies, un cycle de vie basé sur des prototypes évolutifs et des techniques particulières pour réaliser chaque activité. Les principales activités du processus de développement de cette méthodologie sont : les activités de gestion de projet, les activités de développement et les activités de support [20].

Les activités de gestion de projet comprennent :

- La planification, qui permet d'identifier et d'ordonner les tâches, déterminer le temps et les ressources nécessaires (personnel, budget, matériel) à leur réalisation.

- Le contrôle, qui permet de garantir que les tâches identifiées sont réalisées dans le respect de ce qui a été planifié.

- L'assurance qualité, qui assure que tous les produits intermédiaires (ontologie, documentation, logiciel) sont satisfaisants.

Les activités de développement sont [19] :

- La spécification. Au moins les informations suivantes doivent faire partir du document de spécification :

    - L'objectif de l'ontologie ainsi que les scénarios d'utilisation, les utilisateurs cibles, etc.

    - Le degré de formalisme de l'ontologie. Ce degré est obtenu en fonction du formalisme qui sera utilisé pour coder les termes et leurs significations. Ce formalisme peut être le langage naturel, un langage formel strict ou entre les deux.

    - La portée qui inclut l'ensemble des termes à représenter, leurs caractéristiques et granularités.

- La conceptualisation qui structure les connaissances du domaine en modèles significatifs du point de vue des connaissances.

- La formalisation qui consiste à transformer le modèle conceptuel en modèle formel ou computationnel.

- L'implémentation qui transforme les modèles computationnels en langage computationnel.

- La maintenance qui permet de corriger et mettre à jour l'ontologie.

Les activités de support se font parallèlement avec les activités de développement. Ces activités de support sont :

- L'acquisition des connaissances

- L'évaluation, après chaque phase pour rendre un jugement technique sur les ontologies, leurs environnements et documentations, par rapport à un cadre de référence.

- La documentation claire et exhaustive de chaque étape et de ses produits résultants

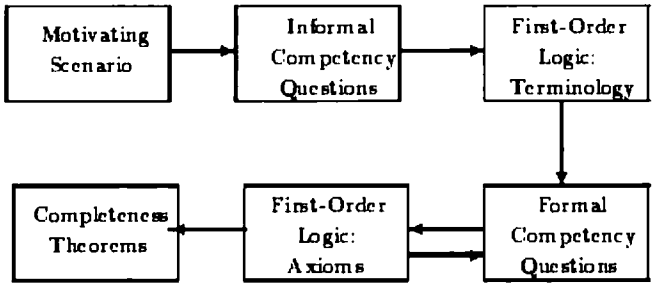
- La gestion de configuration pour enregistrer toutes les versions de la documentation, du code de l'ontologie et du logiciel afin de pouvoir contrôler les changements.

Gómez-Pérez et al. [25][26] décrivent en détail comment réaliser certaines de ces activités comme la planification, le contrôle, l'assurance qualité, l'implémentation, la conceptualisation et l'acquisition des connaissances.

2.4.3 La méthodologie de Grueninger et Fox

Cette méthodologie découle de l'expérience obtenue au cours du projet ontologique TOVE (Toronto Virtual Enterprise). TOVE est un projet du domaine des processus d'affaires et des activités de modélisation. La méthodologie proposée par Grueninger et Fox consiste essentiellement à construire un modèle logique des connaissances qui doivent faire partie de l'ontologie. Pour arriver à ce modèle, une description informelle des spécifications de l'ontologie est faite, et cette description est formalisée par la suite [20]. La figure 2.4 résume les étapes à suivre pour développer et évaluer une ontologie selon Grueninger et Fox.

Figure 2.4 Étapes de conception et d'évaluation d'une ontologie [17].



Voici les détails de cess étapes nécessaires pour réaliser ce modèle logique [17] :

1. Scénario(s) de motivation

   Les scénarios de motivation sont des descriptions des problèmes ou des exemples qui n'ont pas été pris en considération par les ontologies existantes. Ces scénarios proposent aussi des solutions possibles aux problèmes décrits. Ces solutions donnent ainsi une idée informelle du sens des objets et des relations qui seront plus tard introduits dans l'ontologie. Les scénarios de motivation sont importants surtout dans le cas où l'on a différents objets dans les différents scénarios. Que l'on veuille créer une nouvelle ontologie ou faire

évoluer une ontologie existante, il faut avoir un ou des scénarios de motivation comprenant la description du problème et la ou les solutions à ce problème.

2. Questions informelles de compétence

Dans cette phase, il faut formuler en langue naturelle des questions basées sur les scénarios de motivation. Ces questions sont des exigences que l'ontologie doit satisfaire ; elles permettent d'évaluer l'ontologie. L'ontologie doit être capable de reproduire ces questions et d'y répondre. Ces questions permettent d'identifier la portée de l'ontologie, car elles permettent de connaitre le type de réponses contenues ou non dans l'ontologie.

3. Formalisme de l'ontologie

Après avoir formulé de façon informelle les questions de compétences, on identifie les termes importants, leurs attributs et les relations entre-eux. Ces termes permettent de définir la terminologie de l'ontologie dans un langage formel comme la logique de premier ordre.

4. Questions formelles de compétence

Dans cette étape, il s'agit de formuler de façon formelle les questions de compétence en utilisant la terminologie définie à l'étape précédente.

5. Spécification des axiomes

Dans cette étape, on spécifie les axiomes de l'ontologie en définissant les termes identifiés à l'étape 3 (formalisme de l'ontologie) et les restrictions qui s'appliquent à leur interprétation. Si ces axiomes ne suffisent pas pour

représenter de façon formelle les questions de compétences et répondre à ces questions, alors il faut ajouter des objets ou des axiomes jusqu'à ce qu'il soit possible de formaliser les questions et leurs réponses. Le développement des axiomes en accord avec les questions de compétence est donc un processus itératif.

6. Complétude

Une fois les axiomes spécifiés, il faut définir les conditions qui doivent être remplies afin que les réponses aux questions de compétence soient complètes.

2.4.4 La méthodologie d'Uschold et al.

Cette méthodologie se base sur l'expérience acquise lors du développement de l'ontologie Enterprise Ontology, qui est une ontologie pour les processus de modélisation d'entreprise. Cette ontologie a pour but de faciliter l'interopérabilité entre les différents outils utilisés dans l'entreprise pour réaliser des modèles. Cette méthodologie recommande de respecter les étapes suivantes pour développer une ontologie [18] :

1. Identification des objectifs et de la portée de l'ontologie

Cette étape consiste à spécifier très clairement les raisons pour lesquelles l'ontologie est construite. Elle renseigne aussi sur les cas d'utilisation de l'ontologie. En effet, identifier les cas d'utilisation possibles permet de déterminer les objectifs de l'ontologie.

2. Construction de l'ontologie

La construction proprement dite de l'ontologie se déroule en trois étapes qui sont :

a) Capture de l'ontologie

La capture de l'ontologie consiste à identifier les concepts clés et les relations existantes du domaine d'intérêt, donner des définitions textuelles de ces concepts et de ces relations de la façon la moins ambiguë possible et identifier les termes qui correspondent à ces concepts et relations. Cette approche utilisée pour déterminer tous les concepts de l'ontologie est appelée approche middle-out. On identifie d'abord les concepts importants. Ensuite, on fait une généralisation ou une spécialisation à partir de ces concepts importants identifiés. Les travaux de Douglas Skuce [21] proposent une représentation plus formelle que la langue naturelle et moins formelle qu'un langage formel pour les définitions des concepts et relations.

b) Codage de l'ontologie

Le codage consiste à représenter explicitement dans un langage formel les termes résultants de l'étape précédente. Pour cela, les termes sont groupés en catégories (classes, entités, relations, propriétés, etc.) et sont codés dans le langage formel choisi.

c) Intégration des ontologies existantes

Pendant les phases de capture et de codage de l'ontologie, il faut se poser la question de savoir si une ou des ontologies déjà existantes peuvent être utilisées en partie ou complètement. Si oui, voir comment on peut les utiliser dans l'ontologie que l'on construit.

3. Évaluation

Durant cette phase, on met en application l'évaluation dans le contexte de la technologie du partage des connaissances (Knowledge sharing technology) défini dans [24], comme suit :

to make a technical judgement of the ontologies, their associated software environment, and documentation with respect to a frame of reference ... The frame of reference may be requirements specifications, competency questions, and/or the real world.

Ma traduction : « *pour rendre un jugement technique sur les ontologies, leurs environnements associés et leur documentation en respectant un cadre de référence... Le cadre de référence peut être les spécifications des exigences, les questions de compétence ou le monde réel.* »

4. Documentation

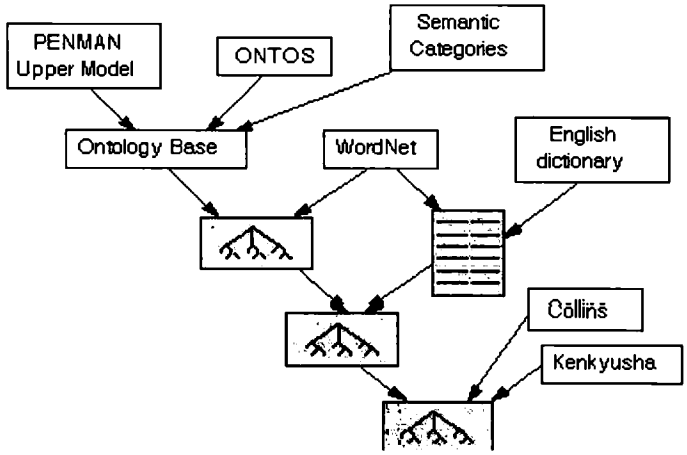
La documentation d'une ontologie doit contenir toutes les hypothèses importantes à propos des concepts majeurs définis dans l'ontologie, ainsi que les primitives utilisées dans les définitions contenues dans celle- ci. La documentation peut varier dépendamment du type et de l'objectif de l'ontologie.

2.4.5 La méthodologie basée sur l'approche SENSUS

SENSUS est une ontologie de plus de 50000 concepts organisés hiérarchiquement en fonction de leur niveau d'abstraction. Cette ontologie est destinée à être utilisée dans le traitement de la langue naturelle afin de fournir une structure conceptuelle pour le développement des outils de traduction [20][23]. La base de cette ontologie a été obtenue en extrayant et en fusionnant manuellement les informations provenant de diverses sources de connaissances électroniques (PENMAN Upper Model, ONTOS, catégories sémantiques d'un dictionnaire). WordNet a été fusionné avec cette base et un dictionnaire anglais. Le résultat de ces deux fusions a été ensuite fusionné avec le

dictionnaire Espagnol/Anglais Collins, puis le dictionnaire Japonais/Anglais Kenkyusha [22]. La figure 2.5 illustre ces fusions.

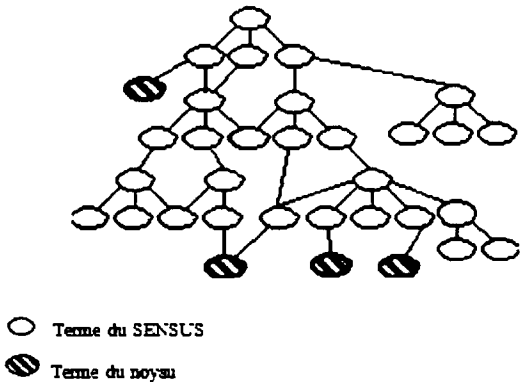
Figure 2.5 Stratégie de fusion de SENSUS [23].



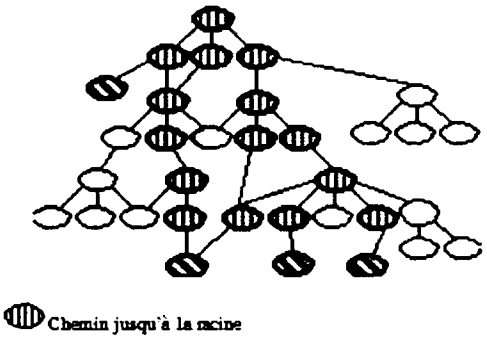
Les étapes à suivre lorsqu'on veut développer une ontologie en se basant sur l'approche SENSUS sont les suivantes [20][23] :

1. Construction du noyau à partir d'une série de termes.

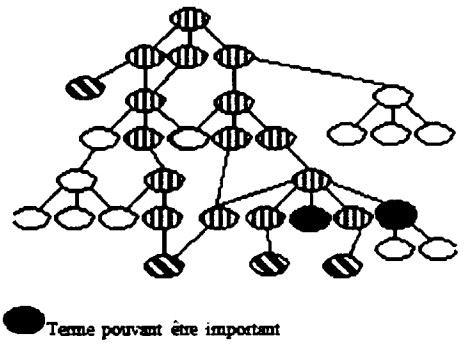
2. Liaison manuelle entre les termes du noyau et SENSUS. Les termes du noyau sont les termes clés d'un domaine choisi.



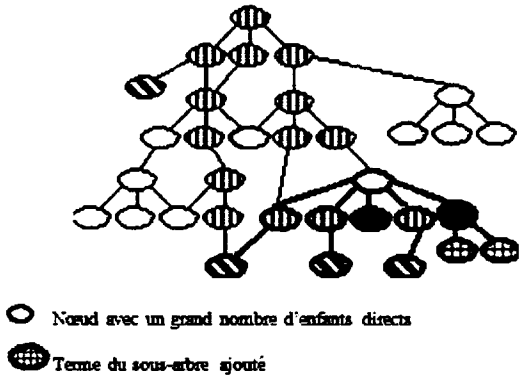
3. Inclusion de tous les concepts qui se trouvent entre les termes du noyau et la racine de SENSUS



4. Ajout des termes pouvant être importants dans le domaine qui ne sont pas encore inclus.



5. Ajout des sous-arbres se trouvant sous les nœuds par lesquels passe un grand nombre de chemins. Les termes de ces sous-arbres sont ajoutés parce qu'on se base sur l'idée que si plusieurs nœuds d'un sous arbre sont importants, il est donc très probable que les autres nœuds de ce sous-arbre soient aussi importants.



2.4.6 La méthodologie de Berneras et al

L'approche de Berneras et al. s'appuie sur le projet KACTUS dont l'un des objectifs est de déterminer 1) dans quelle mesure il est possible de réutiliser des connaissances dans les systèmes techniques complexes et, 2) le rôle des ontologies pour supporter cette réutilisation. Cette approche de développement d'une ontologie est étroitement liée à l'application qui utilisera cette ontologie [20].

Les étapes de développement d'une ontologie selon l'approche de Berneras et al. sont [20] :

1. Spécification :

Il faut lister toutes les spécifications de l'application. Ces spécifications renseignent sur le contexte d'utilisation de l'application et donnent un aperçu des composantes que l'application veut modeler.

2. Conception préliminaire basée sur les catégories importantes d'ontologies de haut niveau

   La liste des termes et des tâches identifiés dans la phase de spécification est utilisée comme donnée d'entrée pour obtenir plusieurs vues du modèle global en accord avec les catégories d'ontologies de haut niveau choisies. Cette phase de conception comprend la recherche des ontologies développées pour d'autres applications, afin de les modifier de façon à pouvoir les utiliser dans la nouvelle application.

3. Structuration et raffinement de l'ontologie

   L'ontologie est structurée et raffinée au besoin pour obtenir un design définitif. On peut utiliser les principes de couplage minimal pour s'assurer que la dépendance entre les modules est faible et que les modules sont le plus cohérents possible.

2.5 Comparaison des différents processus de développement d'ontologies

Étant donné que les ontologies se focalisent uniquement sur les connaissances, il est important de voir si et jusqu'à quel niveau un processus de construction d'ontologies hérite de l'ingénierie des connaissances. Quand on parle de processus, on s'attend à avoir une liste des différentes étapes à suivre pour obtenir un produit final. Ces étapes doivent avoir un ordre et contenir chacune des détails comme : qu'est-ce qui est fait à cette étape, comment (techniques, approches, etc.), quelles sont les ressources nécessaires, quels sont les conseils ou recommandations pour mener à bien cette

étape. Pour cette raison, on prendra en considération le degré de détails dans la comparaison des différents processus de développement d'ontologies.

L'une des motivations de la construction des ontologies est d'éliminer toute ambigüité afin de faciliter le partage et l'utilisation des connaissances. La meilleure façon d'éliminer ces ambigüités est de formaliser autant que possible les connaissances. Cela est possible grâce aux formalismes tels que la logique (logique de premier ordre, logique de description, etc.), les réseaux sémantiques, les « frames », etc. On verra donc si ces processus font des recommandations au niveau du choix des formalismes, si oui, que recommande chaque processus. Un point qui oriente rapidement et aide dans le choix d'un processus, c'est la stratégie que celui-ci adopte. Une stratégie ne peut être appliquée qu'à un type de situation donnée. Cela veut dire qu'en face d'une liste de stratégies et des processus correspondants, on peut déterminer celle qui s'applique à notre situation et ainsi réduire l'éventail des choix de processus qui s'offrent à nous. Dans le cas de la construction d'ontologies, on distingue trois stratégies déterminées en fonction du niveau de dépendance de l'ontologie par rapport à l'application ou au système qui l'utilisera : stratégie dépendante de l'application, stratégie semi-dépendante de l'application et stratégie indépendante de l'application. On comparera donc aussi nos processus en fonction de ce critère. Par analogie avec les stratégies pour la construction d'ontologies, on distingue aussi des stratégies ou approches pour identifier les concepts (classes) d'une ontologie. Ces approches sont l'approche top-down, bottom-up et middle-out. Nous en tiendrons compte dans notre comparaison. Pour finir, nous choisissons comme dernier critère de comparaison, la maturité des processus. Pour connaitre cette maturité, on se base sur le nombre et la complexité des ontologies qui ont été construites à partir de chacun de ces processus et on cherche aussi à connaitre si des systèmes ont été construits sur la base de ces ontologies.

En résumé, nos critères de comparaison sont [20] : l'héritage de l'ingénierie des connaissances, le niveau de détails, les recommandations pour le formalisme, la stratégie utilisée pour construire les ontologies, la stratégie d'identification des concepts de l'ontologie, les techniques recommandées (voir tableau 2.1 et 2.2) et la maturité du processus.

Tableau 2.1 Comparaison des processus de développement d'ontologies.

	Héritage ingénierie des connaissances	Niveau de détails	Formalisme
Uschold et al.	Héritage partiel, car elle identifie clairement les phases d'acquisition, de codage et d'évaluation, mais elle ne propose aucune étude de faisabilité ni de prototypage.	Peu de détails, car elle ne décrit pas avec précision les techniques et les	Aucune recommandation particulière.
Grueninger et Fox	La seule similitude est le fait de déterminer des questions considérées ici comme des questions de compétence	Aucune description détaillée des activités.	La logique.
METHONTHOLOGY	Elle hérite entièrement de l'ingénierie des connaissances parce qu'elle résulte d'une méthodologie pour le développement des systèmes à base de connaissances	Une bonne partie est très détaillée.	Aucune recommandation particulière, mais si on utilise l'outil ODE (Ontology Design Environment) on n'a pas besoin de se poser de questions sur le choix du formalisme, car cet outil génère automatiquement le code formel correspondant aux connaissances définies dans l'ontologie.
Approche Sensus	Aucune similitude avec l'ingénierie des connaissances, car la construction des ontologies se fait par ajout de nouveaux termes à l'ontologie SENSUS déjà existante.	Elle est moyennement détaillée.	Les réseaux sémantiques.
Berneras et al.	Elle s'inspire de l'ingénierie des connaissances car la construction des ontologies se fait en même temps que le développement des systèmes à base de connaissances qui utiliseront ces ontologies.	Elle est très peu détaillée.	Aucune recommandation.
Noy et McGuinness	Une similitude lors de la détermination des questions de compétence.	Assez détaillée.	Aucune recommandation.

Tableau 2.2 Comparaison des processus de développement d'ontologies (suite et fin).

	Stratégie de construction d'ontologies	Stratégie d'identification des concepts de	Techniques
Uschold et al.	Stratégie indépendante de l'application.	Stratégie middle-out	Recommande de déterminer les concepts et les relations pendant l'acquisition, sans dire comment le faire.
Grueninger et Fox	Stratégie semi-dépendante de l'application, car on utilise les scénarios identifiés dans la phase de spécification.	Stratégie middle-out	Formulation des questions de compétence sans description de cette technique.
METHONTHOLOGY	Stratégie indépendante de l'application.	Stratégie middle-out	Absence de technique recommandée pour l'activité de contrôle.
Approche Sensus	Stratégie semi-dépendante de l'application, car les termes du noyau sont obtenus avec une application en tête.	Stratégie bottom-up	Aucune technique particulière.
Berneras et al.	Stratégie dépendante de l'application. La construction des ontologies est étroitement liée aux applications qui les utiliseront.	Stratégie top-down	Aucune technique particulière
Noy et McGuinness	Stratégie indépendante de l'application.	Stratégie middle-out	Formulation des questions de compétence.

En considérant la maturité de ces différents processus de développement d'ontologies, on a les faits suivants :

Méthodologie d'Uschold et al. : des projets complexes ont été développés uniquement dans le domaine d'affaires. L'un de ces projets est le projet « Enterprise Ontology ». Cette ontologie est utilisée dans l'application « Enterprise toolset ».

Méthodologie de Grueninger et Fox : des projets complexes ont été développés en utilisant cette méthodologie. Ces projets se rapportent à un seul domaine. Le projet TOVE comprend : Enterprise Design Ontology, Project Ontology, Scheduling Ontology et Service Ontology.

METHONTOLOGY : plusieurs ontologies et applications dans différents domaines ont été développées en suivant cette méthodologie. Parmi ces ontologies, on peut citer quelques-unes importantes comme CHEMICALS, les ontologies sur les polluants environnementaux, Reference-Ontology et Knowledge Acquisition ontology. Comme applications, on peut citer : OntoAgent qui utilise Reference-Ontology, chemical OntoAgent qui utilise CHEMICALS, Ontogeneration qui utilise CHEMICALS, en plus d'autres ontologies.

Méthodologie basée sur l'approche SENSUS : développement dans plusieurs domaines des ontologies et les applications les utilisant. Certaines de ces ontologies et leurs applications portent sur les armes, les participants, les dirigeants, les scénarios et les campagnes aériennes militaires, etc.

Méthodologie de Berneras et al. : trois ontologies ont été construites pour trois applications dans le domaine des réseaux électriques. Une application pour diagnostiquer les erreurs dans un réseau électrique, une application pour planifier la

reprise de service après une erreur dans le réseau électrique et une application qui contrôle le réseau à partir des applications précédentes.

Méthodologie de Noy et McGuinness : plusieurs projets, mais aucune information sur ces projets.

À la suite de cette comparaison, on ne peut certes pas recommander l'un de ces processus comme processus standard, mais on peut remarquer que la majorité de ces processus gagneraient à être aussi détaillés que le processus METHONTOLOGY et à être utilisés dans plusieurs domaines et non un seul. Étant donné que dans le cadre de notre projet notre ontologie doit être indépendante de toute application, cela réduit notre choix de processus à : Uschold et al, METHONTOLOGY et Noy et McGuinness. La méthodologie de Uschold et al. hérite partiellement de l'ingénierie des connaissances, mais son principal handicap est le manque considérable de détails des activités. Ceci rend donc son utilisation difficile et par conséquent demande beaucoup de temps pour combler ses lacunes. La méthodologie Noy et McGuinness est simple et bien détaillée, la seule similitude qu'elle a avec l'ingénierie des connaissances c'est lors de la détermination des questions de compétences. Elle a été conçue pour ceux qui sont nouveaux dans le domaine ontologique. La méthodologie METHONTOLOGY hérite entièrement de l'ingénierie des connaissances, elle est la plus mature de la catégorie « indépendante de l'application » et en plus de cela elle est bien détaillée et précise sur ses différentes activités, ce qui facilite la compréhension et l'évolution de l'ontologie. Pour ces raisons (indépendance de l'application, niveau d'héritage de l'ingénierie des connaissances, maturité et facilité de modification), nous avons choisi la méthodologie METHONTOLOGY.

CHAPITRE III

L'ONTOLOGIE SUR LES ATTRIBUTS ET MODÈLES DE QUALITÉ LOGICIELLE SWQAO (SOFTWARE QUALITY ATTRIBUTES ONTOLOGY)

Dans cette section, nous allons appliquer les étapes de spécification et conceptualisation du processus METHONTOLOGY pour construire notre ontologie. Nous nous limiterons aux termes clés de notre ontologie. Pour une spécification plus détaillée, consultez APPENDICE A.

3.1 Spécification

Domaine : génie logiciel

Objectifs de l'ontologie : permettre à tous les acteurs d'avoir la même compréhension et utilisation des métriques et des attributs de qualité qui s'y rattachent, ainsi que des modèles de qualité qui permettent d'établir des liens entre ces métriques et les attributs internes et externes de la qualité logicielle. Notre ontologie doit être utilisée pour orienter l'utilisateur dans ses choix pendant son travail de conception, codage et maintenance afin que le produit résultant respecte les critères de qualité identifiés comme étant importants pour le produit final. Tout comme cette ontologie devrait centraliser des connaissances precises du génie logiciel; elle doit aussi servir et faciliter la diffusion de ces connaissances du

domaine. Elle doit servir de guide dans les phases de conception et de programmation orientées-objets. Les experts du domaine, quelque soit leur localisation, pourront enrichir cette ontologie de connaissances pertinentes pour l'amélioration continue de la qualité logicielle.

Portée de l'ontologie : notre ontologie se focalise sur la maintenabilité, la modularité et la réutilisabilité comme attributs de qualité logicielle, sur les modèles de qualité les plus populaires et les métadonnées de provenance des métriques liées aux attributs de qualité cités ci-dessus. Notre ontologie doit permettre de répondre, au moins, aux questions suivantes (*competency questions*) :

- ✓ Quels sont les attributs de qualité mesurables ?
- ✓ Quels sont les attributs de qualité non mesurables ?
- ✓ Quelles sont les métriques qui permettent de mesurer tel attribut de qualité mesurable ?
- ✓ Quels sont les différents types de modèles de qualité ?
- ✓ Quels sont les modèles de qualité les plus populaires ?
- ✓ Comment a été obtenu tel modèle de qualité ?
- ✓ Quelle est la structure de chaque modèle de qualité ?
- ✓ Quels sont les différents types de métriques ?
- ✓ Comment est définie une métrique donnée ?
- ✓ Quelle est la formule d'une métrique donnée ?
- ✓ D'où proviennent ces métriques ?

- Liste des concepts clés :
    - Métrique, métriques de conception, métriques de couplage classe-attribut, métriques de couplage classe-méthode, métriques de couplage méthode-méthode, métriques d'implémentation, métriques de couplage entre classes, métriques de cohésion, métriques de complexité, métriques d'héritage, modification, attributs de qualité, maintenabilité, attributs de qualité mesurables, cohésion, complexité, couplage, couplage classe-attribut, couplage entre classe, couplage classe-méthode, couplage méthode-méthode, héritage, attributs de qualité non mesurables, réutilisabilité, modularité, impact, effort, propension aux fautes, état.
    - Modèle de qualité, réseaux bayésiens, règles, arbres de décision, composantes des modèles de qualité : table de probabilité (variables d'entrée, variables de sortie, variables intermédiaires), structure (nœud d'entrée, nœud de sortie, nœud intermédiaire), condition d'une règle, nœuds de décision, racine, feuille, arc, pertinence, probabilité, poids.
    - Document, provenance, article, livre, évènement (conférence, workshop, symposium), organisme, auteur.
- Liste des instances clés :
    - Métriques : ACAIC, ACMIC, AID, AMMIC, CBO, CBO', CBOIUB, CBONA, CBOU, CIS, CLD, Co, Coh, CSB, DAC, DAC', DCAEC, DCMEC, DIT, DMMEC, FCAEC, FCMEC, FMMEC, ICH, ICP, IFCAIC, IFCMIC, IFMMIC, IH-ICP, LCC, LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, MPC, NAD, NIH-ICP, NMA, NMI,

NMO, NOA, NA, NOC, NOD, NOM, NOO, NOP, NOPM, NOT, NPA, NPM, NRA, OCAEC, OCAIC, OCMEC, OCMIC, OMMEC, OMMIC, RFC, RFC', RFCAlpha, SIX, TCC.

- Modèles de qualité : BNScenario2[17], BNScenario3, BNScenario4, R1[18], ..., R15.

- Liste des relations clés :

  hasComputationalInfluenceOn, from, hasBayesianNetworksNode, hasCondition, hasConsequence, hasCausalRelationWith, hasDefinition alRelationWith, hasDecisionTreeNode, hasImpact, hasInput, hasMetric, hasParameters, hasParent, hasProbabilityTable, hasState, isProbabilityTableOf, hasStructure, isQuantifiedBy, hasSubcharacteristic, isSubcharacteristicOf, quantifies, isParentOf, representsMetric, representsQualityCharacteristic, hasPertinence.

- Liste des propriétés :

  hasFormula, hasName, hasValue, hasWeight, hasProbability, successfulCases, unsuccessfulCases.

Provenance des connaissances :

- Bases de connaissances
- Bases de données
- Entretien avec l'expert

[17] BN : Bayesian Network.

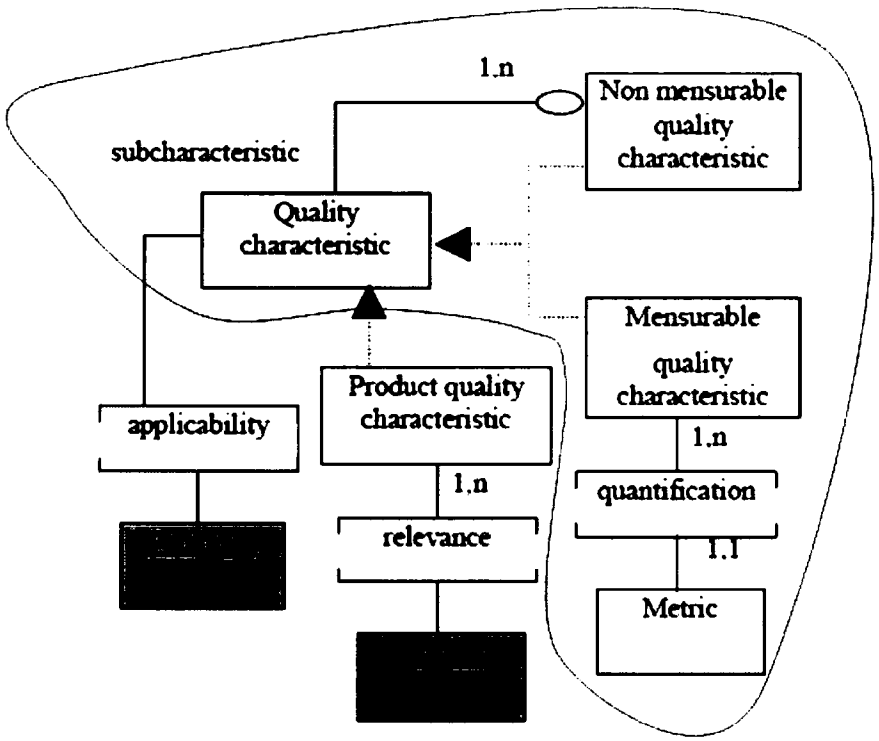
[18] R1 : Règle 1.

- Documents (voir section Bibliographie)

3.2 Conceptualisation

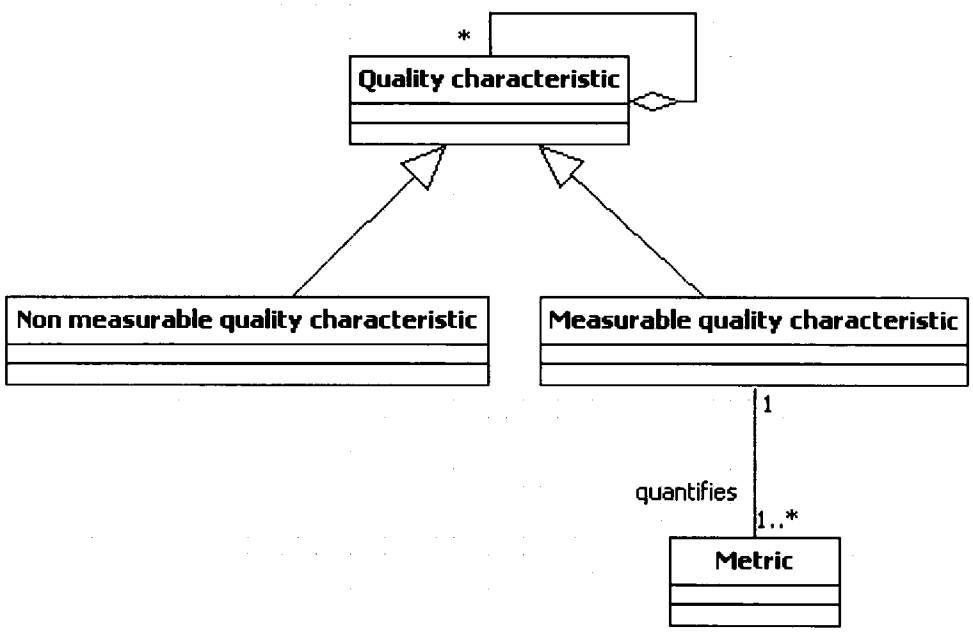
Pendant les phases de spécification et de conceptualisation, il faut voir s'il est possible de réaliser notre ontologie à partir d'une ontologie déjà existante et dans quelle mesure d'autres ontologies peuvent être intégrées à la nôtre. Parmi les ontologies de la qualite logicielle existantes on a l'ontologie sur la qualité logicielle présentée en partie dans [27]. Cette ontologie intègre aussi l'ontologie de processus logiciels. On a aussi Ontology for Anti-patterns, Bad Codes Smells and Refactoring techniques (OABR) [49] qui a pour but d'améliorer le partage et la compréhension des *anti-patterns*, *bad code smells* et le *refactoring*, ainsi que leurs relations dans la communauté du génie logiciel, et ainsi améliorer la qualité logicielle par la détection et l'élimination des problèmes logiciels qui sont détectés par les *anti-patterns* et les *bad codes smells*. Pour le développement de notre ontologie, nous allons nous inspirer de l'ontologie sur la qualité logicielle, car nous nous concentrons sur les attributs et métriques de qualité logicielles qui font partie de cette ontologie à laquelle nous rajouterons les modèles de qualité.

Figure 3.1 Diagramme partiel de software quality ontology [27].



La figure 3.1 présente une partie du diagramme de l'ontologie sur la qualité logicielle. Dans ce diagramme on voit que : les attributs de qualité mesurables, non mesurables et de produit sont les differents types d'attributs de qualité; un attribut de qualité peut être composé d'un ou plusieurs attribut(s) de qualité non mesurable(s); un attribut de qualité produit a une pertinence par rapport à un artefact; un attribut de qualité mesurable est quantifié par une métrique; un attribut de qualité s'applique a un paradigme. Il est à noter que *Paradigm* et *Artifact* sont des concepts de l'ontologie de processus et que *applicability*, *relevance* et *quantification* sont des relations. Il est aussi à noter que ce diagramme utilise une nomenclature différente de celle d'UML. Nous avons légèrement modifié la partie du diagramme de cette ontologie qui nous intéresse (partie encadrée) et l'avons représentée en UML. Ceci nous donne la figure suivante :

Figure 3.2 Diagramme UML partiel de notre ontologie.



Nous allons étendre cette partie (voir figure 3.2) afin de créer l'ontologie qui répondra à nos besoins. L'étape de conceptualisation de METHONTOLOGY consiste à réaliser plusieurs représentations intermédiaires de l'ontologie. Les sous-sections suivantes représentent les différentes représentations intermédiaires de notre ontologie tel qu'indiqué par ce processus.

3.2.1 Dictionnaire des termes de l'ontologie SwQAO

Cette étape de la conceptualisation consiste à décrire tous les termes de notre ontologie. Cependant, nous ne décrirons que les termes clés dans cette section. Pour la description des termes restants, consultez APPENDICE B.

Description des attributs et modèles de qualité

Attributs de qualité : ce sont des critères et des facteurs qui permettent de déterminer le degré de qualité d'un logiciel.

Attributs de qualité mesurables : ce sont des attributs de qualité que l'on peut calculer.

Attributs de qualité non mesurables : attributs de qualité non calculables.

Maintenabilité : aptitude d'un logiciel à être facilement modifiable.

Réutilisabilité : aptitude d'un composant logiciel à être utilisé, tout ou en partie, dans un nouveau projet logiciel.

Modularité : aptitude d'un logiciel à être structuré en composants ou modules indépendants. Évaluer la modularité revient à juger de la pertinence de la fonction de chaque module et de ses interactions avec les autres modules.

Impact : attribut de qualité qui renseigne sur les effets secondaires d'une modification apportée à un logiciel.

Effort : attribut de qualité qui renseigne sur l'effort nécessaire pour accomplir une tâche (modifier un module, corriger une erreur, etc.).

Propension aux fautes : c'est la tendance qu'un module a, à avoir des fautes.

Couplage : décrit la force des liens qui existent entre un module composant et les autres modules composants du système.

Couplage de classe : décrit les liens qui existent entre les classes d'un système.

Couplage classe-attribut : décrit la relation classe-attribut qui existe entre deux classes. Deux classes *a* et *b* ont cette relation lorsque la classe *a* est le type d'un attribut de la classe *b*.

Couplage classe-méthode : décrit la relation classe-méthode qui existe entre deux classes. Deux classes *a* et *b* ont cette relation si : (1) la classe a est le type d'un paramètre de la méthode m_b de la classe *b*, (2) la classe *a* est le type de retour de la méthode m_b.

Couplage méthode-méthode : décrit la relation méthode-méthode qui existe entre deux classes. Deux classes *a* et *b* ont cette relation si : (1) la méthode m_b de la classe *b* invoque directement la méthode m_a de la classe *a*, (2) la méthode m_b reçoit un pointeur sur m_a comme paramètre, c'est-à-dire m_b invoque m_a indirectement.

Cohésion : mesure la force de dépendance entre les éléments composant un module.

Complexité : attribut de qualité mesurable qui renseigne sur le degré de simplicité d'un module.

Héritage : permet de propager les fonctionnalités d'un objet à un autre.

Modification : tout changement apporté à un logiciel.

Métrique : c'est une mesure des propriétés d'un logiciel.

Métriques de conception : ce sont des métriques qui portent sur la conception d'un logiciel. Elles sont obtenues à partir des modèles conceptuels d'un logiciel.

Métriques d'implémentation : ce sont des métriques obtenues à partir du code source.

Modèle de qualité : c'est un modèle qui a pour objectif de décrire, d'évaluer et/ou de prédire la qualité. [41]

Réseaux bayésiens : modèle de qualité qui est composé d'un graphe et de tables de probabilités proportionnelles au nombre de nœuds du graphe. Le graphe, encore appelé structure, est composé de nœuds d'entrée, intermédiaires et de sortie.

Paramètres : ce sont les paramètres du réseau bayésien. Ces paramètres sont les variables d'entrée, intermédiaires et de sortie, des nœuds correspondants.

Variables d'entrée : les valeurs de ces variables influencent les variables intermédiaires.

Variables intermédiaires : les valeurs de ces variables influencent les variables de sortie.

Variables de sortie : les valeurs de ces variables ont une influence sur les attributs de qualité représentés par les nœuds de sortie.

Structure : c'est la topologie d'un réseau bayésien. Elle est composée de nœuds d'entrée, intermédiaires et de sortie. Ces nœuds sont reliés entre eux par des arcs orientés.

Nœud d'entrée : il a comme entrée une métrique qui a une influence computationnelle sur les nœuds intermédiaires.

Nœud de sortie : il représente des attributs de qualité.

Nœud intermédiaire : il représente des catégories de métriques, ces catégories étant obtenues sur la base de certains critères. Comme exemple de catégories, on peut avoir des métriques de conception et des métriques d'implémentation. Tout nœud intermédiaire a une influence computationnelle sur le nœud de sortie.

Règles : c'est un modèle de qualité. Nos règles portent sur les relations entre les métriques subjectives (études empiriques, observations) et les métriques objectives (mesurées à partir du code ou du modèle conceptuel).

Arbre de décision : c'est un modèle hiérarchique dans lequel les régions locales sont identifiées à la suite de divisions récursives.

Nœud de décision : il représente une métrique. Le chemin emprunté à partir de ce nœud dépend de la valeur de la métrique.

Racine : c'est le nœud de départ. Il représente un attribut de qualité.

Feuille : c'est un nœud de fin. Il représente un attribut de qualité.

Description des métriques

ACAIC : la métrique Ancestors Class-Attribute Import Coupling compte toutes les relations classe-attribut de la classe c avec ses ancêtres.

ACMIC : la métrique Ancestors Class-Method Import Coupling compte toutes les relations classe-méthode de la classe c à ses ancêtres.

AID : la métrique Average Inheritance depth calcule la profondeur moyenne de l'arbre d'héritage d'une classe.

AMMIC : la métrique Ancestors Method-Method Import Coupling calcule le nombre de classes parents avec lesquelles une classe a la relation de type méthode-méthode et un couplage de type import.

CBO : la métrique Coupling Between Object classes calcule le nombre de classes auxquelles une classe est couplée. Le couplage entre les classes reliées par l'héritage (couplage basé sur l'héritage) est pris en compte dans le calcul.

CBO' : la métrique CBO' calcule le nombre de classes auxquelles une classe est couplée, sans prendre en compte le couplage basé sur l'héritage.

CBOIUB : la métrique Coupling Between Object classes Is Used By réfère aux classes qui utilisent la classe cible.

CBONA : la métrique Coupling Between Object No Ancestors calcule le nombre de classes auxquelles une classe est couplée, sans considérer les classes ancêtres.

CBOU : la métrique Coupling Between Object Using réfère aux classes utilisées par la classe cible.

CIS : la métrique Class Interface Size compte le nombre de méthodes publiques dans une classe.

CLD : la métrique Class-to-Leaf Depth donne le nombre maximal de niveaux d'héritage d'une classe à une feuille qui appartient à la même hiérarchie.

Co : la métrique Connectivity est définie par la formule Co = 2 (|E| - (|V| - 1)) / ((|V| - 1) (|V| - 2)). Soit G un graphe, V représente le nombre de sommets de G et E le nombre d'arêtes de G.

Coh : est une variante de la métrique LCOM5.

CSB : la métrique Class Size in Bytes calcule la somme de la taille de tous les attributs déclarés.

DAC : la métrique Data Abstraction Coupling compte le nombre d'attributs d'une classe qui ont une autre classe comme type.

DAC' : La métrique DAC' compte le nombre de classes différentes qui sont utilisées comme types d'attributs dans une classe.

DCAEC : la métrique Descendents Class-Attribute Export Coupling compte toutes les relations classe-attribut des descendants de la classe c à la classe c.

DCMEC : la métrique Descendents Class-Method Export Coupling compte toutes les relations classe-méthode des descendants de la classe c à la classe c.

DIT : la métrique Depth of Inheritance Tree calcule la distance maximale entre une classe et la classe racine de l'arbre d'héritage.

DMMEC : la métrique Descendents Method-Method Export Coupling calcule le nombre de sous-classes avec lesquelles une classe a la relation de type méthode-méthode et un couplage de type export.

FCAEC : la métrique Friends Class Attribute Export Coupling compte toutes les relations classe-attribut des amis de la classe c à la classe c.

FCMEC : la métrique Friends Class-Method Export Coupling compte toutes les relations classe-méthode des amis de la classe c à la classe c.

FMMEC : la métrique Friends Method-Method Export Coupling compte toutes les relations méthode-méthode des amis d'une classe c à la classe c.

ICH : la métrique Information-flow-based cohesion pour une classe est définie comme le nombre d'appels d'autres méthodes de la même classe, pondéré par le nombre de paramètres de la méthode appelée.

ICP : la métrique Information-flow-based coupling calcule le nombre de méthodes de la classe c qui invoquent les methodes de la classe d, pondéré par le nombre de paramètres des méthodes invoquées.

IFCAIC : la métrique Inverse Class-Attribute Import Coupling compte toutes les relations classe-attribut de la classe c à ses amis inverses.

IFCMIC : la métrique Inverse Friend Class-Method Import Coupling compte toutes les relations classe-méthode de la classe c à ses amis inverses.

IFMMIC : la métrique Inverse Friend Method-Method Import Coupling compte toutes les relations méthode- méthode de la classe c à ses amis inverses.

IH-ICP : comme ICP, mais compte le nombre d'invocations des méthodes des ancêtres des classes.

LCC : la métrique Loose Class Cohesion calcule le pourcentage de paires de méthodes publiques d'une classe qui sont directement ou indirectement connectées.

LCOM1 : la métrique Lack of Cohesion in Methods 1 compte le nombre de paires de méthodes dans une classe qui n'utilisent aucun attribut commun.

LCOM2 : la métrique Lack of Cohesion in Methods 2 compte le nombre de paires de méthodes dans une classe qui n'utilisent aucun attribut commun, moins le nombre de paires de méthodes qui ont des attributs communs. Si cette différence est négative alors la métrique prend la valeur zéro.

LCOM3 : la métrique Lack of Cohesion in Methods 3 compte le nombre de composants connectés d'un graphe G. G étant un graphe non orienté ou les sommets correspondent aux méthodes d'une classe et il y a une arête entre deux sommets si les méthodes correspondant à ces deux sommets utilisent au moins un attribut en commun.

LCOM4 : la métrique Lack of Cohesion in Methods 4 fait le même calcul que LCOM3 et ajoute à ce calcul les arêtes entre les sommets représentants les méthodes qui s'appellent, quelque soit le sens de l'appel.

LCOM5 : la métrique Lack of Cohesion in Methods est spécifiée à partir du nombre de fonctions accédant à chaque attribut.

MPC : la métrique Message Passing Coupling compte le nombre d'invocations de méthodes dans une classe.

NAD : la métrique Number of Abstract Data types compte le nombre d'objets définis par l'utilisateur.

NIH-ICP : compte les invocations des classes qui ne sont pas liées par héritage.

NMA : la métrique Number of Methods Added compte le nombre de nouvelles méthodes d'une classe, qui ne sont ni héritées, ni surchargées.

NMI : la métrique Number of Methods Inherited compte le nombre de méthodes héritées, mais non surchargées.

NMO : la métrique Number of Methods Overridden compte le nombre de méthodes héritées qui ont été redéfinies dans la sous-classe.

NOA : la métrique Number Of Ancestors compte le nombre de classes distinctes à partir desquelles une classe hérite directement ou indirectement.

NA : la métrique Number of Attributes compte le nombre d'attributs d'une classe.

NOC : la métrique Number Of Children compte le nombre de sous-classes qui héritent directement d'une classe donnée.

NOD : la métrique Number Of Descendents compte le nombre de classes qui héritent directement ou indirectement d'une classe.

NOM : la métrique Number Of Methods compte le nombre de méthodes locales d'une classe.

NOO : la métrique Number of Overloaded Operators compte le nombre de méthodes opérateurs surchargées définies dans une classe.

NOP : la métrique Number Of Parents compte le nombre de classes à partir desquelles une classe donnée hérite directement.

NOPM : la métrique Number Of Polymorphic Methods compte le nombre de méthodes qui peuvent montrer un comportement polymorphique, y compris surchargé.

NOT : la métrique Number Of Trivial methods compte le nombre de méthodes triviales qui sont marquées comme incorporées dans une classe.

NPA : la métrique Number of Public Attributes compte le nombre d'attributs déclarés publics dans une classe.

NPM : la métrique Number of Parameters per Method compte le nombre moyen de paramètres par méthode dans une classe.

NRA : la métrique Number of Reference Attributes compte le nombre de pointeurs et de références utilisés comme attributs dans une classe.

OCAEC : la métrique Other Class-Attribute Export Coupling compte les relations classe-attribut des classes qui ne sont ni amis ni descendants de la classe c.

OCAIC : la métrique Other Class-Attribute Import Coupling compte les relations de la classe c aux classes qui ne sont ni ancêtres ni amis inverses de la classe c.

OCMEC : la métrique Others Class-Method Export Coupling compte toutes les relations classe-méthode des classes qui ne sont ni amis ni descendants de la classe c.

OCMIC : la métrique Others Class-Method Import Coupling compte toutes les relations classe-méthode de la classe c aux classes qui ne sont ni ses ancêtres ni ses amis inverses.

OMMEC : la métrique Others Method-Method Export Coupling compte toutes les relations méthode-méthode des classes qui ne sont ni amis ni descendants de la classe c.

OMMIC : la métrique Others Method-Method Import Coupling compte toutes les relations méthode-méthode de la classe c aux classes qui ne sont ni ancêtres ni amis inverses de la classe c.

RFC : la métrique Response set For Class donne un ensemble de méthodes M d'une classe et un ensemble de méthodes directement et indirectement invoquées par les méthodes contenues dans M.

RFC' : c'est le nombre de méthodes qui peuvent être possiblement invoquées en réponse à un message envoyé à une classe c, incluant le nombre de ses méthodes, les méthodes invoquées par les méthodes de celle-ci et ainsi de suite.

RFCAlpha : la métrique Response For Class Alpha est une version de la métrique RFC qui prend en compte le niveau d'emboitement des invocations de méthodes.

SIX : la métrique Specialization IndeX est le rapport du produit NMO et DIT sur le nombre total de méthodes d'une classe donnée.

TCC : la métrique Tight Class Cohesion considère les méthodes qui utilisent les attributs directement (en les référençant) ainsi que les attributs utilisés indirectement par une méthode.

Description des relations clés

from : c'est une relation qui renseigne sur la provenance d'une métrique ou d'autres termes.

hasArc : c'est une relation qui nous dit qu'un arbre de décision est composé d'arcs.

hasBayesianNetworksNode : c'est une relation qui nous dit qu'un réseau bayésien est composé de nœuds.

hasComputationalInfluenceOn : c'est une relation qui nous dit que chaque nœud d'un réseau bayésien a une influence computationnelle sur ses nœuds enfants.

hasDecisionTreeNode : c'est une relation qui nous dit qu'un arbre de décision est composé de nœuds.

hasImpact : c'est une relation qui nous dit que toute modification apportée à un logiciel a un impact sur celui-ci.

hasInput : c'est une relation qui nous dit que tout nœud d'entrée d'un réseau bayésien reçoit des métriques comme données en entrée.

hasParameters : c'est une relation qui nous dit qu'un réseau bayésien a des paramètres. hasParent : c'est une relation qui nous dit si un nœud est un nœud enfant ou pas.

hasProbabilityTable : c'est une relation qui nous dit que chaque nœud d'un réseau bayésien est lié à une table de probabilités.

hasCausalRelationWith : définit la relation de causalité entre deux nœuds d'un réseau bayésien.

hasDefinitionalRelationWith : définit la relation de définition qui existe entre un nœud parent et son enfant.

hasConsequence : c'est une relation qui nous dit qu'une règle a une partie conséquence qui est un ou plusieurs attributs de qualité non mesurables.

hasStructure : c'est une relation qui nous dit qu'un réseau bayésien a une structure qui est composée de nœuds reliés entre eux par des arcs.

hasSubcharacteristic : c'est une relation qui nous renseigne sur les sous-attributs d'un attribut de qualité.

isParentOf : c'est une relation qui nous dit si un nœud est un nœud parent ou pas.

isProbabilityTableOf : c'est une relation qui nous renseigne sur le fait que les variables contenues dans la table de probabilités se rapportent au nœud du réseau bayésien correspondant.

isQuantifiedBy : c'est la relation qui existe entre un attribut de qualité mesurable et une métrique.

isSubcharacteristicOf : c'est une relation qui nous renseigne sur l'attribut parent d'un attribut donné.

quantifies : c'est la relation qui existe entre une métrique et un attribut de qualité mesurable.

representsMetric : c'est une relation qui indique la métrique qu'un nœud représente.

representsQualityCharacteristic : c'est une relation qui indique l'attribut de qualité qu'un nœud représente.

aComparisonSign : relation utilisée dans une règle pour comparer une métrique par rapport à un seuil.

hasFormula : c'est une propriété qui donne la formule d'une métrique.

hasCondition : c'est la relation qui indique qu'une règle a une partie condition qui respecte un format précis.

hasName : c'est une propriété qui donne le nom d'une instance.

hasValue : c'est une propriété qui donne la valeur numérique d'une métrique.

hasWeight : c'est le poids d'une règle qui indique le nombre de cas que la règle réussit à correctement classifier versus le nombre de cas pour lesquels elle échoue.

hasProbability : c'est la probabilité pour que la conséquence d'une règle ait lieu en fonction d'une condition précise, lorsqu'on utilise un modèle de qualité probabiliste.

Les tableaux suivant résument les métriques utilisées dans notre ontologie et leurs provenances. La provenance d'un terme renseigne sur l'auteur ou les auteurs du terme, le document dans lequel le terme a été rendu public, le lieu, la date, etc.

Tableau 3.1 Provenance des métriques

Terme (métriques)	Provenance
CIS, NOPM	C. Davis, J. Bansiya, A hierarchical model for object-oriented design quality assessment, IEEE transactions on software engineering, vol. 28, no. 1, pp. 4-17, January 2002
NRA	C. Davis, J. Bansiya, Automated metrics and object-oriented development : Using QMOOD++ for object-oriented metrics, Dr Dobbs Journal, pp. 44-48, vol. 22, no. 12, December 1997
TCC, LCC	J. M. Bieman, B.-K. Kang, Cohesion and reuse in an object- oriented system, Symposium on software reusability, pp. 259- 262, 1995
ACAIC, DCAEC, FCAEC, IFCAIC, OCAIC, OCAEC, ACMIC, DCMEC, FCMEC, IFCMIC, OCMIC, AMMIC, DMMEC, FMMEC, OCMEC, IFMMIC, OMMEC, OMMIC, Coh	L. Briand, P. Devanbu, W. Melo, An investigation into coupling measures for C++, 19th International conférence on software engineering, Boston, pp. 412-421, May 1997
RFC`	L. Briand, J. Wuest, J. Daly, A unified framework for coupling measurement in object-oriented systems, IEEE transactions on software engineering, vol. 25, no. 1, pp. 91-121, 1999
CBOIUB, CBONA, CBOU	M. A. Chaumun, H. Kabaili, R. K. Keller, F. Lustman, G. Saint-Denis, Design properties and object-oriented software changeability, Conference on software maintenance and reengineering, pp. 45-54, February 2000
CBO, LCOM2	S. R. Chidamber, C. F. Kemerer, A metrics suite for object- oriented design, IEEE transactions on software engineering, vol. 20, no. 6, pp. 476-493, June 1994
CBO', RFC, LCOM1, RFC∞, DIT, NOC	S. R. Chidamber, C. F. Kemerer, Towards a metrics suite for object-oriented design, Conference on object-oriented programming : Systems, languages and applications, vol. 26, no. 11, pp. 197-211, October 1991
CSB	K. Emam, W. Melo, N. Goel, S. Benlarbi, H. Lounis, S. Rai, The optimal class size for object-oriented software, IEEE transactions on software engineering, vol. 28, no. 5, May 2002
Co, LCOM3, LCOM4, LCOM5	M. Hitz, B. Montazeri, Measuring coupling and cohesion in object-oriented systems, International symposium on applied corporate computing,October 1995
NOP	A. Lake, C. Cook, Use of factor analysis to develop OOP software complexity metrics, Annual Oregon workshop on software metrics, April 1994

Tableau 3.2 Provenance des métriques (suite et fin)

Terme (métriques)	Provenance
DAC, DAC', MPC	W. Li, S. Henry, Object-oriented metrics that predict maintainability, Journal of systems and software, vol. 23, no. 2, pp. 111-122, November 1993
NOM	W. Li, S. Henry, Object-oriented metrics which predict maintainability, 35 pages, February 1993
ICP, IH-ICP, NIH-ICP, ICH	B. Liang, F.J. Wang, S.F. Wu, Y.S. Lee, Measuring the coupling and cohesion of an object-oriented program based on information flow, International conference on software quality, Maribor, Slovenia, 1995
NAD, NOA, NOD	W. Li, Another metric suite for object-oriented programming, Journal of systems and software, vol. 44, no. 2, pp. 155-162, 1998
NOT	J. Michura, M. Capretz, S. Wang, Extension of object-oriented metrics suite for software maintenance, International scholarly research network software engineering, 14 pages, 2013
NPA	M. Lanza, R. Marinescu, Object-oriented metrics in practice : Using software metrics to characterize, evaluate, and improve the design of object-oriented systems, Springer, 2006
NPM, AID, NMA, NMO, SIX	B. Henderson-Sellers, Object-oriented metrics : Measures of complexity, Prentice Hall, 1996
NA, NMA, NMI	M. Lorenz, J. Kidd, Object-oriented software metrics, Prentice Hall, 1994
CLD	D. P. Tegarden, S. D. Sheetz, D. E. Monarchi, A software complexity model of object-oriented systems, decision support systems, vol. 13, no. 34, pp. 241-262, March 1993

3.2.2 Construction des arbres de classification des concepts

Dans cette partie, on construit la taxonomie des concepts de notre ontologie. Nous avons utilisé l'approche middle-out qui consiste tout d'abord à identifier les concepts importants de notre domaine d'application et par la suite effectuer des généralisations ou spécialisations à partir des ces concepts importants.

Les concepts importants dans notre cas sont les attributs de qualité, les métriques, les modèles de qualité et les métadonnées de provenance des métriques. Comme mentionné dans l'introduction les attributs de qualité importants dans notre cas sont la maintenabilité et ses sous-attributs modularité et réutilisabilité. Il n'existe pas de formules ou de méthodes directes pour donner une valeur précise à ces attributs. Pour cette raison, ils se classent dans la catégorie d'attributs de qualité non mesurables. À cette catégorie s'ajoutent d'autres concepts tels que modification, impact, effort et propension aux fautes. En effet dans la phase de maintenance on effectue des modifications qui ont des impacts sur le reste du code ou le produit. Il est important de pouvoir estimer la propension aux fautes de l'application et l'effort de conception, d'implémentation et de maintenance. La modularité et la réutilisabilité ont comme sous-attibuts le couplage, la cohésion, la complexité et l'héritage, et ces sous-attributs sont mesurables grâce à des métriques qu'on peut classifier en deux groupes : métriques de conception et métriques d'implémentation. Parmis les métriques de conception on distingue les métriques de couplage classe-attributs, classe-méthode, méthode-méthode et parmis les métriques d'implémentation on distingue les métriques de cohésion, complexité et d'héritage. Comme modèles de qualité, on s'est intéréssé aux modèles les plus utilises, à savoir les réseaux bayésiens, arbres de décision et règles, ainsi qu'à leurs structures et leurs composantes (arcs, noeuds, paramètres, poids, etc.). Comme métadonnées nous avons identifié des concepts tels que : personnes (auteurs), livres, articles, organisations, etc. Voici à quoi ressemble la taxonomie des concepts de notre ontologie :

- Metric
  - DesignMetric
    - ClassAttributeCouplingMetric
    - ClassMethodCouplingMetric
    - MethodMethodCouplingMetric
  - ImplementationMetric
    - ClassCouplingMetric
    - CohesionMetric
    - ComplexityMetric
    - InheritanceMetric
- Modification
- QualityCharacteristic
  - MeasurableQualityCharacteristic
    - Cohesion
    - Complexity
    - Coupling
    - ClassAttributeCoupling
      - ClassCoupling
      - ClassMethodCoupling
      - MethodMethodCoupling
    - Inheritance
  - NonMeasurableQualityCharacteristic
    - Maintainability
    - Reusability
    - Modularity
    - Impact
    - Effort
    - FaultProneness
- QualityModel
  - BlackBoxModel
  - GlassBoxModel
    - BayesianNetwork
    - DecisionTree
    - Rules
- QualityModelComponents
  - Arc
  - MixType
    - SimpleCondition
    - CombinedCondition
  - Node
    - BayesianNetworksNode
      - EndNode
      - EntryNode
      - IntermediateNode
    - DecisionTreeNode
      - DecisionNode
      - Leaf
      - Root
  - Parameter
    - EntryVariableParameter
    - EndVariableParameter
    - IntermediateVariableParameter
  - Pertinence
    - Probability
    - Weight
  - Sign
    - ComparisonSign
  - State
  - Structure
- *Occasion*
  - *Conference*
  - *Symposium*
  - *Workshop*
- *Organization*
- *Person*
- *Provenance*
  - *Article*
  - *Book*

Les concepts en italique souligné de cette hiérarchie sont utilisés pour fournir les informations sur la provenance des métriques. Ces concepts sont équivalents à des concepts d'ontologies déjà existantes et populaires. Par exemple, le concept *Occasion* de notre ontologie est équivalent au concept *Event* de l'ontologie event.owl[19]. Les concepts *Person* et *Organization* de notre ontologie sont respectivement équivalents aux classes *Person* et *Organization* du vocabulaire FOAF[20]. Le concept *Provenance* de notre ontologie est équivalent au concept *Document* de l'ontologie bibliographique BIBO[21].

3.2.3 Dictionnaire des concepts

Nous avons réaliser le dictionnaire des concepts sous forme de tableau. Ce tableau ne contient que des concepts clés de notre ontologie. Voir APPENDICE C pour les autres concepts.

[19] http ://motools.sourceforge.net/event/event.html

[20] http ://xmlns.com/foaf/spec/

[21] http ://bibliontology.com/specification

Nom du concept	Instances	Attributs des instances	Données sur les instances	Relations
Metric	s/o	hasName, hasFormula	from	quantifies
DesignMetric	s/o	s/o	s/o	s/o
ClassAttributeCouplingMetric	ACAIC, DCAEC, FCAEC, IFCAIC, OCAEC, OCAIC	s/o	s/o	s/o
ClassMethodCouplingMetric	ACMIC, DCMEC, FCMEC, IFCMIC, OCMEC, OCMIC	s/o	s/o	s/o
MethodMethodCouplingMetric	AMMIC, DMMEC, FMMEC, IFMMIC, OMMEC, OMMIC	s/o	s/o	s/o
ImplementationMetric	s/o	s/o	s/o	s/o
ClassCouplingMetric	CBO, CBO', CBOIUB, CBONA, CBOU, DAC, DAC', ICP, IH-ICP, MPC, NIH- ICP, RFC, RFC', RFCAlpha	s/o	s/o	s/o
CohesionMetric	Co, Coh, ICH, LCC, LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, TCC	s/o	s/o	s/o
ComplexityMetric	CIS, CSB, NAD, NA, NOM, NOO, NOPM, NOT, NPA, NPM, NRA	s/o	s/o	s/o
InheritanceMetric	AID, CLD, DIT, NMA, NMI, NMO, NOA, NOC, NOD, NOP, SIX	s/o	s/o	s/o

Nom du concept	Instances	Attributs des instances	Données sur les instances	Relations
Modification	s/o	s/o	s/o	hasImpact
Node	s/o	s/o	s/o	hasParent, isParentOf, representsMetric, representsQC
BayesianNetworksNode	s/o	s/o	s/o	hasProbabilityTable, hasCausalRelationWith, hasDefinitionalRelationWith
EndNode	s/o	s/o	s/o	s/o
EntryNode	s/o	s/o	s/o	hasInput
IntermediateNode	s/o	s/o	s/o	s/o
DecisionTreeNode	s/o	s/o	s/o	s/o
DecisionNode	s/o	s/o	s/o	s/o
Leaf	s/o	s/o	s/o	s/o
Root	s/o	s/o	s/o	s/o
Parameter	s/o	s/o	s/o	isProbabilityTableOf, hasComputationalInfluenceOn
EntryVariable-Parameter	s/o	s/o	s/o	s/o
EndVariable-Parameter	s/o	s/o	s/o	s/o
Intermediate-VariableParameter	s/o	s/o	s/o	s/o

Nom du concept	Instances	Attributs des instances	Données sur les instances	Relations
QualityCharacteristic	s/o	s/o	s/o	hasSubcharacteristic
MeasurableQuality Characteristic	s/o	s/o	s/o	isSubcharacteristicOf, isQuantifiedBy
Cohesion	s/o	s/o	s/o	s/o
Complexity	s/o	s/o	s/o	s/o
Coupling	s/o	s/o	s/o	s/o
ClassAttributeCoupling	s/o	s/o	s/o	s/o
ClassCoupling	s/o	s/o	s/o	s/o
ClassMethodCoupling	s/o	s/o	s/o	s/o
MethodMethodCoupling	s/o	s/o	s/o	s/o
Inheritance	s/o	s/o	s/o	s/o
NonMeasurable QualityCharacteristic	s/o	s/o	s/o	hasSubcharacteristic
Maintainability	s/o	s/o	s/o	s/o
Reusability	s/o	s/o	s/o	s/o
Modularity	s/o	s/o	s/o	s/o
Impact	Impact, ImpactR1, ..., ImpactR15	s/o	s/o	s/o
Effort	s/o	s/o	s/o	s/o
FaultProneness	s/o	s/o	s/o	s/o
QualityModel	s/o	s/o	s/o	s/o
BlackBoxModel	s/o	s/o	s/o	s/o
GlassBoxModel	s/o	s/o	s/o	s/o
BayesianNetwork	s/o	s/o	s/o	hasParameters, hasStructure
DecisionTree	s/o	s/o	s/o	hasArc, hasDTNode
Rules	R1, ..., R15	hasName	s/o	hasCondition, hasConsequence, hasPertinence
Structure	s/o	s/o	s/o	hasBNNode

3.2.4 Construction de la table des relations binaires

Cette étape consiste à donner plus de détails sur les relations listées dans le glossaire des concepts en fournissant leurs cardinalités et les concepts qu'elles relient. Voir APPENDICE D pour le reste des relations binaires.

Nom de la relation : hasState
Concept source : THING
Cardinalité de la source : 1
Concept cible : State
Cardinalité de la cible : 1, n
Relation inverse : -

Nom de la relation : from
Concept source : THING
Cardinalité de la source : 1
Concept cible : Provenance
Cardinalité de la cible : 1, n
Relation inverse : -

Nom de la relation : hasCondition
Concept source : Rules, CombinedCondition
Cardinalité de la source : 1, n
Concept cible : MixType
Cardinalité de la cible : 1, n
Relation inverse : -

Nom de la relation : hasConsequence Concept source : Rules
Cardinalité de la source : 1
Concept cible : NonMeasrurableQualityCharacteristic
Cardinalité de la cible : 1, n
Relation inverse : -

Nom de la relation : hasPertinence
Concept source : Rules
Cardinalité de la source : 1
Concept cible : Pertinence
Cardinalité de la cible : 1
Relation inverse : -

Nom de la relation : hasCausalRelationWith
Concept source : BayesianNetworksNode
Cardinalité de la source : 1
Concept cible : BayesianNetworksNode
Cardinalité de la cible : 1, n
Relation inverse : -

Nom de la relation : isParentOf
Concept source : Node
Cardinalité de la source : 1, n
Concept cible : Node
Cardinalité de la cible : 0, n
Relation inverse : hasParent

Nom de la relation : hasImpact
Concept source : Modification
Cardinalité de la source : 1
Concept cible : Impact
Cardinalité de la cible : 1
Relation inverse : -

Nom de la relation : hasInput
Concept source : EntryNode
Cardinalité de la source : 1, n
Concept cible : Metric
Cardinalité de la cible : 1, n
Relation inverse : -

Nom de la relation : hasParameters
Concept source : BayesianNetwork
Cardinalité de la source : 1
Concept cible : Parameter
Cardinalité de la cible : 2, n
Relation inverse : -

Nom de la relation : hasParent
Concept source : Node
Cardinalité de la source : 1, n
Concept cible : Node
Cardinalité de la cible : 0, n
Relation inverse : isParentOf

Nom de la relation : hasProbabilityTable
Concept source : BNNode
Cardinalité de la source : 1
Concept cible : Parameter
Cardinalité de la cible : 2, n
Relation inverse : isProbabilityTableOf

Nom de la relation : hasStructure
Concept source : BayesianNetwork
Cardinalité de la source : 1
Concept cible : Structure
Cardinalité de la cible : 1
Relation inverse : -

Nom de la relation : hasSubcharacteristic
Concept source : QualityCharacteristic
Cardinalité de la source : 1
Concept cible : QualityCharacteristic
Cardinalité de la cible : 1, n
Relation inverse : isSubCharacteristicOf

Nom de la relation : representsMetric
Concept source : Node
Cardinalité de la source : 1
Concept cible : Metric
Cardinalité de la cible : 1, n
Relation inverse : -

Nom de la relation : representsQC
Concept source : Node
Cardinalité de la source : 1
Concept cible : QualityCharacteristic
Cardinalité de la cible : 1, n
Relation inverse : -

Nom de la relation : isProbabilityTableOf
Concept source : Parameter
Cardinalité de la source : 1
Concept cible : BNNode
Cardinalité de la cible : 1, n
Relation inverse : hasProbabilityTable

Nom de la relation : isQuantifiedBy
Concept source : MeasurableQualityCharacteristic
Cardinalité de la source : 1
Concept cible : Metric
Cardinalité de la cible : 1, n
Relation inverse : quantifies

Nom de la relation : isSubcharacteristicOf
Concept source : QualityCharacteristic
Cardinalité de la source : 1, n
Concept cible : QualityCharacteristic
Cardinalité de la cible : 1
Relation inverse : hasSubcharacteristic

Nom de la relation : quantifies
Concept source : Metric
Cardinalité de la source : 1, n
Concept cible : MeasurableQualityCharacteristic
Cardinalité de la cible : 1
Relation inverse : isQuantifiedBy

Nom de la relation : hasDefinitionalRelationWith
Concept source : BayesianNetworksNode
Cardinalité de la source : 1, n
Concept cible : BayesianNetworksNode
Cardinalité de la cible : 1
Relation inverse : -

Nom de la relation : hasBNNode
Concept source : Structure
Cardinalité de la source : 1
Concept cible : BNNode
Cardinalité de la cible : 3, n
Relation inverse : -

Nom de la relation : hasComputationalInfluenceOn
Concept source : BNNode
Cardinalité de la source : 1, n
Concept cible : BNNode
Cardinalité de la cible : 1, n
Relation inverse : -

Nom de la relation : hasDTNode
Concept source : DecisionTree
Cardinalité de la source : 1
Concept cible : DTNode
Cardinalité de la cible : 3, n
Relation inverse : -

3.2.5 Construction de la table des instances

Cette étape consiste à donner plus de détails sur les instances listées dans le glossaire des concepts en fournissant les valeurs de leurs attributs. Nous listons uniquement les métriques dans cette section. Voir APPENDICE E pour le reste des instances.

Instance	**Attributs**	**Valeur**
ACAIC	Nom	Ancestors Class-Attribute Import Coupling
	Formule	$ACAIC(c) = \sum_{d \in Ancestors(c)} CA(c, d)$, $CA(c, d) = \lvert \{a \mid a \in A_I(c^{\downarrow}) and T(a) = d\} \rvert$

Instance	**Attributs**	**Valeur**
DCAEC	Nom	Descendents Class-Attribute Export Coupling
	Formule	$DCAEC(c) = \sum_{d \in Descendents(c)} CA(d, c)$

Instance	**Attributs**	**Valeur**
FCAEC	Nom	Friends Class-Attribute Export Coupling
	Formule	$FCAEC(c) = \sum_{d \in Friends(c)} CA(d, c)$

Instance	**Attributs**	**Valeur**
IFCAIC	Nom	Inverse Friend Class-Attribute Import Coupling
	Formule	$IFCAIC(c) = \sum_{d \in Friends^{-1}(c)} CA(c, d)$

Instance	**Attributs**	**Valeur**
OCAEC	Nom	Other Class-Attribute Export Coupling
	Formule	$OCAEC(c) = \sum_{d \in Others(c) \cup Friends^{-1}(c)} CA(d, c)$

Instance	**Attributs**	**Valeur**
OCAIC	Nom	Other Class-Attribute Import Coupling
	Formule	$OCAIC(c) = \sum_{d \in Others(c) \cup Friends(c)} CA(c, d)$

Instance	**Attributs**	**Valeur**
ACMIC	Nom	Ancestors Class-Method Import Coupling
	Formule	$ACMIC(c) = \sum_{d \in Ancestors(c)} CM(c, d)$, $CM(c, d) = \sum \lvert \{a \mid a \in Par(m) and T(a) = d\} \rvert$

Instance	**Attributs**	**Valeur**
DCMEC	Nom	Descendents Class-Method Export Coupling
	Formule	$DCMEC(c) = \sum_{d \in Descendents(c)} CM(d, c)$

Instance	**Attributs**	**Valeur**
OCMEC	Nom	Others Class-Method Export Coupling
	Formule	$OCMEC(c) = \sum_{d \in Others(c) \cup Friends^{-1}(c)} CM(d, c)$

Instance	**Attributs**	**Valeur**
OCMIC	Nom	Others Class-Method Import Coupling
	Formule	$OCMIC(c) = \sum_{d \in Others(c) \cup Friends(c)} CM(c, d)$

Instance	**Attributs**	**Valeur**
AMMIC	Nom	Ancestors Method-Method Import Coupling
	Formule	$AMMIC(c) = \sum_{d \in Ancestors(c)} MM(c, d)$

Instance	**Attributs**	**Valeur**
DMMEC	Nom	Descendents Method-Method Export Coupling
	Formule	$DMMEC(c) = \sum_{d \in Descendents(c)} MM(d, c)$

Instance	**Attributs**	**Valeur**
FMMEC	Nom	Friends Method-Method Export Coupling
	Formule	$FMMEC(c) = \sum_{d \in Friends(c)} MM(d, c)$

Instance	**Attributs**	**Valeur**
IFMMIC	Nom	Inverse Friend Method-Method Import Coupling
	Formule	$IFMMIC(c) = \sum_{d \in Friends^{-1}(c)} MM(c, d)$

Instance	**Attributs**	**Valeur**
OMMEC	Nom	Other Method-Method Export Coupling
	Formule	$OMMEC(c) = \sum_{d \in Others(c) \cup Friends^{-1}(c)} MM(d, c)$

Instance	**Attributs**	**Valeur**
OMMIC	Nom	Other Method-Method Import Coupling
	Formule	$OMMIC(c) = \sum_{d \in Others(c) \cup Friends(c)} MM(c, d)$

Instance	**Attributs**	**Valeur**
CBO	Nom	Coupling Between Object classes
	Formule	$CBO(c) = \lvert\{d \in C - \{c\} \mid uses(c, d) \vee uses(d, c)\}\rvert$

Instance	**Attributs**	**Valeur**
CBOIUB	Nom	Coupling Between Object classes Is Used By
	Formule	-

Instance	**Attributs**	**Valeur**
CBONA	Nom	Coupling Between Object No Ancestors
	Formule	-

Instance	**Attributs**	**Valeur**
CBOU	Nom	Coupling Between Object Using
	Formule	-

Instance	**Attributs**	**Valeur**
DAC	Nom	Data Abstraction Coupling
	Formule	$DAC(c) = \lvert \{a \mid a \in AI(c) and T(a) \in C\} \rvert$

Instance	**Attributs**	**Valeur**
DAC'	Nom	Data Abstraction Coupling
	Formule	$DAC'(c) = \lvert \{T(a) \mid a \in AI(c) and T(a) \in C\} \rvert$

Instance	**Attributs**	**Valeur**
ICP	Nom	Information-flow-based coupling
	Formule	$ICP(c) = IH - ICP(c) + NIH - ICP(c)$

Instance	**Attributs**	**Valeur**
IH-ICP	Nom	Inheritance Information-flow-based coupling
	Formule	$IH - ICP(c) = \sum_{m \in MI(c)} IH - ICPc(m)$ $IH - ICPc(m) = \sum(1 + \lvert Par(m^t) \rvert) \cdot NPI(m, m^t)$

Instance	**Attributs**	**Valeur**
MPC	Nom	Message Passing Coupling
	Formule	$MPC(c) = \sum\sum NSI(m, m^t)$, $m \in MI(c)$, $m^t \in SIM(m) - MI(c)$

Instance	**Attributs**	**Valeur**
CSB	Nom	Class Size in Bytes
	Formule	$CSB(c) = \sum_{a \in A(c)} size(a)$

Instance	**Attributs**	**Valeur**
NAD	Nom	Number of Abstract Data types
	Formule	-

Instance	**Attributs**	**Valeur**
NA	Nom	Number Of Attributes
	Formule	$NA(c) = \|A(c)\|$

Instance	**Attributs**	**Valeur**
NOM	Nom	Number Of Methods
	Formule	$NOM(c) = \|M(c)\|$

Instance	**Attributs**	**Valeur**
NOO	Nom	Number of Overloaded Operators
	Formule	$NOO(c) = \|MOVL(c)\|$

Instance	**Attributs**	**Valeur**
NOPM	Nom	Number Of Polymorphic Methods
	Formule	-

Instance	**Attributs**	**Valeur**
NOT	Nom	Number Of Trivial methods
	Formule	-

Instance	**Attributs**	**Valeur**
NPA	Nom	Number of Public Attributes
	Formule	$NPA(c) = \|APUB(c)\|$

Instance	**Attributs**	**Valeur**
NMA	Nom	Number of Methods Added
	Formule	$NMA(c) = \lvert MNEW(c) \rvert$

Instance	**Attributs**	**Valeur**
NMI	Nom	Number of Methods Inherited
	Formule	$NMI(c) = \lvert MINH(c) \rvert$

Instance	**Attributs**	**Valeur**
NMO	Nom	Number of Methods Overridden
	Formule	$NMO(c) = \lvert MOVR(c) \rvert$

Instance	**Attributs**	**Valeur**
NOA	Nom	Number Of Ancestors
	Formule	$NOA(c) = \lvert Ancestors(c) \rvert$

Instance	**Attributs**	**Valeur**
NOC	Nom	Number Of Children
	Formule	$NOC(c) = \lvert Children(c) \rvert$

Instance	**Attributs**	**Valeur**
NOD	Nom	Number Of Descendents
	Formule	$NOD(c) = \lvert Descendents(c) \rvert$

Instance	**Attributs**	**Valeur**
NOP	Nom	Number Of Parents
	Formule	$NOP(c) = \lvert Parents(c) \rvert$

Instance	**Attributs**	**Valeur**
NIH-ICP	Nom	Non-inheritance Information-flow-based coupling
	Formule	$NIH-ICP(c)=\sum NIH-ICPc(m)_{m\in MI(c)}$
		$NIH-ICPc(m)=\sum_{m'\in R}(1+\lvert Par(m')\rvert)\cdot NPI(m,m')$
		$R = PIM(m)\cap(U_{c'\in Ancestors(c)}\ M(c'))$

Instance	**Attributs**	**Valeur**
RFC	Nom	Response set For Class
	Formule	$RFC(c) = RFC_1(c)$

Instance	**Attributs**	**Valeur**
RFC'	Nom	Response For Class
	Formule	$RFC'(c) = RFC_\infty(c)$

Instance	**Attributs**	**Valeur**
Co	Nom	Connectivity
	Formule	$Co=2(\lvert E\rvert-(\lvert V\rvert-1))/((\lvert V\rvert-1)(\lvert V\rvert-2))$

Instance	**Attributs**	**Valeur**
ICH	Nom	Information flow based cohesion
	Formule	$ICH=\sum_{m\in MI(c)} ICHc(m),$
		$ICHc(m)=\sum(1+\lvert Par(m')\rvert)\cdot NPI(m,m')$

Instance	Attributs	Valeur
LCOM1	Nom	Lack of COhesion in Methods 1
	Formule	$LCOM1(c)=\{[m1,m2]: m1,m2 \in MI(c) and\, m1 \neq m2\ and\, AR(m1) \cap AR(m2) \cap AI(c)=\emptyset\}$

Instance	Attributs	Valeur
LCOM2	Nom	Lack of COhesion in Methods 2
	Formule	$Ec(c)=\{[m1,m2]: m1,m2 \in Vc(c) and\, AR(m1) \cap AR(m2) \cap AI(c) \neq \emptyset\}, Vc=MI(c)$

Instance	Attributs	Valeur
LCOM3	Nom	Lack of COhesion in Methods 3
	Formule	$Ec(c)=\{[m1,m2]: m1,m2 \in Vc(c) and\, AR(m1) \cap AR(m2) \cap AI(c) \neq \emptyset \vee m1 \in SIM(m2) \vee m2 \in SIM(m1)\}, Vc=MI(c)$

Instance	Attributs	Valeur
NRA	Nom	Number of Reference Attributes
	Formule	-

Instance	Attributs	Valeur
CIS	Nom	Class Interface Size
	Formule	$CIS(c) = \mid MPUB(c) \mid$

Instance	Attributs	Valeur
FCMEC	Nom	Friends Class-Method Expoprt Coupling
	Formule	$FCMEC(c) = \sum_{d \in Friends(c)} CM(d,c)$

Instance	Attributs	Valeur
IFCMIC	Nom	Inverse Friend Class-Method Import Coupling
	Formule	$IFCMIC(c) = \sum_{d \in Friends^{-1}(c)} CM(c,d)$

Instance	Attributs	Valeur
RFCAlpha	Nom	Response For Class Alpha
	Formule	$RFC\alpha(c) = \mid UR_{i=0 \ldots \alpha}\ i(c) \mid,\ \alpha = 1,2,3,\ldots$
		$R0(c) = M(c),\ Ri + 1(c) = U_{m \in Ri(c)}\ PIM(m),$
		PIM : set of polymorphistically invoked methods

Instance	Attributs	Valeur
Coh	Nom	Variation on lack of cohesion in methods 5
	Formule	$Coh(c) = \frac{\sum_{j=1 \ldots a} \mu(Aj)}{m \cdot a}$

Instance	Attributs	Valeur
LCC	Nom	Lose Class Cohesion
	Formule	$LCC(c) = \frac{|\{[m1,m2] : m1,m2 \in MI(c) \cap MPub(c) \text{ and } m1 \neq m2 \text{ and } cau(m1,m2)\}|}{|MI(c) \cap Mpub(c)|(|MI(c) \cap Mpub(c)| - 1)}$
		$cau(m1,m2) = (\cup AR(m)) \cap (\cup AR(m)) \cap (\cup AI(c) \neq 0$ $m \in \{m1\} \cup SIM(m1)\, m \in \{m2\} \cup SIM(m2)$

Instance	Attributs	Valeur
NPM	Nom	Number of Parameters per Pethod
	Formule	$NPM(c) = \frac{\sum_{m \in M(c)} Par(m)}{|M(c)|}$

Instance	Attributs	Valeur
AID	Nom	Average Inheritance Depth
	Formule	$AID(c) = \begin{cases} 0, \text{ if Parents(c)=0} \\ \sum_{c' \in Parents(c)} \frac{1 + AID(c')}{\lvert Parents(c) \rvert} \end{cases}$

Instance	Attributs	Valeur
CLD	Nom	Class-to-Leaf Depth
	Formule	$CLD(c) = \begin{cases} 0, \text{ if Descendents(c)=0} \\ max\{DIT(c') - DIT(c), c' \in Descendents(c)\} \end{cases}$

Instance	Attributs	Valeur
DIT	Nom	Depth of Inheritance Tree
	Formule	$DIT(c) = \begin{cases} 0, \text{ if Parents(c)=0} \\ 1 + max\{DIT(c'), c' \in Parents(c)\} \end{cases}$

Instance	Attributs	Valeur
SIX	Nom	Specialization IndeX
	Formule	$SIX(c) = \frac{NMO(c) \cdot DIT(c)}{\lvert M(c) \rvert}$

Instance	Attributs	Valeur
LCOM4	Nom	Lack of COhesion in Methods 4
	Formule	$LCOM4(c) = \begin{cases} 0, \text{ if\|P\|<\|Q\|} \\ \lvert P \rvert - \lvert Q \rvert, \text{ P=condition of LCOM1(c)} \end{cases}$ $Q = \{[m1, m2] : m1, m2 \in Vc(c) and AR(m1) \cap AR(m2) \cap AI(c) \neq \emptyset$

Instance	Attributs	Valeur
LCOM5	Nom	Lack of COhesion in Methods 5
	Formule	$LCOM5 = (\frac{1}{a}) \frac{(\sum_{j=1...a} \mu(Aj)) - m}{1 - m}$, μ(Aj) is the number of methods which reference

Instance	**Attributs**	**Valeur**
TCC	Nom	Tight Class Cohesion
	Formule	$TCC(c) = \frac{\lvert\{[m1,m2] : m1,m2 \in MI(c) \cap MPub(c) \text{ and } m1 \neq m2 \text{ and } cau(m1,m2)\}\rvert}{\lvert MI(c) \cap MPub(c)\rvert\,(\lvert MI(c) \cap Mpub(c)\rvert - 1}$

Instance	**Attributs**	Valeur
CBO'	Nom	Coupling Between Object classes
	Formule	$CBO'(c) = \lvert\{d \in C - (\{c\} \cup Ancestors(C)) \mid uses(c,d) \vee uses(d,c)\}\rvert$

CHAPITRE IV
IMPLÉMENTATION, INTERROGATION ET ÉVALUATION DE SWQAO

4.1 Implémentation

METHONTOLOGY ne fait aucune recommandation particulière du langage de formalisme à utiliser. De plus, cette méthodologie précise que lorsqu'on utilise un environnement de développement d'ontologie (ODE), celui-ci génère automatiquement le code de l'ontologie dans le formalisme qu'il supporte. Pour notre implémentation, nous avons choisi l'outil *Protégé* à cause de sa popularité dans le développement d'ontologies, sa gratuité, sa facilité d'utilisation et sa communauté présente et active sur internet. Le formalisme supporté par *Protégé* est la logique de description.

Au cours de l'utilisation de *Protégé*, nous avons fait face au problème de versions. En effet, les différentes versions de Protégé n'ont pas une compatibilité ascendante. Nous avons commencé l'implémentation de notre ontologie avec la version 4.1 qui était la plus récente à ce moment. Lorsque nous avons voulu interroger localement notre ontologie, on s'est rendu compte que cette version ne supporte pas SPARQL et qu'il n'existait pas de composant externe pouvant être intégré à cette version. Nous avons donc dû utiliser la version 4.2, sortie entre-temps, qui supporte SPARQL. Au cours de notre développement, on s'est aussi heurté au problème de comment définir le type de la partie condition du concept règles. En effet, ce type doit permettre d'accepter uniquement comme condition des données représentant une instance du concept métrique comparée à une valeur réelle ou une combinaison de ce type de

comparaisons. L'outil Protégé ne proposant pas la possibilité de créer directement un type issu de la combinaison d'un concept et d'un type prédéfini (exemple : literal, string, double, etc.), nous avons essayé les différentes possibilités qui s'offraient à nous, à savoir :

- Les expressions régulières : la limite des expressions régulières par rapport à notre problème est que celles-ci sont adaptées pour les chaînes de caractères. En effet, si on utilise les expressions régulières pour définir notre type on pourrait avoir des chaînes de caractères qui ne sont pas des métriques. De plus, *Protégé* 4.2 ne supporte pas les expressions régulières, mais la version 4.1 si, et seul le raisonneur HermiT les reconnait. Donc, même si les expressions régulières étaient la solution à notre problème, on aurait toujours été contraint à trouver une autre solution, car *Protégé* 4.2 nous permet de construire une ontologie interrogeable et donc utilisable par d'autres systèmes. Par contre, on ne peut pas utiliser les expressions régulières dans nos requêtes.

- La logique de description : on a créé un concept *ComparisonSign* qui a pour instances les signes de comparaison, un concept *SimpleCondition* qui, à l'aide de relations, regroupe le concept *métrique*, les signes de comparaison et les seuils pour les comparaisons et le concept *CombinedCondition* qui permet d'avoir plusieurs conditions simples, ces conditions simples étant reliées implicitement entre elles par des conjonctions. Si on a des disjonctions entre nos conditions simples, cela veut dire en fait que la règle en question regroupe plusieurs règles. Il faudra donc séparer cette règle en plusieurs règles simples ne comportant que des conjonctions dans leur partie condition.

  Considérons la règle suivante : R1 : MPC $\leq$ 1 OMMIC $\leq$ 4 AMMIC $\in$]0, 3] impact : very − weak(52.0/3.0)

  On peut soit :

- Définir very-weak comme une instance du concept Impact, ainsi notre formule sera écrite en *Protégé* tel que suit : R1 : MPC ≤ 1 OMMIC ≤ 4 AMMIC > 0 AMMIC ≤ 3 very − weak, avec « 52.0/3.0 » comme la valeur de la propriété poids de la règle. Le problème avec ce choix est que lorsqu'on voudra ajouter une règle dont la partie conséquence porte sur un autre attribut interne de qualité tel que l'effort de correction, la propension aux fautes, etc. on ne pourra pas créer une instance « very-weak » de type Effort dans le cas où very-weak fera parti des degrés d'effort. Tout ceci parce qu'on ne peut créer des individus syntaxiquement identiques même si leurs propriétés sont différentes.

- Définir very-weak comme une instance de la classe State (État). Cela nous donne la formule suivante : MPC ≤ I OMMIC ≤ 4 AMMIC > 0 AMMIC ≤ 3 ImpactR1 : very − weak, avec « 52.0/3.0 » comme la valeur de la propriété poids de la règle et ImpactR1 étant une instance de la classe Impact. L'avantage de ce choix est qu'il est facile d'ajouter d'autres attributs internes ayant la même échelle de degré. Ainsi, on peut avoir les attributs impact et effort de correction qui peuvent être classés selon l'échelle très-faible, faible, ..., très-fort, pour traduire le degré d'impact d'une modification et l'effort nécessaire pour effectuer une correction.

4.2 Interrogation et diffusion de SwQAO

Il existe plusieurs langages de requêtes qui ont été proposés pour les fichiers RDF et RDFS : RDQL, SeRQL, SPARQL, etc. De tous ces langages, SPARQL est celui

recommandé par W3C et par conséquent le plus utilisé. Pour l'interrogation de notre ontologie sur internet, nous avons le choix entre les répertoires d'ontologies publiques et le système Sesame[22].

4.2.1 Le langage de requêtes SPARQL

SPARQL Protocol And RDF Query Language est un langage de requêtes qui sert à accéder et manipuler les données au format RDF(S) contenues dans des bases de données. Ce langage est basé sur l'appariement des patrons de graphes des requêtes avec les graphes RDF. Les patrons de graphes sont construits à partir de patrons triplets. Un patron triplet est comme un triplet RDF (sujet-prédicat-objet), avec pour différence que le sujet, le prédicat ou l'objet peut-être remplacé par une variable. Une variable commence toujours par un point d'interrogation (?).

Exemple de patron triplet :

<http :/example.org/book/book1> <http ://purl.org/dc/elements/1.1/title> ?title.

Dans cet exemple, on a comme :

Sujet : <http :/example.org/book/book1>

Prédicat : <http ://purl.org/dc/elements/1.1/title>

Objet : ?title qui est une variable.

On distingue deux grands types de patrons de graphes SPARQL :

1. Patrons de graphes de groupe

[22] http ://www.openrdf.org/

Ces patrons sont les plus utilisés dans les requêtes SPARQL. Ils sont construits à partir :

a. Des patrons de graphes basiques. Un (patron de) graphe basique est une séquence de (patrons) triplets séparés par des points. Chaque point représente une conjonction. Les triplets peuvent être délimités par des accolages. {} est un graphe de groupe vide.

   Exemple :

   ?x foaf :name ?name . ?x foaf :mbox ?mbox , {?x foaf :name ?name . ?x foaf :mbox ?mbox}, {{ ?x foaf :name ?name .} { ?x foaf :mbox ?mbox .}}

   Ces trois graphes de groupe sont équivalents.

b. Des filtres (« FILTER »).

   Exemple :

   Prefix dc : <http ://purl.org/dc/elements/1.1/>

   Prefix ns : <http ://example.org/ns#>

   SELECT ?title ?price WHERE { ?x ns :price ?price . FILTER (?price < 30.5) ?x dc :title ?title .}

c. Des patrons de graphes optionnels (« OPTIONAL »).

   Exemple :

   Prefix foaf : <http ://xmlns.com/foaf/0.1>

SELECT ?name ?mbox ?hpage WHERE { ?x foaf :name ?name . OPTIONAL { ?x foaf :mbox ?mbox .} OPTIONAL { ?x foaf :homepage ?hpage .}}

d. Des patrons de graphes alternatifs (« UNION »).

Exemple :

Prefix dc10 : <http ://purl.org/dc/elements/1.0/>

Prefix dc11 : <http ://purl.org/dc/elements/1.1/>

SELECT ?title WHERE { { ?book dc10 :title ?title } UNION { ?book dc11 :title ?title }}

2. Patrons de graphes nommés.

SPARQL a des mots réservés (select, where, order by, from, etc.) semblables à ceux de SQL, mais malgré cela SPARQL ne fonctionne pas comme SQL. Le groupe de travail SPARQL [47] donne plus de détails sur le contenu de cette section.

4.2.2 Diffusion et cas d'utilisation de l'ontologie

Pour rendre une ontologie publique, en plus de lui donner une URL, on peut :

- L'ajouter à un ou plusieurs répertoires d'ontologies publiques disponibles sur le web.
- La publier dans une version facile à interpréter par la machine (RDF/OWL, etc.) ou dans le format html plus facile à lire et à interpréter pour l'humain.

- La rendre explorable dans un éditeur d'ontologies en ligne.

Les répertoires d'ontologies aussi appelés librairies d'ontologies sont des systèmes qui collectent les ontologies de différentes sources afin de faciliter leur recherche, leur exploration et leur utilisation. Mathieu d'Aquin et al. [28] font une revue de ces répertoires qui se différencient par leurs tailles, leurs objectifs, leurs fonctionnalités, les formats qu'ils supportent ou la façon d'accéder à leur contenu. Après élimination des répertoires réservés aux domaines différents du nôtre, notre choix se limitait aux répertoires TONES[23], Cupboard[24] et OntoSearch2[25]. L'inconvénient principal du répertoire TONES par rapport à nos besoins est son incapacité à pouvoir effectuer des recherches et des requêtes à l'intérieur d'une ontologie. Le répertoire Cupboard ne supporte pas la recherche d'ontologies par mots-clés. Il faut connaitre l'URL de l'ontologie qu'on cherche. Pour ajouter son ontologie dans Cupboard il faut devenir membre. Pour devenir membre, il faut faire une demande et les critères de sélection sont inconnus.

Le répertoire Ontosearch2 s'est avéré être le mieux adapté à nos besoins, car il ne collecte pas que les ontologies propres à un seul domaine. Il supporte aussi la recherche d'ontologies par mots-clés et il est possible d'effectuer des requêtes SPARQL sur les ontologies qu'il contient. Cependant, Ontosearch2 n'accepte que les ontologies rédigées en OWL-Lite. L'expressivité faible d'OWL-Lite par rapport à OWL-DL ne nous permet pas de créer une ontologie conforme à nos connaissances. Nous avons donc utilisé OWL-DL pour formaliser nos connaissances. Pour cette

[23] http://owl.cs.manchester.ac.uk/repository/

[24] http://cupboard.open.ac.uk :8081/cupboard

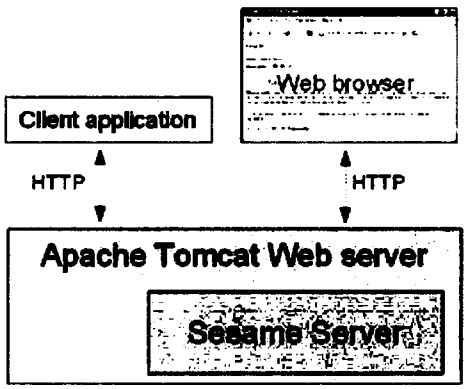
[25] http://dipper.csd.abdn.ac.uk/OntoSearch/

raison, nous n'avons pas utilisé le répertoire Ontosearch2 pour permettre l'interrogation publique de notre ontologie.

Nous avons utilisé l'outil Sesame comme serveur (voir figure 4.1) pour offrir la possibilité de formuler des requêtes à notre ontologie en étant connecté sur internet. Nous allons utiliser les informations qui se trouvent sur le site officiel de Sesame[26] pour le présenter brièvement.

Sesame est un cadre de travail Java gratuit utilisé pour la sauvegarde, l'interrogation et le raisonnement avec RDF et RDFS. Il peut être utilisé comme base de données pour les données en format RDF(S) ou comme une librairie Java pour les applications qui ont besoin de travailler avec RDF à l'interne. Sesame peut être aussi utilisé comme un serveur avec lequel l'application client ou l'utilisateur peut communiquer grâce au protocole HTTP.

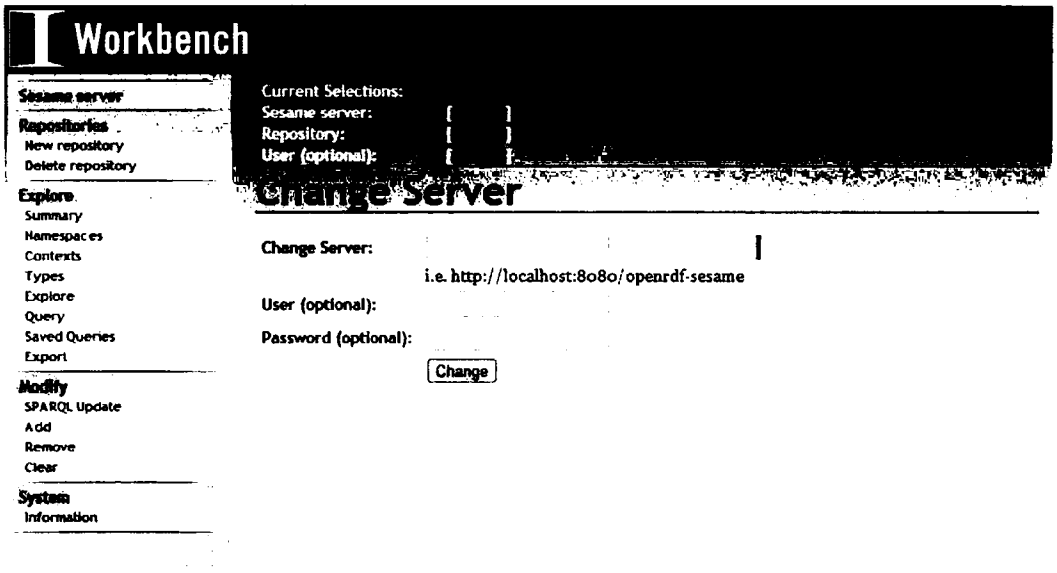
Figure 4.1 Sesame comme serveur.



[26] http://www.openrdf.org/doc/sesame/users/

Comme le montre la figure 4.2, la page principale de l'espace de travail que nous offre le serveur Sesame nous permet de saisir l'adresse du serveur, notre identifiant et mot de passe si nécessaire. Cet espace de travail liste à sa droite toutes les opérations qu'on peut effectuer. On peut ajouter ou supprimer des répertoires qui abritent plusieurs ontologies. On peut explorer un répertoire choisi : en affichant son résumé ou les URL des préfixes utilisés par les ontologies contenues dans le répertoire, en affichant ses contextes et ses types, en formulant des requêtes, en exportant son contenu, etc. On peut aussi modifier le contenu d'un répertoire.

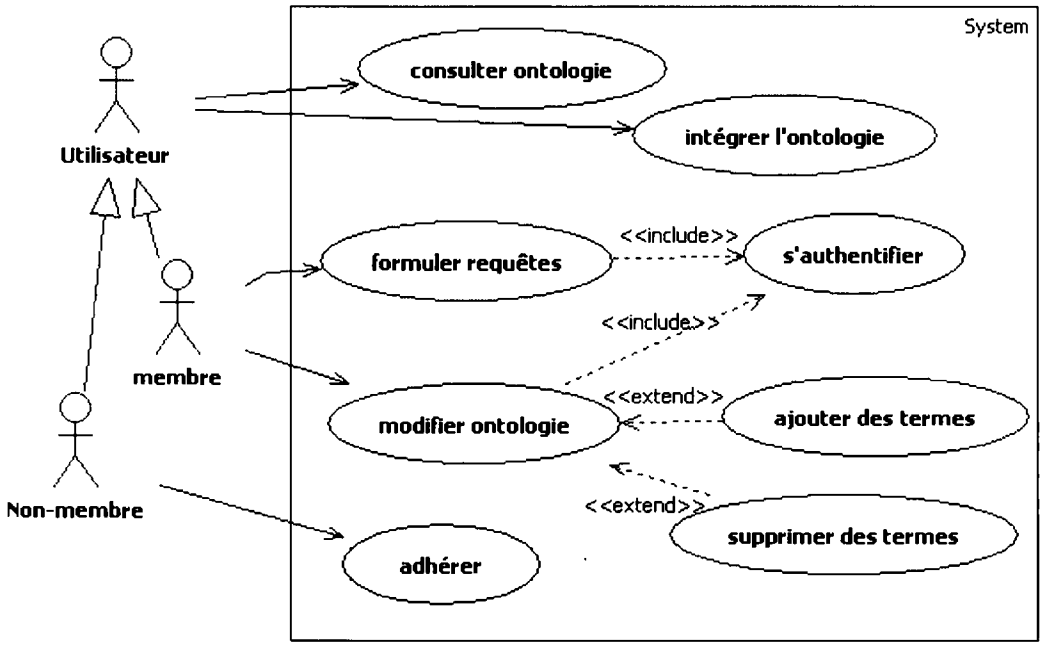
Figure 4.2 Page principale du serveur Sesame.



Cas d'utilisation

La figure 4.3 suivante présente les différents cas d'utilisation de notre ontologie :

Figure 4.3 Diagramme de cas d'utilisation de SwQAO.



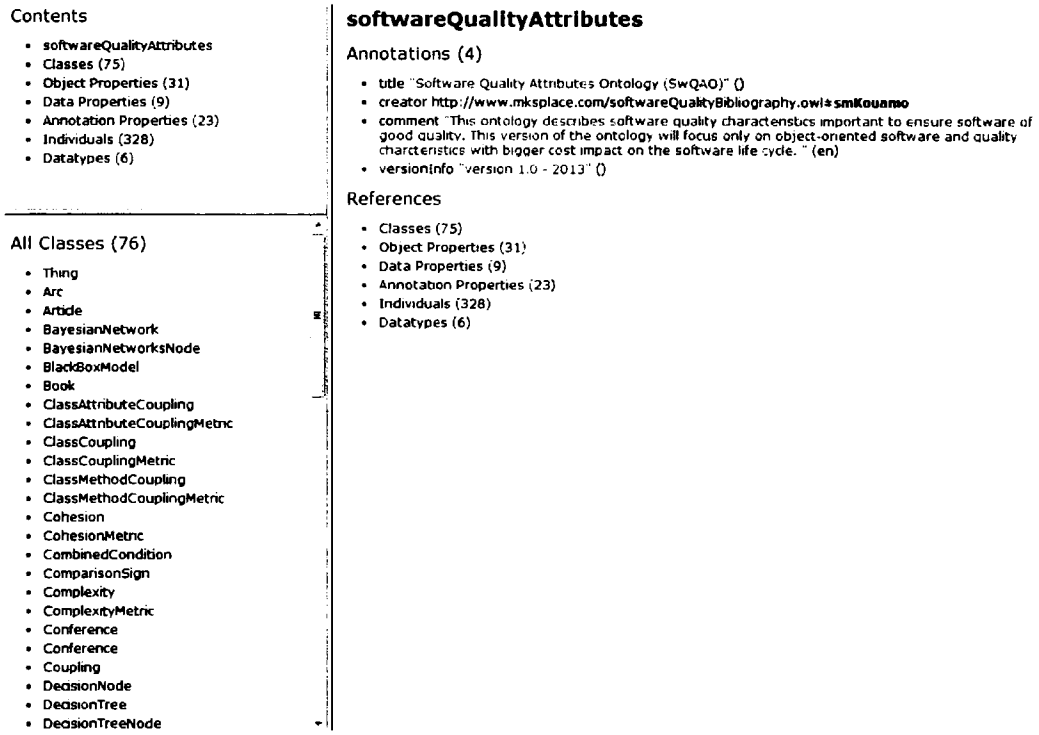
Cas d'utilisation : consulter l'ontologie

Acteur : utilisateur

Description : un utilisateur quelconque peut se familiariser avec l'ontologie en la consultant directement sur internet. Il n'a pas besoin de télécharger et d'ouvrir l'ontologie dans son éditeur local pour se familiariser avec celle-ci. Étant donné qu'il est plus facile de se familiariser avec les concepts, relations et propriétés d'une ontologie lorsque celle-ci est présentée dans un format qui est facile à lire et qui ne demande aucune connaissance technique pour comprendre son contenu, notre ontologie peut-être consultée dans le format HTML. Ce format facilite sa

compréhension, et de ce fait son utilisation. Voici quelques extraits de la version HTML de notre ontologie :

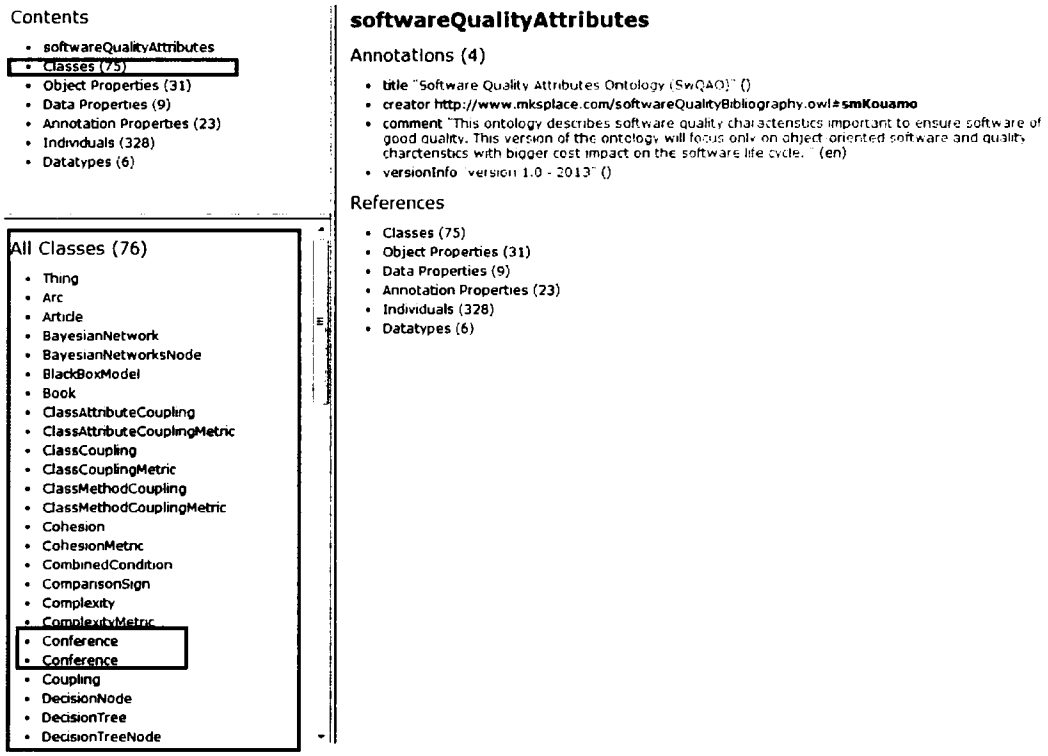
Figure 4.4 Version HTML de notre ontologie.



Cette figure présente notre ontologie. La partie supérieure gauche (*Contents*) présente l'ontologie (*software Quality Attributes*) et les différents groupes qui constituent l'ontologie, à savoir *Classes*, *Object Properties*, *Individuals*, etc, avec entre parenthèses le nombre total d'éléments de chaque groupe. Juste en dessous de cette section, nous avons la liste de tous les termes de l'ontologie et leur nombre total entre parenthèses. Ces termes sont les éléments des groupes de la partie *Contents*. À droite, on a les informations comme le titre, l'auteur, la

version et la description de l'ontologie. En choisissant un groupe dans la partie *Contents*, on a la liste de ses éléments dans la partie juste en dessous de *Contents*, comme le montre la figure suivante lorsqu'on sélectionne le groupe *Classes*.

Figure 4.5 Classes de l'ontologie SwQAO (version HTML).

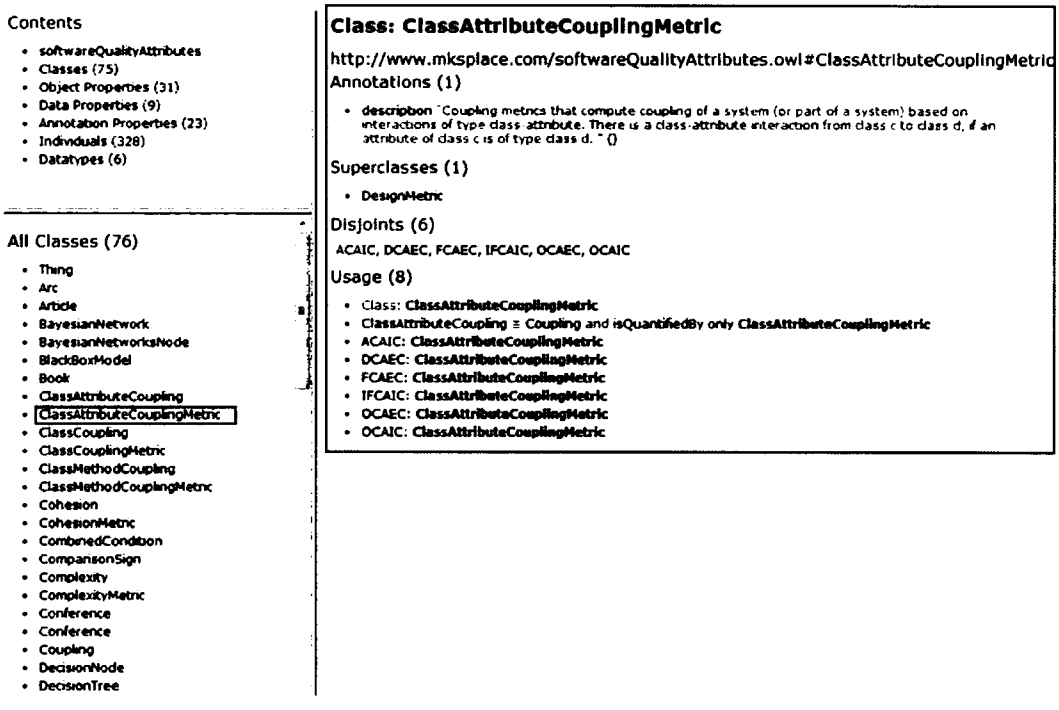


Dans la partie *Contents*, on a 75 comme nombre total de classes et la liste *All Classes* a 76 classes. Cela s'explique par le fait que la liste prend en considération les classes *Thing* et *Nothing* qui représentent respectivement la classe la plus générale et la classe la plus spécifique dans toute ontologie. Dans la partie *Contents*, seule la classe *Thing* est prise en considération car dans un éditeur d'ontologie la taxonomie des classes a toujours comme racine la Classe *Thing*, et la classe

Nothing ne figure nulle part dans cette taxonomie. On a aussi des doublons comme c'est le cas avec la classe *Conference*, parce que celle-ci a été définie dans notre ontologie et elle est équivalente à celle de l'ontologie bibliographique BIBO du groupe PURL.

Il suffit de sélectionner une classe de la liste et on a les détails de cette classe dans la partie droite de la page web. Les détails sont : la définition informelle et formelle, s'il y a lieu, de la classe ; son parent ; les termes de l'ontologie avec lesquels elle est disjointe (si cela a été explicitement spécifié) ; tous les endroits où le terme sélectionné est utilisé dans l'ontologie, etc. La figure suivante nous illustre l'exemple d'une classe de notre ontologie.

Figure 4.6 Détails de la classe *ClassAttributeCouplingMetric*.



Si on veut donc avoir plus d'informations sur un terme de l'ontologie, il suffit de le sélectionner. Par exemple, si on veut connaitre la formule de la métrique *ACAIC*, sa provenance, son ou ses auteurs, il suffit de la sélectionner et on obtient une fenêtre comme celle de la figure 4.7. Pareil, si pour une personne donnée on veut connaitre de quels documents ou métriques il est l'auteur, il suffit de sélectionner la classe Person, dans la partie *Usage* à droite de la page, et choisir la personne pour laquelle on veut plus de détails, comme nous le montrent les figures 4.8 et 4.9.

Figure 4.7 Détails d'une instance de la classe *Metric*.

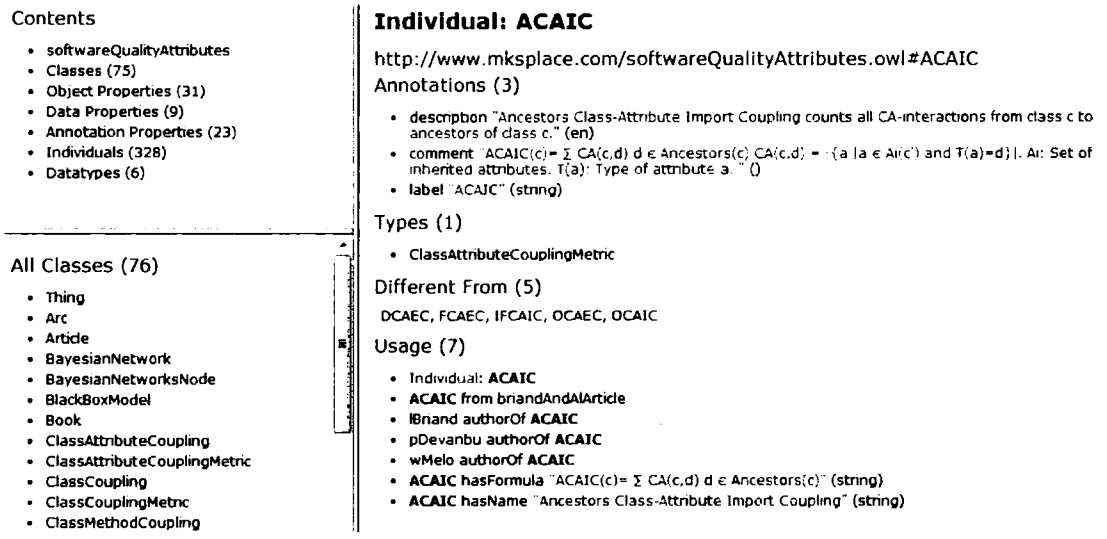


Figure 4.8 Détails de la classe *Person*.

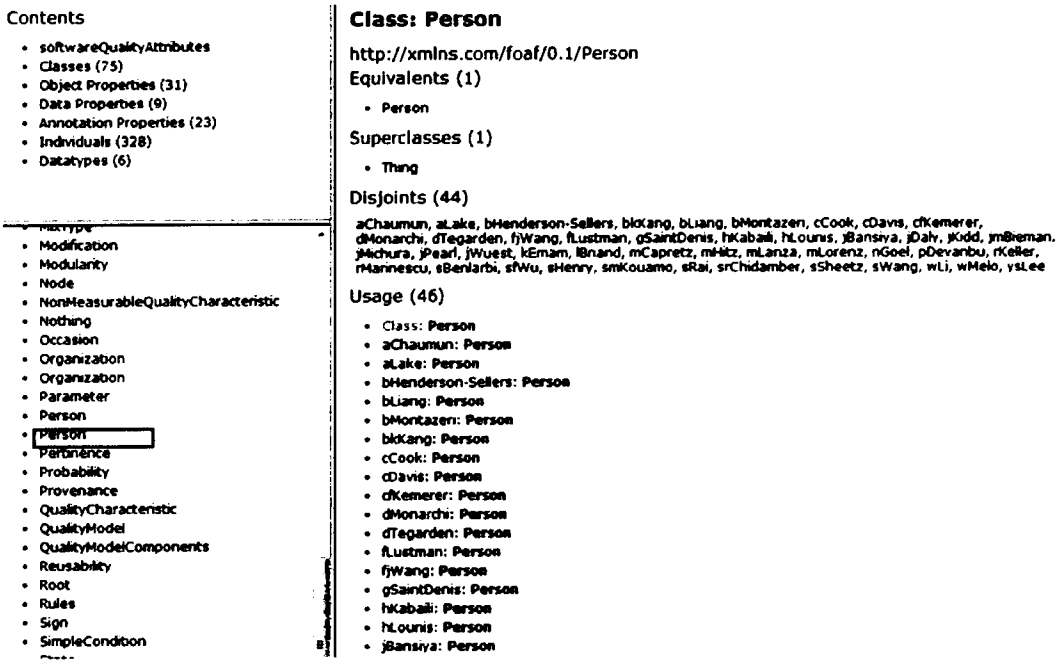
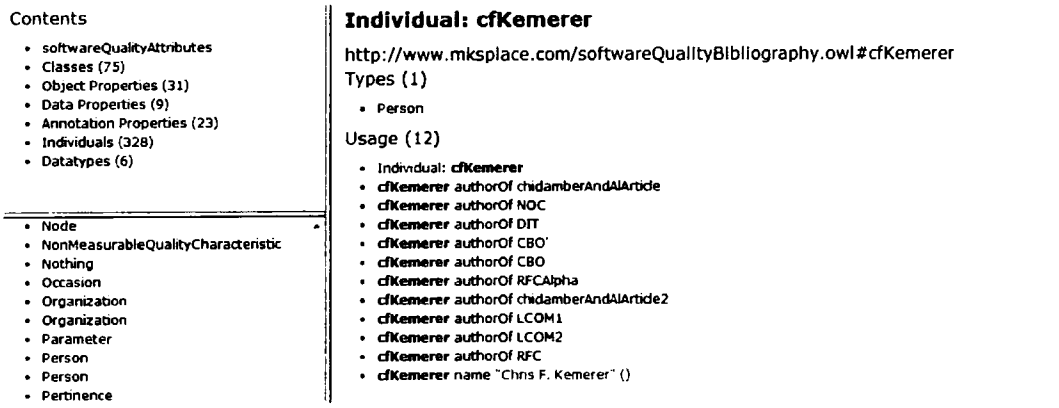
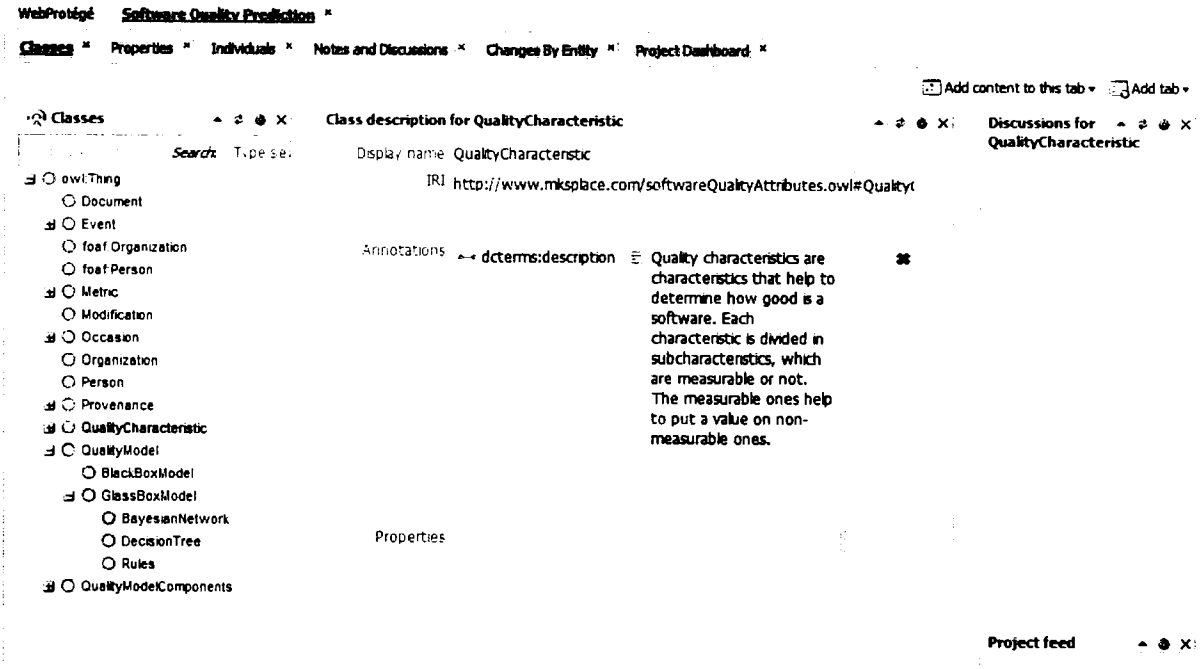


Figure 4.9 Détails d'une instance de la classe *Person*.



On peut aussi explorer notre ontologie sur internet dans un éditeur d'ontologies. Ceci a pour avantages de découvrir l'ontologie dans son cadre de développement et de faciliter la navigation à travers l'ontologie. Voici à quoi ressemble notre ontologie dans l'éditeur en ligne *WebProtégé*. Étant donné que nous avons utilisé *Protégé* comme éditeur dans le cadre de notre travail, nous avons utilisé *WebProtégé* comme éditeur en ligne pour notre ontologie.

Figure 4.10 L'ontologie SwQAO dans l'éditeur web *WebProtégé*.



Cas d'utilisation : modifier l'ontologie

Acteur primaire : utilisateur membre. **Acteur secondaire** : système de vérification.

Description : seul un utilisateur membre peut modifier l'ontologie. L'utilisateur membre doit d'abord s'authentifier en saisissant son nom d'utilisateur et son mot de passe. Si la vérification de ses informations par le système de vérification est un succès, l'utilisateur peut commencer les modifications. Par contre, si la vérification de ses informations est un échec, il doit recommencer l'authentification. Si l'utilisateur ne réussit toujours pas à s'authentifier après plusieurs tentatives, celui-ci doit recommencer le processus d'adhésion pour redevenir membre. Comme modifications, un utilisateur membre peut aussi ajouter ou supprimer des termes de l'ontologie.

Cas d'utilisation : formuler des requêtes

Acteur : utilisateur membre

Description : Il est possible d'interroger notre ontologie en formulant des requêtes SPARQL via Sesame. Pour cela, l'utilisateur doit être membre et s'authentifier au préalable avec succès. Les figures 4.11 et 4.12 nous montrent à quoi ressemble, respectivement, une requête SPARQL et une réponse à une requête dans l'interface de Sesame.

Figure 4.11 Requête SPARQL avec Sesame.

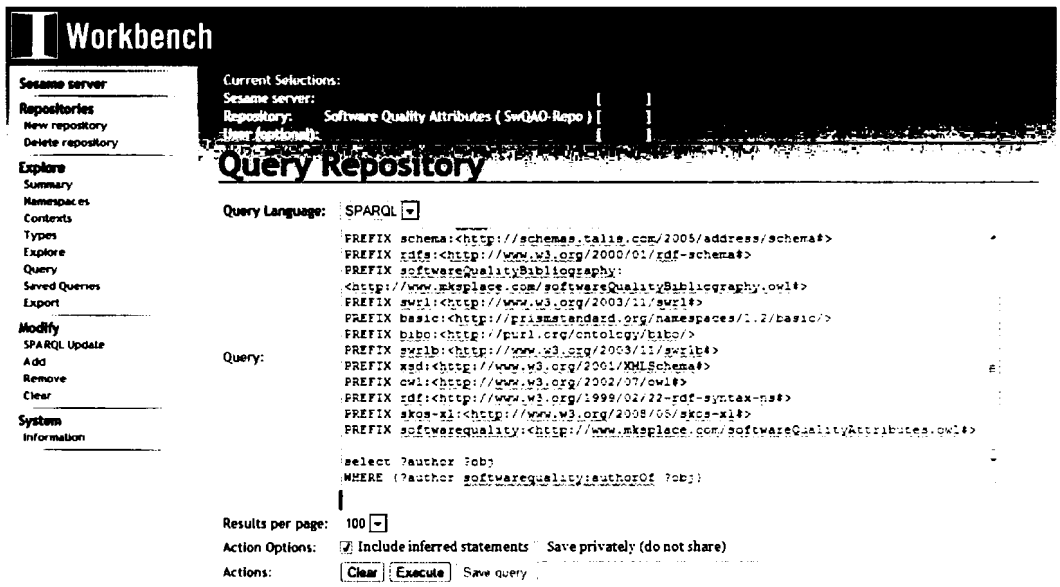


Figure 4.12 Réponse à une requête dans Sesame.

Query Result (1-100 of 214)

Download format: [ ] Download

Results per page: 100

Results offset: Previous 100 Next 100

Show data types & language tags: ☑

Author	Obj
softwareQualityBibliography:aChaumun	softwarequality:CBOIUB
softwareQualityBibliography:aChaumun	softwarequality:CBONA
softwareQualityBibliography:aChaumun	softwarequality:CBOU
softwareQualityBibliography:aChaumun	softwareQualityBibliography:chaumunAndAlArticle
softwareQualityBibliography:aLake	softwarequality:NOP
softwareQualityBibliography:aLake	softwareQualityBibliography:lakeAndCookArticle
softwareQualityBibliography:bHenderson-Sellers	softwarequality:AID
softwareQualityBibliography:bHenderson-Sellers	softwarequality:NMA
softwareQualityBibliography:bHenderson-Sellers	softwarequality:NMO
softwareQualityBibliography:bHenderson-Sellers	softwarequality:NPM
softwareQualityBibliography:bHenderson-Sellers	softwarequality:SIX
softwareQualityBibliography:bHenderson-Sellers	softwareQualityBibliography:henderson-SellersBook
softwareQualityBibliography:bLiang	softwarequality:ICH
softwareQualityBibliography:bLiang	softwarequality:ICP
softwareQualityBibliography:bLiang	softwarequality:IH-ICP
softwareQualityBibliography:bLiang	softwarequality:NIH-ICP
softwareQualityBibliography:bLiang	softwareQualityBibliography:liangAndAlArticle
softwareQualityBibliography:bMontazeri	softwarequality:Co

Cas d'utilisation : intégrer l'ontologie dans une application

Acteur : utilisateur

Description : l'utilisateur peut intégrer l'ontologie dans un système (système d'aide à la décision, système d'aide à l'apprentissage, etc.).

Cas d'utilisation : adhérer

Acteur : utilisateur non-membre

Description : si un utilisateur non-membre veut modifier l'ontologie, il doit remplir le formulaire de demande de membre. Ce statut de membre lui donnera le droit d'écriture sur l'ontologie. En remplissant le formulaire de demande de membre, il doit clairement spécifier la raison pour laquelle il veut ce statut. Si la demande est approuvée, les informations pour s'authentifier lui seront retournées par courriel à l'adresse qu'il aura fourni dans le formulaire de demande.

4.3 Évaluation de l'ontologie SwQAO

On peut évaluer une ontologie sur différentes bases, à savoir [40] :

- La base terminologique : sur cette base on évalue l'ontologie en la vérifiant et en la validant. Vérifier le contenu d'une ontologie consiste à analyser sa complétude, sa consistance, sa concision, sa sensibilité et sa capacité d'évolution. Valider une ontologie consiste à montrer que l'ontologie correspond au domaine qu'elle est censée représenter. En plus de cela, on peut aussi démontrer l'utilité et la facilité d'utilisation de l'ontologie.

- La base des critères : ici on doit vérifier si les concepts, les relations et les propriétés de l'ontologie respectent les critères de définition, si la hiérarchie des concepts est correcte.

- La base des méthodes : la plus populaire est la méthode qui consiste à évaluer une ontologie par rapport à ses questions de compétence. En utilisant cette méthode, on s'assure que l'ontologie est en mesure de fournir des réponses correctes à toutes les questions de compétence.

- La base des éditeurs d'ontologie : on peut évaluer une ontologie par rapport à un éditeur d'ontologie. En choisissant Ontolingua comme éditeur, on peut utiliser son parseur pour déterminer si les définitions sont bien formulées et générer un rapport qui contient les concepts non définis et les dépendances entre les termes de l'ontologie.

Nous allons allons utiliser la base terminologique, car elle est bien détaillée et plus complète. Pour la validation de notre ontologie, nous allons utiliser les questions de compétence définies dans la section 3.1. Nous allons donc combiner l'évaluation sur la base terminologique et la base de méthodes.

4.3.1 Consistance

La vérification de la consistance se fait en plusieurs étapes. On doit garantir, dans la mesure du possible, la consistance interne et métaphysique des définitions. La consistance interne des définitions demande de s'assurer que les définitions informelles et formelles puissent être interprétées de la même façon, l'une ne doit pas dire moins ou plus que l'autre. Cette consistance est facile à vérifier lorsque les termes à définir sont des énumérations. La consistance métaphysique de la définition informelle et formelle garantit que la définition informelle et formelle d'un terme est en respect avec le monde réel. Voici un exemple pour illustrer comment nous avons vérifié la consistance des termes de notre ontologie :

Définition informelle de la métrique ACAIC : La métrique Ancestors Class-Attribute Import Coupling compte toutes les relations classe-attribut de la classe c avec ses ancêtres. Il existe une relation classe-attribut entre les classes c et d, si un attribut de la classe c a pour type la classe d.

Définition formelle ou formule de la métrique ACAIC :

$$ACAIC(c) = \sum_{d \in Ancestors(c)} CA(c, d),\ CA(c, d) = |\{a | a \in Ai(c') \quad and \quad T(a) = d\}|,$$

Ai : ensemble des attributs hérités, T(a) : Type de l'attribut a.

On voit bien que les définitions informelle et formelle du terme ACAIC ont la même interprétation, ce qui montre la consistance interne de ce terme. La définition informelle respecte le monde réel, car cette définition est fidèle à ce que calcule ou représente la métrique ACAIC dans le monde réel. La consistance métaphysique de la définition informelle du terme ACAIC est vérifiée. Pour ce qui est de la formule d'ACAIC qui représente aussi sa définition formelle, elle n'est fidèle au monde réel que lorsqu'on la complète avec la représentation formelle du terme CA utilisé dans la formule. La ligne ci-dessus qui donne la formule de CA complète la formule et est ajoutée dans la partie commentaire de notre ontologie pour ne pas alourdir la propriété hasFormula de la métrique.

4.3.2 Complétude

Gomez-Perez [40] suggère trois étapes pour vérifier la complétude d'une ontologie :

- Vérifier la complétude de la hiérarchie des classes pour chaque classe définie.

  Cette étape consiste à vérifier si la hiérarchie des classes est fidèle au monde réel avant et après inférence. Pour cela, on s'assure que les classes disjointes en réalité soient clairement identifiées comme disjointes dans l'ontologie, que les classes dites superclasses/sous-classes soient effectivement des superclasses/sous-classes dans la réalité.

  La hiérarchie suivante est bien fidèle au monde réel lorsqu'on se fie à la

littérature.

QualityModel

BlackModel

GlassModel

BayesianNetwork

DecisionTree

Rules

On fait la même vérification pour la partie restante de la taxonomie de notre ontologie.

- Vérifier la complétude du domaine et co-domaine de chaque relation et s'assurer que tous les domaines sont bien définis dans la hiérarchie des classes. On doit s'assurer que les domaines et co-domaines soient les mêmes que ceux définis dans le monde réel. La figure suivante illustre les domaines et co-domaines des relations de notre ontologie.

Figure 4.13 Relations de l'ontologie avec leurs domaines et co-domaines respectifs.

domaine	relation	co_domaine
DecisionTree	hasArc	Arc
BayesianNetworksNode	hasDefinitionalRelationWith	BayesianNetworksNode
Structure	hasBNNode	BayesianNetworksNode
Parameter	isProbabilityTableOf	BayesianNetworksNode
BayesianNetworksNode	hasCausalRelationWith	BayesianNetworksNode
SimpleCondition	aComparisonSign	ComparisonSign
DecisionTree	hasDTNode	DecisionTreeNode
Modification	hasImpact	Impact
Metric	quantifies	MeasurableQualityCharacteristic
NonMeasurableQualityCharacteristic	hasElement	MeasurableQualityCharacteristic
Node	representsMetric	Metric
EntryNode	hasInput	Metric
SimpleCondition	hasMetric	Metric
MeasurableQualityCharacteristic	isQuantifiedBy	Metric
CombinedCondition or Rules	hasCondition	MixType
Node	hasParent	Node
Node	isParentOf	Node
MeasurableQualityCharacteristic	isElementOf	NonMeasurableQualityCharacteristic
QualityCharacteristic	hasSubcharacteristic	NonMeasurableQualityCharacteristic
Rules	hasConsequence	NonMeasurableQualityCharacteristic
BayesianNetwork	hasParameters	Parameter
Parameter	hasComputationalInfluenceOn	Parameter
BayesianNetworksNode	hasProbabilityTable	Parameter
NonMeasurableQualityCharacteristic	isSubcharacteristicOf	QualityCharacteristic
Node	representsQC	QualityCharacteristic
	hasState	State
BayesianNetwork	hasStructure	Structure
Provenance	presentedAt	Occasion
	from	Provenance

Lorsque le domaine ou le co-domaine est vide, cela est implicitement remplacé par THING qui est la superclasse de toutes les classes définies dans l'ontologie. C'est le cas pour la relation hasState dont le domaine est vide et le co-domaine est la classe State. Ceci veut dire que toute classe de l'ontologie peut avoir un état (*State*).

- Vérifier la complétude des classes.

Il faut s'assurer qu'aucune classe ne manque d'attributs ou a des attributs qui n'existent pas dans le monde réel. Il en est de même pour les relations de chaque classe avec d'autres classes. Pendant le déroulement de cette vérification, il faut garder en mé- moire que les concepts d'une ontologie sont décomposés jusqu'au niveau de granu- larité nécessaire pour la compréhension du domaine ou sous-domaine choisi. Prenons par exemple le cas d'un nœud d'entrée d'un

réseau bayésien et d'une métrique. Ce qui nous intéresse sur un nœud d'entrée est qu'il reçoit en entrée des métriques, il a une table de probabilité, il n'a aucun parent et il a un ou plusieurs enfants. Ceci est représenté comme suit dans notre ontologie :

EntryNode

- hasInput some Metric
- hasProbabilityTable some EntryVariableParameter
- isParentOf some BayesianNetworksNode
- hasParent max 0 Node

Une métrique a un nom, une formule et une valeur et aide à quantifier des attributs de qualité logicielle quantifiable.

Metric

- hasName some string
- hasFormula some string
- hasValue some double
- quantifies some MeasurableQualityCharacteristic

4.3.3 Concision et évolution

La concision d'une ontologie consiste à s'assurer que les informations contenues dans une ontologie sont précises et utiles. Elle est difficile à démontrer parce qu'il existe plusieurs définitions possibles pour chacun des termes d'une ontologie. Il faut

toutefois s'assurer que notre ontologie contient des définitions utiles, ceci en évitant les redondances dans les définitions, dans les superclasses et les sous-classes, bien que dans certains cas un certain degré de redondance est justifié.

Notre ontologie peut évoluer facilement. Qu'on veuille ajouter d'autres types de modèles de qualité, ajouter d'autres instances des modèles de qualité déjà considérés dans l'ontologie, ajouter un nouveau type de métriques, ajouter des définitions, supprimer/ajouter des propriétés ou relations, l'architecture de notre ontologie le permet sans avoir à re- structurer une grande partie de l'ontologie.

4.3.4 Validation

Comme mentionné ci-dessus, nous utilisons les questions de compétence pour valider notre ontologie. Les réponses à certaines questions sont déjà affichées dans la version HTML de notre ontologie. Cette version HTML (http://www.mksplace.com/owlDoc/) qui est obtenue grâce au plugin OWLDoc permet de se familiariser à l'ontologie tout en répondant à certaines questions. On y trouve les réponses aux questions telles que : Quelles sont les concepts de notre ontologie ? (confirme qu'il est bien question d'attributs et modèles de qualité), d'où proviennent nos métriques ? (Il suffit de cliquer sur la propriété objet *from*), Quels sont tous les endroits où un terme donné de l'ontologie est utilisé ? Quelles sont les conditions et conséquences d'une règle donnée ? etc.
Voici les requêtes SPARQL qui montrent que notre ontologie répond correctement aux questions auxquelles l'ontologie est censée répondre. Pour des raisons d'espace nous ne formulerons pas toutes les requêtes et leurs réponses. Pour visualiser les réponses aux requêtes suivantes et formuler vos propres requêtes, nous vous invitons à consulter le lien http://mksplace.com/form.html et faire une demande de membre.

PREFIX rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl : <http://www.w3.org/2002/07/owl#>

PREFIX xsd : <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs : <http://www.w3.org/2000/01/rdf-schema#>

PREFIX skos : <http://www.w3.org/2008/05/skos-xl#>

PREFIX softwarequality : http://www.mksplace.com/softwareQualityAttributes.owl#>

PREFIX terms : <http://purl.org/dc/terms/>

- Quelles sont les métriques de cohésion ?

  SELECT?cohesionMetric

  WHERE { ?cohesionMetric rdf:type ?type . FILTER (?type = softwarequality :CohesionMetric)}

  Résultat : TCC, LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Co, Coh, LCC, ICH

- Quelles sont les métriques de complexité ?

  SELECT ?complexityMetric

  WHERE{?complexityMetric rdf:type ?type . FILTER (?type = softwarequality :ComplexityMetric)}

  Résultat : CIS, CSB, NA, NAD, NOM, NOO, NOPM, NOT, NPA, NPM, NRA

- Quelles sont les métriques d'héritage ?

  SELECT ?inheritanceMetric

  WHERE {?inheritanceMetric rdf:type ?type . FILTER (?type = softwarequality :InheritanceMetric)}

  Résultat : AID, CLD, DIT, NMA, NMI, NMO, NOA, NOC, NOD, NOP, SIX

- Quelles sont les métriques de conception ?

  SELECT ?designMetric

  WHERE {?designMetric rdf:type ?type . ?type rdfs:subClassOf softwarequality:DesignMetric}

  Résultat : ACAIC, DCAEC, FCAEC, IFCAIC, OCAEC, OCAIC, ACMIC, DCMEC, IFCMIC, OCMEC, OCMIC, AMMIC, DMMEC, FMMEC, IFMMIC, OMMEC, OMMIC

- Quelles sont les métriques de couplage ?

  SELECT ?couplingMetric

  WHERE {?couplingMetric rdf:type ?type .
  FILTER (?type = softwarequality:ClassCouplingMetric ||
  ?type = softwarequality:ClassAttributeCouplingMetric ||
  ?type = softwarequality:ClassMethodCouplingMetric ||
  ?type = softwarequality:MethodMethodCouplingMetric)}

  Résultat : CBO, CBO', CBOIUB, CBONA, CBOU, DAC, DAC', ICP, IH-ICP, MPC, NIH-ICP, RFC, RFC', RFCAlpha, ACAIC, DCAEC, FCAEC,

IFCAIC, OCAEC, OCAIC, ACMIC, DCMEC, IFCMIC, OCMEC, OCMIC, AMMIC, DMMEC, FMMEC, IFMMIC, OMMEC, OMMIC

- Quelles sont les métriques d'implémentation ?

  SELECT ?implementationMetric

  WHERE {?implementationMetric rdf:type ?type . ?type rdfs:subClassOf softwarequality:ImplementationMetric}

  Résultat : CBO, CBO', CBOIUB, CBONA, CBOU, DAC, DAC', ICP, IH-ICP, MPC, NIH-ICP, RFC, RFC', RFCAlpha, TCC, LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Co, Coh, LCC, ICH, CIS, CSB, NA, NAD, NOM, NOO, NOPM, NOT, NPA, NPM, NRA, AID, CLD, DIT, NMA, NMI, NMO, NOA, NOC, NOD, NOP, SIX

- Quelles sont les caractéristiques d'un réseau bayésien ?

  SELECT ?bayesianNetworkCharacteristic

  WHERE {?sub rdfs:subClassOf ?bayesianNetworkCharacteristic . FILTER (?sub = softwarequality:BayesianNetwork)}

  Résultat : GlassBoxModel, hasStructure some Structure, hasParameters some Parameter

- Quels sont les attributs de qualité non mesurables ?

  SELECT ?nonMeasurableQC

  WHERE {?nonMeasurableQC rdfs:subClassOf?class .

FILTER (?class = softwarequality:NonMeasurableQualityCharacteristic) }

Résultat : Effort, FaultProneness, Maintainability, Impact, Modularity, Reusability

- Quels sont les attributs de qualité mesurables ?

  SELECT ?measurableQC

  WHERE {?measurableQC rdfs :subClassOf?class .

  FILTER (?class = softwarequality :MeasurableQualityCharacteristic) }

  Résultat : Inheritance, Coupling, Complexity, Cohesion

- Quels sont les modèles de qualité contenus dans mon ontologie ?

  SELECT ?qualityModel

  WHERE { ?modelType rdfs :subClassOf ?sub . ?qualityModel rdfs :subClassOf ?modelType .

  FILTER (?sub = softwarequality :QualityModel) . }

  Résultat : BayesianNetwork, DecisionTree, Rules

- Quelles sont les règles de notre ontologie ?

  SELECT DISTINCT ?rules ?condition ?consequence

  WHERE { ?rules rdf :type softwarequality :Rules .

  ?rules softwarequality :hasCondition ?condition .

  ?rules softwarequality :hasConsequence ?consequence . } order by ?rules

Résultat :

Rules	Condition	Consequence
softwarequality:R1	softwarequality:ConditionRule1	softwarequality:ImpactR1
softwarequality:R10	softwarequality:ConditionRule10	softwarequality:ImpactR10
softwarequality:R11	softwarequality:ConditionRule11	softwarequality:ImpactR11
softwarequality:R12	softwarequality:ConditionRule12	softwarequality:ImpactR12
softwarequality:R13	softwarequality:ConditionRule13	softwarequality:ImpactR13
softwarequality:R14	softwarequality:ConditionRule14	softwarequality:ImpactR14
softwarequality:R15	softwarequality:ConditionRule15	softwarequality:ImpactR15
softwarequality:R2	softwarequality:ConditionRule2	softwarequality:ImpactR2
softwarequality:R3	softwarequality:ConditionRule3	softwarequality:ImpactR3
softwarequality:R4	softwarequality:ConditionRule4	softwarequality:ImpactR4
softwarequality:R5	softwarequality:ConditionRule5	softwarequality:ImpactR5
softwarequality:R6	softwarequality:ConditionRule6	softwarequality:ImpactR6
softwarequality:R7	softwarequality:ConditionRule7	softwarequality:ImpactR7
softwarequality:R8	softwarequality:ConditionRule8	softwarequality:ImpactR8
softwarequality:R9	softwarequality:ConditionRule9	softwarequality:ImpactR9

Note : le contenu de chaque cellule est un lien qui donne plus d'informations sur une règle donnée et ses composants.

– Quelles sont les informations contenues dans nos règles ?

SELECT DISTINCT ?rules ?condition ?metric ?sign ?threshold ?impact ?pertinence

WHERE { ?rules rdf :type softwarequality :Rules .

?rules softwarequality :hasCondition ?condition .

?condition softwarequality :hasCondition ?subcondition .

?subcondition softwarequality :hasMetric ?metric .

?subcondition softwarequality :aComparisonSign ?comparison .

?comparison skos :altLabel ?sign .

?condition softwarequality :hasCondition ?subcondition .

?subcondition softwarequality :hasMetric ?metric .

?subcondition softwarequality :hasMetric ?metric .

?subcondition softwarequality :aComparisonSign ?comparison .

?comparison skos :altLabel ?sign .

?subcondition softwarequality :numberToCompareWith ?threshold .

?rules softwarequality :hasConsequence ?consequence .

?consequence softwarequality :hasState ?impact .

?rules softwarequality :hasPertinence ?pertinence . } order by ?rules

Résultat :

Rules	Condition	Metric	Sign	Threshold	Impact	Pertinence
softwarequality:R1	softwarequality:ConditionRule1	softwarequality:AMMIC	'>'	3.0	softwarequality:Weak	softwarequality:Weight32
softwarequality:R1	softwarequality:ConditionRule1	softwarequality:MPC	'<='	21.0	softwarequality:Weak	softwarequality:Weight32
softwarequality:R1	softwarequality:ConditionRule1	softwarequality:OMMIC	'<='	4.0	softwarequality:Weak	softwarequality:Weight32
softwarequality:R10	softwarequality:ConditionRule10	softwarequality:OMMIC	'>'	4.0	softwarequality:Weak	softwarequality:Weight6
softwarequality:R10	softwarequality:ConditionRule10	softwarequality:CBOU	'<='	1.0	softwarequality:Weak	softwarequality:Weight6
softwarequality:R10	softwarequality:ConditionRule10	softwarequality:MPC	'<='	13.0	softwarequality:Weak	softwarequality:Weight6
softwarequality:R11	softwarequality:ConditionRule11	softwarequality:CBOU	'>'	11.0	softwarequality:Average	softwarequality:Weight1/1
softwarequality:R11	softwarequality:ConditionRule11	softwarequality:MPC	'>'	36.0	softwarequality:Average	softwarequality:Weight1/1
softwarequality:R11	softwarequality:ConditionRule11	softwarequality:AMMIC	'<='	5.0	softwarequality:Average	softwarequality:Weight1/1
softwarequality:R12	softwarequality:ConditionRule12	softwarequality:CBONA	'<='	3.5	softwarequality:VeryWeak	softwarequality:Probability0.46
softwarequality:R12	softwarequality:ConditionRule12	softwarequality:CBOU	'<='	0.5	softwarequality:VeryWeak	softwarequality:Probability0.46
softwarequality:R13	softwarequality:ConditionRule13	softwarequality:CBOU	'>'	0.5	softwarequality:Weak	softwarequality:Probability0.54
softwarequality:R13	softwarequality:ConditionRule13	softwarequality:AMMIC	'<='	9.5	softwarequality:Weak	softwarequality:Probability0.54
softwarequality:R13	softwarequality:ConditionRule13	softwarequality:CBONA	'<='	3.5	softwarequality:Weak	softwarequality:Probability0.54
softwarequality:R13	softwarequality:ConditionRule13	softwarequality:CBOU	'<='	1.5	softwarequality:Weak	softwarequality:Probability0.54
softwarequality:R14	softwarequality:ConditionRule14	softwarequality:AMMIC	'>'	9.5	softwarequality:VeryWeak	softwarequality:Probability0.76
softwarequality:R14	softwarequality:ConditionRule14	softwarequality:CBOU	'>'	9.5	softwarequality:VeryWeak	softwarequality:Probability0.76
softwarequality:R14	softwarequality:ConditionRule14	softwarequality:CBONA	'<='	3.5	softwarequality:VeryWeak	softwarequality:Probability0.76
softwarequality:R14	softwarequality:ConditionRule14	softwarequality:CBOU	'<='	1.5	softwarequality:VeryWeak	softwarequality:Probability0.76
softwarequality:R15	softwarequality:ConditionRule15	softwarequality:CBONA	'>'	3.5	softwarequality:Strong	softwarequality:Probability0.48
softwarequality:R15	softwarequality:ConditionRule15	softwarequality:CBOU	'>'	36.5	softwarequality:Strong	softwarequality:Probability0.48
softwarequality:R2	softwarequality:ConditionRule2	softwarequality:AMMIC	'='	9.0	softwarequality:Weak	softwarequality:Weight119/51
softwarequality:R2	softwarequality:ConditionRule2	softwarequality:MPC	'<='	21.0	softwarequality:Weak	softwarequality:Weight119/51
softwarequality:R2	softwarequality:ConditionRule2	softwarequality:OMMIC	'<='	4.0	softwarequality:Weak	softwarequality:Weight119/51
softwarequality:R3	softwarequality:ConditionRule3	softwarequality:OMMIC	'>'	4.0	softwarequality:Weak	softwarequality:Weight68/12
softwarequality:R3	softwarequality:ConditionRule3	softwarequality:CBOU	'<='	7.0	softwarequality:Weak	softwarequality:Weight68/12
softwarequality:R3	softwarequality:ConditionRule3	softwarequality:MPC	'<='	21.0	softwarequality:Weak	softwarequality:Weight68/12
softwarequality:R4	softwarequality:ConditionRule4	softwarequality:CBOU	'>'	7.0	softwarequality:Average	softwarequality:Weight9/1
softwarequality:R4	softwarequality:ConditionRule4	softwarequality:OMMIC	'>'	4.0	softwarequality:Average	softwarequality:Weight9/1
softwarequality:R4	softwarequality:ConditionRule4	softwarequality:MPC	'<='	21.0	softwarequality:Average	softwarequality:Weight9/1

Note : Résultat de quelques règles.

– Quelle est la structure de chaque modèle de qualité ?

SELECT ?modelType ?model ?characteristic

WHERE { ?modelType rdfs :subClassOf ?sub . ?model rdfs :subClassOf ?modelType .

FILTER (?sub = softwarequality :QualityModel) . ?model rdfs :subClassOf ?characteristic}

Résultat :

ModelType	Model	Characteristic
softwarequality:GlassBoxModel	softwarequality:BayesianNetwork	softwarequality:GlassBoxModel
softwarequality:GlassBoxModel	softwarequality:BayesianNetwork	:node18asot2lox10
softwarequality:GlassBoxModel	softwarequality:BayesianNetwork	:node18asot2lox11
softwarequality:GlassBoxModel	softwarequality:DecisionTree	softwarequality:GlassBoxModel
softwarequality:GlassBoxModel	softwarequality:DecisionTree	:node18asot2lox40
softwarequality:GlassBoxModel	softwarequality:DecisionTree	:node18asot2lox41
softwarequality:GlassBoxModel	softwarequality:Rules	softwarequality:GlassBoxModel
softwarequality:GlassBoxModel	softwarequality:Rules	:node18asot2lox88
softwarequality:GlassBoxModel	softwarequality:Rules	:node18asot2lox89
softwarequality:GlassBoxModel	softwarequality:Rules	:node18asot2lox90
softwarequality:GlassBoxModel	softwarequality:Rules	:node18asot2lox91

Note : En cliquant sur les identifiants des noeuds (:nodexxxxxxxxxx) on obtient les informations sur la structure du modèle concerné.

- Quels sont les différents types de métriques ?

  Étant donné que les informations obtenues par inférence ne sont pas considérées dans les réponses, on doit faire une cascade dans la requête.

  SELECT ?metric
  WHERE { ?category rdfs :subClassOf ?sub .
  ?metric rdfs :subClassOf ?category .

  FILTER (?sub = softwarequality :Metric) . }

  Résultat: ClassAttributeCouplingMetric, ClassMethodCouplingMetric, MethodMethodCouplingMetric, ClassCouplingMetric, CohesionMetric, ComplexityMetric, InheritanceMetric

- Quelle est la formule de DIT ?

  SELECT ?formula

  WHERE { ?metric softwarequality:hasName ?name .

FILTER (?name = "Depth of inheritance tree") . metric softwarequality :hasFormula ?formula . }

Résultat : $DIT(c) = 0$, if $Parents(c) = 0$ OR $DIT(c) = 1 + \max\{DIT(c^t) : c^t \in Parents(c)\}$

- Comment est définie la métrique DIT ?

SELECT ?def

WHERE { ?metric softwarequality:hasName ?name .

FILTER (?name = "Depth of inheritance tree") . ?metric dc:description ?def . }

Résultat : Depth of inheritance tree of class is the maximum distance between a class and the root class of the inheritance tree.

- D'où provient la métrique ACAIC ?

SELECT ?provenance

WHERE { ?subject softwarequality :from ?provenance . ?subject rdfs :label ?label .

FILTER (?label = "ACAIC") }

Résultat : briandAndAlArticle. (En cliquant sur la réponse on a les informations sur cet article)

- Quels sont les concepts qui ont des instances ?

SELECT DISTINCT ?concept

WHERE { ?concept a owl:Class . ?ind rdf:type ?concept } order by ?concept

Résultat: BayesianNetwork, ClassAttributeCouplingMetric, ClassMethodCouplingMetric, MethodMethodCouplingMetric, ClassCouplingMetric, CohesionMetric, ComplexityMetric, InheritanceMetric, CombinedCondition, ComparisonSign, EndNode, EntryVariableParameter, Impact, IntermediateNode, IntermediateVariableParameter, Probability, Rules, SimpleCondition, State, Structure, Weight, Article, Book, Conference, Organization, Symposium, Workshop, Person.

- Quelles sont les métriques prises en considération dans les instances de modèles de qualité ?

SELECT DISTINCT ?metric

WHERE { ?subject softwarequality :hasInput ?metric .

OPTIONAL { ?sub softwarequality :hasMetric ?metric .} }

Résultat : AMMIC, CBONA, CBOU, MPC, OMMIC

CONCLUSION

Nous avons commencé ce travail avec pour but de contribuer à réduire le fossé entre les connaissances accumulées et le besoin grandissant de comprendre et d'exploiter ces connaissances. Notre but était de trouver un moyen de partager efficacement les observations empiriques et les bonnes pratiques du génie logiciel qui portent sur la qualité logicielle. Nous avons exploité ce que le web semantique offre pour pouvoir partager et diffuser des connaissances. Nous avons dû répondre aux questions telles que : existe-t-il des modèles pour déterminer la qualité d'un logiciel ? Quels sont ces modèles ? Comment les utilise-t-on ? Quelles sont les caractéristiques importantes pour déterminer la qualité d'un logiciel ? Quels sont les outils qui peuvent nous aider dans notre travail ? Quel est le formalisme adéquat ? etc.

Après un état de l'art des travaux portant sur la qualité logicielle et ses modèles, nous avons choisi les modèles de qualité populaires (arbres de décision, réseaux bayésiens et règles) ainsi que les métriques orientées-objets utilisées dans ces travaux pour construire une ontologie. Nous avons donc regroupé en une ontologie les connaissances et données qui aident à prédire la qualité d'un produit logiciel. Pour développer cette ontologie, nous avons fait une étude approfondie des processus existants de développement d'une ontologie. Nous avons utilisé le processus METHONTOLOGY qui est plus détaillé, précis et mature, car ces détails et précisions facilitent la maintenance et l'évolution de notre ontologie.

Cette ontologie élimine le temps de fouille et de collecte de connaissances nécessaires pour connaitre les facteurs influençant la qualité logicielle et savoir s'en servir. Les

métadonnées de provenance des métriques de notre ontologie renseignent sur la source de ces métriques, et par la même occasion renforcent leur crédibilité, leur pertinence et leur utilité. Cette ontologie est destinée à être utilisée dans un système d'apprentissage ou d'aide à la décision. En plus de pouvoir la consulter sur internet sous le format HTML, il est aussi possible de la consulter dans un éditeur d'ontologies. En ayant le droit d'écriture (il faut faire une demande pour cela) il est possible d'interroger et aussi d'enrichir l'ontologie dans son environnement de développement (éditeur d'ontologies) quelque soit la situation géographique de l'utilisateur. Cet aspect est très important car le but est de centraliser le maximum de connaissances sur la qualité logicielle et les rendre utilisables, réutilisables et accessibles partout. En quelques requêtes, il est possible de connaitre les facteurs influençant la qualité logicielle et comment s'en servir. Malgré l'existance de plusieurs répertoires d'ontologies en ligne, seul le repertoire Ontosearch était le mieux adapté à nos besoins (recherche par mots-clés, formuler des requêtes SPARQL). Cependant, celui-ci avait comme contrainte l'utilisation de OWL-Lite. OWL-Lite ayant une faible expressivité, l'utiliser pour notre ontologie nous empêchait d'exprimer fidèlement toutes nos connaissances. Pour cette raison, nous avons utilisé l'outil Sesame.

On pourrait aller plus loin en implémentant un module qui permettrait de transformer les instances d'un modèle de qualité donné en instances d'un autre modèle de qualité. Ces transformations éviteraient à l'utilisateur de transformer lui même les instances d'un modèle de qualité en instances du modèle de qualité qu'il aimerait utiliser. Ces transformations éliminent aussi la contrainte d'utiliser un modèle différent de celui qu'on aurait voulu utiliser.

Pour atteindre un plus grand nombre d'utilisateurs, on pourrait réaliser une revue de littérature, car il y a des cas où différents acronymes désignent une même métrique.

Exemple : NAD (Number of Abstract Datatypes) et CTA (Coupling Through Abstract datatypes) font allusion à la même métrique. Pareil pour NOA (Number Of

Attributes) et NA (Number of Attributes).

En incluant toutes les variantes d'acronymes ou d'appellations des métriques et d'autres termes de l'ontologie, cela élimine toute confusion et permet à tout acteur du domaine de se retrouver quelque soit l'acronyme ou l'appellation qui lui est familière. Dans le même souci d'atteindre plus d'utilisateurs, il faudra réaliser des appariements (« matching ») avec les ontologies existantes du domaine.

Cette ontologie est destinée à être intégrée dans un système d'aide la décision en génie logiciel afin de semi-automatiser le processus d'amélioration de la qualité logicielle. Elle peut aussi être intégrée dans un système d'aide à l'apprentissage qui a pour but d'apprendre assez tôt aux acteurs du génie logiciel les connaissances importantes qui permettent de garantir la bonne qualité d'un logiciel. Pour réduire le fossé entre la production des connaissances et leur exploitation, il y a d'autres contributions à apporter comme : définir des patrons de consommation pour l'amélioration de la consommation des connaissances, définir un nuage de données liées (« Linked Open Data cloud »), etc.

APPENDICE A

COMPLÉMENT DE LA SPÉCIFICATION

Cette partie complète la section 3.1 en fournissant, conformément à l'étape de spécification du processus METHONTOLOGY, le reste des concepts, instances, relations et propriétes de notre ontologie.

- Liste complémentaire des instances :
    - Composantes modèles de qualité : StructureScenario2, StructureScenario3, StructureScenario4, AMMICNodeScenario2, AMMICNodeScenario3, AMMICNodeScenario4, CBONANodeScenario2, CBONANodeScenario3, CBONANodeScenario4, CBOUNodeScenario2, CBOUNodeScenario3, CBOUNodeScenario4, MPCNodeScenario2, MPCNodeScenario3, MPCNodeScenario4, OMMICNodeScenario2, OMMICNodeScenario3, OMMICNodeScenario4, DesignMetricsNodeScenario2, DesignMetricsNodeScenario3, DesignMetricsNodeScenario4, ImplementationMetricNodeScenario2, ImplementationMetricsNodeScenario3,

ImplementationMetricsNodeScenario4, ImpactNodeScenario2, ImpactNodeScenario3, ImpactNodeScenario4, EVPTableAMMICNodeScenario2[27], EVPTableAMMICNodeScenario3, EVPTableAMMICNodeScenario4, EVPTableCBONANodeScenario2, EVPTableCBONANodeScenario3, EVPTableCBONANodeScenario4, EVPTableCBOUNodeScenario2, EVPTableCBOUNodeScenario3, EVPTableCBOUNodeScenario4, EVPTableMPCNodeScenario2, EVPTableMPCNodeScenario3, EVPTableMPCNodeScenario4, EVPTableOMMICNodeScenario2, EVPTableOMMICNodeScenario3, EVPTableOMMICNodeScenario4, IVPTableDMNodeScenario2[28], IVPTableDMNodeScenario3, IVPTableDMNodeScenario4, IVPTableIMNodeScenario2, IVPTableIMNodeScenario3, IVPTableIMNodeScenario4, EndVPTableImpactNodeScenario2[29], EndVPTableImpactNodeScenario3, greaterThanOrEqualTo, greaterThan, EndVPTableImpactNodeScenario4, lessThan, lessThanOrEqualTo, Average, equalTo, Strong, VeryStrong, VeryWeak, Weak, AverageImpactNodeScenario2, AverageImpactNodeScenario3, AverageImpactNodeScenario4, LargeAMMICNodeScenario2, LargeAMMICNodeScenario4, LargeCBONANodeScenario2, LargeCBONANodeScenario3,

[27] EVP: Entry Variable Parameter.

[28] IVP: Intermediate Variable Parameter.

[29] EndVP: End Variable Parameter.

LargeCBOUNodeScenario2, LargeMPCNodeScenario2, NoDMNodeScenario230, LargeCBOUNodeScenario3, NoDMNodeScenario4,NoIMNodeScenario231, NoIMNodeScenario3, NoIMNodeScenario4, SmallAMMICNodeScenario2, SmallAMMICNodeScenario4, SmallCBONANodeScenario2,SmallCBONANodeScenario3, SmallCBOUNodeScenario2,SmallCBOUNodeScenario3, SmallMPCNodeScenario2,StrongImpactNodescenario3, SmallOMMICNodeScenario2, StrongImpactNodescenario2, StrongImpactNodescenario4, WeakImpactNodescenario3, WeakImpactNodescenario2, WeakImpactNodescenario4, YesDMNodeScenario2,YesDMNodeScenario4, YesIMNodeScenario2, YesIMNodeScenario3, YesIMNodeScenario4, Impact, ImpactR1, ..., ImpactR15.

- Documents : article 1 et 2 de Bansiya et al., article de Bieman et al., article 1 et 2 de Briand et al., article 1 et 2 de Chidamber et al., article de Hitz et al., article 1 et 2 de Li et al., article de Liang. Article de Li, article de Michura et al., article de Tegarden et al., article de Lake et Cook, livre de Henderson-Sellers, livre de Lanza et Marinescu, livre de Lorenz et Kidd.

- Personnes : B. K. Kang, B. Liang, B. Montazeri, C. F. Kemerer, F.J. Wang, J.M. Bieman, L.Briand, M. Hitz, P. Devanbu, S.F. Wu, S. Henry, S.R. Chidamber, W. Li, W. Melo, Y.S. Lee, A. Chaumun, A. Lake, B. Henderson-Sellers, C. Cook, C. Davis, D. Monarchi, D. Tegarden, F. Lustman, G. Saint-Denis, H. Kabaili, H. Lounis, J. Bansiya, J. Daly, J. Michura, J. Wuest, K. Emam, M. Capretz, M.

Lanza, N. Goel, R. Keller, S. Benlarbi, S. Rai, S. Sheetz.

- Évènements : International Conference on Software Engineering, Object-Oriented Programming : Systems, Language and Application; Conference on Software Maintenance and Reengineering, Conference on Software Quality, Conference on the Cognitive Science Society, Symposium on Applied Corporate Computing, Sym- posium on Software Reusability, Workshop on Software Metrics.

- Organismes : IEEE Transactions on software Engineering, SIGPLAN, Software Engineering Notes, journal on Systems Software, Dr Dobbs journal, Prentice Hall, Springer, decision support systems, International Scholarly Research Network Software Engineering.

- Liste complémentaire des relations : hasArc, aComparisonSign, presentedAt.

- Liste complémentaire des propriétés : numberToCompareWith.

APPENDICE B

COMPLÉMENT DU DICTIONNAIRE DES TERMES

Cette partie complète la section 3.2.1 en fournissant, conformément à l'étape de conceptualisation du processus METHONTOLOGY, le reste des définitions des termes de notre ontologie.

Description des concepts liés à la provenance des métriques

Document : sert de référence pour une instance (dans notre cas il s'agit des instances du concept métrique).

Provenance : donne les références des métriques figurant dans l'ontologie.

Article : document où l'on trouve les instances de notre ontologie.

Livre : document où l'on trouve les instances de notre ontologie.

Évènement : rassemblement au cours duquel sont présentés des articles.

Conférence : c'est un évènement.

Workshop : c'est un évènement.

Symposium : c'est un évènement.

Organisme : organisme de publication d'articles ou de livres.

Personne : auteur d'un document, d'une métrique.

Description des articles

Article de Bieman et al. : «Cohesion and Reuse in an Object-Oriented System» rédigé par Bieman et Kang.

Article de Briand et al. : « An Investigation inti Coupling Measures for C++ » rédigé par Briand et Devanbu.

Article 1 de Chidamber et al. : « A Metrics Suite for Object Oriented Design » rédigé par Chidamber et Kemerer.

Article 2 de Chidamber et al. : « Towards a Metrics Suite for Object Oriented design » rédigé par Chidamber et Kemerer.

Article de Hitz et al. : « Measuring Coupling and Cohesion in Object-Oriented Systems » rédigé par Hitz et Montazeri.

Article 1 de Li et al. : « Object-Oriented Metrics that Predict Maintainability » rédigé par Li et Henry.

Article 2 de Li et al. : « Object-Oriented Metrics which Predict Maintainability » rédigé par Li et Henry.

Article de Liang : « Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow » rédigé par Liang, Wang, Wu et Lee.

Description des instances de *personne*

B.K. Kang : auteur de l'article «Cohesion and Reuse in an Object-Oriented System».

B. Liang : auteur de l'article « Measuring the Coupling and Cohesion of an Object- Oriented Program Based on Information Flow ».

B. Montazeri : auteur de l'article « Measuring Coupling and Cohesion in Object- Oriented Systems ». C. F. Kemerer : auteur des articles « A Metrics Suite for Object Oriented Design » et « Towards a Metrics Suite for Object Oriented design ».

F.J. Wang : auteur de l'article « Measuring the Coupling and Cohesion of an Object- Oriented Program Based on Information Flow ».

J.M. Bieman : Auteur de l'article «Cohesion and Reuse in an Object-Oriented System».

L. Briand : auteur de l'article « An Investigation inti Coupling Measures for C++ ».

M. Hitz : auteur de l'article « Measuring Coupling and Cohesion in Object-Oriented Systems ».

P. Devanbu : auteur de l'article « An Investigation inti Coupling Measures for C++».

S.F. Wu : auteur de l'article « Measuring the Coupling and Cohesion of an Object- Oriented Program Based on Information Flow ».

S. Henry : auteur des articles « Object-Oriented Metrics which Predict Maintainability » et « Object-Oriented Metrics that Predict Maintainability ».

S.R. Chidamber : auteur des articles « A Metrics Suite for Object Oriented Design » et « Towards a Metrics Suite for Object Oriented design ».

W. Li : auteur des articles « Object-Oriented Metrics which Predict Maintainability » et « Object-Oriented Metrics that Predict Maintainability ».

W. Melo : Y.S. Lee : auteur de l'article « Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow ».

Description des instances d'*évènement* et des différents degré d'impact

ConfSwQuality : Software quality conference est une conférence sur la qualité logicielle qui regroupe les acteurs de ce domaine. Des articles portant sur ce domaine et reconnus pertinents par un comité y sont présentés.

ICSE97 : c'est la 19e conférence sur le génie logiciel qui a eu lieu en 1997 à Boston aux États-Unis.

IEEEswEng : IEEE software engineering est un organisme de publication.

OOPSLA : conférence sur Object-Oriented Programming : Systems, Languages and Applications.

SACC : symposium international Symposium on Applied Corporate Computing.

SSR : symposium ACM Symposium on Software Reusability.

SWEngNotes : ACM SIGSOFT Software Engineering Notes est un organisme de publication des travaux sur le génie logiciel.

SysSwJournal : Journal of systems and software est une revue qui publie les articles qui traitent de tous les aspects des méthodes de programmation, du génie logiciel et les problèmes des systèmes qui ont un lien avec ces aspects.

VeryStrong : c'est un degré d'impact qu'une modification peut avoir sur un logiciel.

VeryWeak : c'est un degré d'impact qu'une modification peut avoir sur un logiciel.

Weak : c'est un degré d'impact qu'une modification peut avoir sur un logiciel.

APPENDICE C

COMPLÉMENT DU DICTIONNAIRE DES CONCEPTS

Cette partie partie complète la section 3.2.3 en fournissant, conformément à l'étape de conceptualisation du processus METHONTOLOGY, la partie manquante du dictionnaire des concepts de notre ontologie.

Nom du concept	Instances	Attributs des instances	Données sur les instances	Relations
Arc	s/o	s/o	s/o	s/o
Document	s/o	s/o	title, author, pages, date-of-publication	presentedAt
Article	article de Bieman et al., article de Briand et al., article 1 et 2 de Chidamber et al., article de Churcher et al., article de Hitz et al., article 1 et 2 de Li et al., article de Liang	s/o	volume, issue, publisher	s/o
Book	s/o	s/o	s/o	s/o
Event	s/o	s/o	title, location	s/o
Conference	ConfSw- Quality, ICSE97, OOPSLA	s/o	s/o	s/o
Symposium	SACC, SSR	s/o	s/o	s/o
Workshop	s/o	s/o	s/o	s/o
Organization	IEEEswEng, SIGPLAN, SWEngNotes, SysSwJournal	name	s/o	s/o
Person	bkKang, bLiang, bMontazeri, cfKemerer, fj Wang, jmBieman, lBriand, mHitz, mjShepperd, niChurcher, pDevanbu, sfWu, sHenry, srChidamber, wLi, wMelo, ysLee	name	s/o	s/o

APPENDICE D

COMPLÉMENT DE LA TABLE DES RELATIONS BINAIRES

Cette partie complète la section 3.2.4 en fournissant, conformément à l'étape de conceptualisation du processus METHONTOLOGY, les informations sur les relations binaires restantes.

Nom de la relation : hasArc

Concept source : DecisionTree

Cardinalité de la source : 1

Concept cible : Arc

Cardinalité de la cible : 2, n

Relation inverse : -

Nom de la relation : presentedAt

Concept source : Document

Cardinalité de la source : 1, n

Concept cible : Evenement

Cardinalité de la cible : 1, n

Relation inverse : -

APPENDICE E

COMPLÉMENT DE LA TABLE DES INSTANCES

Cette partie complète la section 3.2.5 en fournissant, conformément à l'étape de conceptualisation du processus METHONTOLOGY, les informations sur le reste des instances de notre ontologie.

Instance	**Données**	**Valeur**
biemanAndAlArticle	Titre	Cohesion and Reuse in an Object-Oriented System
	Auteur(s)	J.M. Bieman, B. K. Kang
	Pages	259-262
	Date de publication	1995

Instance	**Données**	**Valeur**
briandAndAlArticle	Titre	An Investigation into Coupling Measures for C++
	Auteur(s)	L. Briand, P. Devanbu, W. Melo
	Pages	412-421
	Date de publication	05-1997
	Volume	-
	Issue	-

	Éditeur	-

Instance	**Données**	**Valeur**
chidamberAndAlArticle	Titre	A Metrics Suite for Object-Oriented Design
	Auteur(s)	C.F. Kemerer, S.R. Chidamber
	Pages	476-493
	Date de publication	06-1994
	Volume	20
	Issue	6
	Éditeur	IEEE software engineering

Instance	**Données**	**Valeur**
chidamberAndAlArticle2	Titre	Towards a metrics Suite for Object-Oriented Design
	Auteur(s)	C.F. Kemerer, S.R. Chidamber
	Pages	197-
	Date de publication	10-1991
	Volume	26
	Issue	11
	Éditeur	SIGPLAN
Instance	**Données**	**Valeur**
churcherAndAlArticle	Titre	Towards a Conceptual Framework for Object-Oriented Software
	Auteur(s)	N. I. Churcher, M. J. Shepperd
	Pages	69-76
	Date de publication	1995

	Volume	20
	Issue	2
	Éditeur	Software engineering Notes

Instance	**Données**	**Valeur**
hitzAndAlArticle	Titre	Measuring Coupling and Cohesion in Object-Oriented Systems
	Auteur(s)	M. Hitz, B. Montazeri
	Pages	-
	Date de publication	10-1995
	Volume	-
	Issue	-
	Éditeur	-

Instance	**Données**	**Valeur**
liAndAlArticle	Titre	Object-Oriented Metrics that Predict Maintainability
	Auteur(s)	W. Li
	Pages	111-122
	Date de publication	11-1993
	Volume	23
	Issue	2
	Éditeur	Journal of systems and software

Instance	**Données**	**Valeur**
LiAndAlArticle2	Titre	Object-Oriented Metrics which Predict Maintainability
	Auteur(s)	W. Li, S. Henry
	Pages	35

	Date de publication	02-1993
	Volume	-
	Issue	-
	Éditeur	-

Instance	**Données**	**Valeur**
liangAndAlArticle	Titre	Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow
	Auteur(s)	B. Liang, F.-J. Wang, S.F. Wu, Y.-S. Lee
	Pages	-
	Date de publication	1995
	Volume	-
	Issue	-
	Éditeur	-

Instance	**Attributs**	**Valeur**
bkKang	Nom	Byung-Kyoo Kang

Instance	**Attributs**	**Valeur**
bLiang	Nom	Baoshuang Liang

Instance	**Attributs**	**Valeur**
bMontazeri	Nom	Behzad Montazeri

Instance	**Attributs**	**Valeur**

cfKemerer	Nom	Chris F. Kemerer

Instance	**Attributs**	**Valeur**
fjWang	Nom	F.-J. Wang

Instance	**Attributs**	**Valeur**
jmBieman	Nom	James M. Bieman

Instance	**Attributs**	**Valeur**
jPearl	Nom	Judea Pearl

Instance	**Attributs**	**Valeur**
lBriand	Nom	Lionel Briand

Instance	**Attributs**	**Valeur**
mHitz	Nom	Martin Hitz

Instance	**Attributs**	**Valeur**
mjShepperd	Nom	Martin J. Shepperd

Instance	**Attributs**	**Valeur**
niChurcher	Nom	Neville I. Churcher

Instance	**Attributs**	**Valeur**
pDevanbu	Nom	Prem Devanbu

Instance	**Attributs**	**Valeur**
sfWu	Nom	S.-F. Wu

Instance	Attributs	Valeur
sHenry	Nom	Sallie Henry

Instance	Attributs	Valeur
srChidamber	Nom	Shyam R. Chidamber

Instance	Attributs	Valeur
wLi	Nom	Wei Li

Instance	Attributs	Valeur
wMelo	Nom	Walcelio Melo

Instance	Attributs	Valeur
ysLee	Nom	Yun-Sun Lee

Instance	Données	Valeur
ICSE97	Titre	19th International Conference on Software Engineering
	Lieu	Boston, USA

Instance	Données	Valeur
OOPSLA	Titre	Conference on Object-Oriented Programming: Systems, Languages and Applications
	Lieu	-

Instance	Données	Valeur
SACC	Titre	International Symposium on Applied Corporate Computing
	Lieu	Monterrey, Mexico

Instance	Données	Valeur
SSR	Titre	ACM Symposium on Software Reusability
	Lieu	-

Instance	Données	Valeur
ConfSwQuality	Titre	International Conference on Software Quality
	Lieu	Maribor, Slovenia

Instance	Données	Valeur
IEEEswEng	Nom	IEEE Transactions on Software Engineering

Instance	Données	Valeur
SIGPLAN	Nom	SIGPLAN Notes

Instance	Données	Valeur
SWEngNotes	Nom	Software Engineering Notes

Instance	Données	Valeur
SysSwJournal	Nom	Journal of Systems and Software

BIBLIOGRAPHIE

[1] Khoshgoftaar, T.M. (1999). Predicting Software Engineering. *IEEE Computer*, 32(10), 32-37.

[2] Basili, V. Condon, El Emam, L. Hendrick, R.B. et Melo, W. (1997). Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components. *IEEE 19th International Conference on Software Engineering*. Actes du colloque, 1997, Boston.

[3] Lounis, L. Gayed, T. et Boukadoum M. (2011, novembre). Using Efficient Machine-Learning Models to Assess Two Important Quality Factors : Maintainability and Reusability. *6th International Conference on Software Process and Product Measurement*. Actes du colloque, novembre 2011, Nara, Japan.

[4] Lounis, L. Gayed, T. et Boukadoum M. (2011, novembre). Machine-Learning Models for Software Quality : a Compromise Between Performance and Intelligibility. *IEEE 23rd International Conference on Tools with Artificial Intelligence*. Actes du colloque, novembre 2011, Boca Raton, USA.

[5] Schauer, R. Keller, R.K. Laguë, B. Knapen, G. Robitaille, S. et Saint-Denis, G. (2001). The SPOOL Design Repository : Architecture, Schema, and Mechanisms. *Springer-Verlag*.

[6] Miller, G.A. (2013). Nouns in WordNet : A Lexical Inheritance System.

[7] Alpaydin, E. (2010). Introduction to Machine Learning. *MIT Press*, second edition.

[8] Liu, L. et Özsu, T.M. (2009). Encyclopedia of Database Systems. *Springer-Verlag*. Récupéré de http://tomgruber.org/writing/ontology-définition-2007.htm.

[9] Gruber, T.R. (1995). Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal Human-Computer Studies*, 43(5/6), 907-928.

[10] Musen, M.A. (1992). Dimensions of knowledge sharing and reuse. *Computers and Biomedical Research,* 25, 435-467.

[11] Gruber, T.R. (1993, juin). A Translation Approach to Portable Ontology specifications. *Knowledge Acquisition,* 5(2), 199-220.

[12] Noy, N.F. et McGuinness, D.L. (2001). Ontology Development 101 : A Guide to Creating Your First Ontology. *Stanford University*, Stanford, California.

[13] Petrov, V. (2011). Ontological Landscapes : Recent Thought on Conceptual Interfaces Between Science and Philosophy. *Ontos Verlag.*

[14] Chaput, B. Benayache, A. Barry, C. et Abel, M.H. (2004). Une expérience de construction d'ontologie d'application pour indexer les ressources d'une formation en statistique. 30ième Journées Françaises de statistique,. Actes du colloque, 2004, Montpellier, France.

[15] Cardoso, J. et Escorcio, L. (2007, mars). Editing Tools for Ontology Construction, Semantic Web Services : Theory, Tools and Applications. *Idea Group.*

[16] Antoniou, G. et van Harmelen, F. (2008). A semantic Web Primer. *The MIT Press*, 2nd edition.

[17] Grueninger, M. et Fox, M.S. (1995, avril). Methodology for the design and evaluation of ontologies. *International Joint Conference AI Workshop on Basic Ontological Issues in Knowledge Sharing.* Actes du colloque, avril 1995.

[18] Uschold, M. et Gruninger, M. (1996, juin). Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(2).

[19] Fernandez, M. Gómez-Pérez, A. et Juristo, N. (1997). METHONTOLOGY : From Ontological Art Towards Ontological Engineering. *AAAI Symposium on Ontological Engineering*. Actes du colloque, 1997, Stanford, California.

[20] Fernández-López, M. (1999). Overview of Methodologies for Building Ontologies. *Workshop on Ontologies and Problem-solving Methods : Lessons Learned and Future Trends.* Actes du colloque, 1999, Stockholm.

[21] Skuce, D. (1995). Conventions for reaching agreement on shared ontologies. *9th Knowledge Acquisition for Knowledge Based Systems Workshop*. Actes du colloque, 1995, Banff Conference Centre, Alberta, Canada.

[22] Swartout, B. Ramesh, P. Knight, K. et Russ, T. (1997, mars). Toward Distributed Use of Large-Scale Ontologies. *Symposium on Ontological Engineering of AAAI.* Actes du colloque, mars 1997, Stanford, California.

[23] B Swartout, B. Ramesh, P. Knight, K. et Russ, T. (1996, septembre). Toward Dis tributed Use of Large-Scale Ontologies. Récupéré de http://www.isi.edu/isd/banff_paper/Banff_final_web/Banff_96_final_2.html.

[24] Gómez-Pérez, A. Juristo, N. et Pazos, J. (1995). Evaluation and assessment of knowledge sharing technology. *Towards Very Large Knowledge Bases - Knowledge Building and Knowledge Sharing, IOS Press*, 289-296.

[25] Fernández-López, M. Gómez-Pérez, A. Pazos-Sierra, A. et Pazos-Sierra, J. (1999). Building a Chemical Ontology Using Methontology and the Ontology Design Environment. *IEEE Intelligent Systems & their applications*, 37-46.

[26] Gómez-Pérez, A. et Rojas, M.D.(1999). Ontological Reengineering and Reuse. *European Knowledge Acquisition Workshop (EKAW)*. Actes du colloque, 1999.

[27] Falbo, R.A. Guizzardi, G. et Duarte, K.C.(2002). An ontological approach to domain engineering. 14th International Conference on Software Engineering and Knowledge Engineering. Actes du colloque, 2002.

[28] d'Aquin, M et Noy, N.F. (2011). Where to Publish and Find Ontologies? A survey of Ontology Libraries. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 11(0).

[29] Wagner, S. (2010, novembre). A Bayesian Network Approach to Assess and Predict Software Quality Using Activity-Based Quality Models. *Information and Software Technology*, 52(11), 1230-1241.

[30] Abdi, M. K. Lounis, H. et Sahraoui, H. (2006). Analyzing Change Impact in Object-Oriented Systems.

[31] Abdi, M. K. Lounis, H. et Sahraoui, H. (2009, juillet). Predicting Change Impact in Object-Oriented Applications with Bayesian Networks. 33rd IEEE International Computer Software and Applications Conference. Actes du colloque, juillet 2009, Seattle, WA .

[32] Murphy, K.P. (2002). Dynamic Bayesian Networks : Representation, inference and learning. (Thèse de doctorat), UC Berkeley, CA.

[33] Wooff, D.A. Goldstein, M. et Coolen, F.P.A. (2002). Bayesian graphical models for software testing, *IEEE Transactions on Software Engineering*, 28(5), 510-525.

[34] Stamelos, I. Angelis, L. Dimou, P. et Sakellaris, E. (2003). On the use of Bayesian Belief Networks for the prediction of software productivity, *Information and Software Technology*, 45(1), 51-60.

[35] Bibi, S. et Stamelos, I. (2004). Software process modeling with Bayesian belief networks. 10th International Software Metrics Symposium. Actes de colloque, 2004, Chicago.

[36] Neil, M. Krause, P. et Fenton, N. E. (2003). Software quality prediction using Bayesian Networks. *Software Engineering with Computational Intelligence*. Actes du colloque, Amsterdam, Kluwer.

[37] Fenton, N.E Marsh, W. Neil, M. Cates, P. Forey, S. et Tailor, T. (2004, mai). Making resource decisions for software projects. *IEEE Computer Society, 26th International Conference on Software Engineering*. Actes du colloque, mai 2004, Edinburgh, United Kingdom.

[38] Wang, H. Peng, F. Zhang, C. et Pietschker, A. (2006, octobre). Software project level estimation model framework based on Bayesian Belief Networks. *6th International Conference on Quality Software*, 209-218.

[39] Lanza, M. et Marinescu, R. (2006). Object-oriented Metrics in Practice : Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-oriented Systems. *Springer*.

[40] Gómez-Pérez, A. (1996). Towards a Framework to Verify Knowledge Sharing Technology, *Expert Systems with Applications*, 11(4), 519-529.

[41] Deissenboeck, F. Juergens, E. Lochmann, K. et Wagner, S. (2009). Software Quality Mod els, Usage Scenarios and Requirements. *IEEE Computer Society, 7th International Workshop on Software Quality.*

[42] Lindvall, M.(1999, novembre). Measurement of change : Stable and Change-Prone Constructs in a commercial C++ System. *6th International Software Metrics Symposium.* Actes du colloque, novembre 1999, Boca Raton, Florida.

[43] Abdi, M.K. Lounis, H. et Sahraoui, H.(2006, juiillet). Using Coupling Metrics for Change Impact Analysy in Object-oriented Systems. *10th Workshop on Quantitative Approaches in Object-Oriented Software Engineering*. Actes du colloques, juillet 2006, Nantes, France.

[44] Sahraoui, H.A. Godin, R. et Miceli, T. (2000). Can metrics help to bridge the gap between the improvement of OO design quality and its automation ? *International conference on Software Maintenance.*

[45] Murthy, S.K.(1998). Automatic Construction of Decision Trees from Data : A multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, 2(4), 345-389.

[46] Pfleeger, S.L. (1990, mai). A Framework for Software Maintenance Metrics, *IEEE Transactions on Software Engineering*, 320-327.

[47] World Wide Web Consortium. SPARQL QUERY Language for RDF. Récupéré de http://www.w3.org/TR/rdf-sparql-query/

[48] Khoshgoftaar, T.M. Liu, Y. et Seliya, N. (2003, novembre). Genetic Programming-Based Decision Trees for Software Quality Classification. *IEEE 15th International Conference on Tools with Artificial Intelligence.* Actes du colloque, novembre 2003.

[49] Luo, Y. (2010). Improving Software Quality Using An Ontology-based Approach. (Thèse de doctorat publiée). Louisiana State University and Agricultural and Mechanical College.

[50] Kabaili, H. (2002). Changeabilité des logiciels orientes objet propriétés architecturales et indicateurs de qualité. (Thèse de doctorat non publiée). Université de Montréal, Canada.