N° d'ordre :……..

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR & DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DJILLALI LIABES - SIDI BEL ABBES
FACULTE DES SCIENCES EXACTES
DEPARTEMENT D'INFORMATIQUE

THESE DE DOCTORAT EN SCIENCE

Présentée et soutenue par
Hamri Mohamed Mehdi

Spécialité : Informatique
Option : Système d'informations et connaissances

Ingénierie ontologique dirigée par les modèles

Dirigée par : Pr. Sidi Mohamed Benslimane
Ecole Supérieur en Informatique, SBA.

Soutenue le --/--/2017, devant le jury composé de :

Président : Pr. Lehirech Ahmed	Professeur	Université Djillali Liabes, SBA.
Examinateur : Pr. Malki Mimoun	Professeur	Ecole Supérieur en Informatique, SBA.
Examinateur : Dr Amar Bensaber Djamel	M.C.A.	Ecole Supérieur en Informatique, SBA
Examinateur : Dr. Djelloul Bouchiha	M.C.A.	Centre Universitaire de Naama.
Examinateur : Dr. Adjouj Reda	M.C.A.	Université Djillali Liabes, SBA.

Année universitaire : 2016-2017

Remerciements

قال النبي صلى الله عليه وسلم :"من لا يشكر الناس لا يشكر الله"

Tout d’abord, je remercie dieu d’avoir mit sur mon chemin les gens qui m’ont permit de réaliser ce travail.

Ensuite je veux exprimer mon immense gratitude envers mon directeur de thèse, le professeur Sidi Mohamed Benslimane, pour avoir dirigé mes premiers pas dans le monde de la recherche au travers de sujets intéressants. Il a su être très présent au cours de ces années tout en me laissant une grande liberté dans mes choix. Ses idées, sa motivation, ainsi que sa grande curiosité et culture scientifique m’ont servi de modèle. Grâce à ses conseils, sa patience et sa gentillesse, j’ai pu mener à bien ce projet de thèse. C’est pour tout cela que je le remercie infiniment et je me permet de lui dire « tu est un grand frère pour moi »

J’aimerais saisir cette opportunité scientifique, pour remercier tout particulièrement Messieurs les membres du jury de m’avoir fait l’honneur d’accepter de consacrer une partie de leur temps que je sais précieux, à l’examen, à l’appréciation, et au jugement scientifique des résultats et des conclusions du présent travail de recherche.

Que Monsieur le Président du jury, et Messieurs les membres du jury de la soutenance de ma thèse, veuillent bien trouver ici l’expression de ma sincère reconnaissance pour leur bienveillante attention, et surtout pour les efforts méritoires (que je salue au passage comme il sied aux personnalités scientifiques de leur rang) qu’ils ont déployés chacun dans son domaine de spécialité, dans le cadre de la promotion de l’enseignement et de la recherche en informatique.

Je ne puis oublier, en cette heureuse circonstance, de remercier de tout cœur, mes parents, mon épouse, mon petit ange Meriem naïla, et mes frères ma sœur, ma belle famille, ainsi que tous mes amis pour leur affection, leur soutien moral et surtout leur remarquable compréhension tout au long de l’élaboration de cette thèse.

Que toutes et tous veuillent bien trouver ici l’expression de ma sincère gratitude.

Table des matières

Table des figures

Liste des tableaux

Résumé

Le Web sémantique est la direction principale du développement futur du Web. Les ontologies sont la partie la plus importante des applications du Web sémantique. Les techniques d'intelligence artificielle sont les plus utilisés pour la création d'ontologie, mais ces techniques sont plus liées aux laboratoires de recherche qu'un public plus large. Durant la dernière décennie, il ya eu beaucoup de propositions visant à utiliser des techniques de génie logiciel, en particulier l'UML pour la création d'ontologie, car elle est la norme du génie logiciel la plus acceptées, afin d'apporter un processus de développement d'ontologie plus proche d'une large population de praticiens. Cependant, UML est basé sur un paradigme orienté objet, et a une certaine limite en matière de développement d'ontologie. Ces limitations peuvent être surmontées en utilisant les extensions d'UML (c. profils UML), ainsi que les normes d'OMG tel que l'Architecture dirigé par les modèles (MDA), le métamodéle de définition d'ontologie (ODM). Actuellement, l'utilisation conjointe d'MDA et du web sémantique a atteint un certain degré de maturité et un franc succès dans le domaine de l'ingénierie d'ontologies.

Cette thèse est une contribution au domaine des procédés logiciel (PL). Nous y proposons le développement d'une ontologie basée sur la norme ISO/IEC 24744 (Software Engineering-Metamodel for Development Methodologies - SEMDM). L'ontologie est générée à partir du méta modèle décrit dans la norme en utilisant un processus de transformation basé MDA . Elle décrit, entre autres, les activités et les tâches à exécuter, les artefacts à manipuler (créer, utiliser ou modifier) et les personnes impliquées. Cette ontologie aidera à systématiser et à mieux maitriser l'utilisation du modèle de la norme ISO/24744. L'ontologie peut lever l'ambigüité terminologique sur les différents concepts de la norme et offre une représentation formelle de son vocabulaire. Enfin, elle peut offrir un moyen d'analyse, de vérification et de validation d'un projet.

Mots clés: Ingénierie ontologique, MDA, procédés logiciels, transformation de modèle, OWL,UML,ISO/IEC 24744

Abstract

The Semantic Web is the main direction of future development of the Web. Ontologies are the most important part of the Semantic Web applications. AI techniques are most used for the creation of ontology, but these techniques are more related to research then the general public. During the last decade, there have been many proposals to use software engineering techniques, particularly the UML for creating ontology as it is the most accepted software engineering standard, to provide a process development of ontology closer to a large population of practitioners. However, UML is based on an object-oriented paradigm, and has a certain limit in the development of ontology. These limitations can be overcome by using the UML extensions (v. UML profiles), as well as standards such as OMG Model Driven Architecture (MDA), ontology definition metamodel (MDGs). Currently, the joint use of MDA and the semantic web has reached a certain maturity and a great success in the field of engineering of ontologies.

This thesis is a contribution to the field of software processes (SP). We propose the development of an ontology based on ISO / IEC 24744 (Software Engineering Metamodel for Development Methodologies-- SEMDM). The ontology is generated from the meta model described in the standard using a MDA based transformation process. It describes, among other, the activities and tasks to be executed, artifacts to manipulate (create, use or edit) and the people involved. This ontology will help to systematize and better control the use of the model of the ISO / 24744 standard. The ontology may lift the terminological ambiguity about the different concepts of the standard and provides a formal representation of its vocabulary. Also the ontology can provide a means of analysis, verification and validation of a project.

Keywords : Ontology engineering, MDA, Software Process, Model transformation, OWL,UML,ISO/IEC 24744

ملخص

والويب الدلالي هو الاتجاه الرئيسي للتنمية المستقبلية للويب. أونتولوجيز هي أهم جزء من تطبيقات الويب الدلالي. وتعتبر تقنيات الذكاء الاصطناعي الأكثر استخداما لإنشاء الأنطولوجيا، ولكن هذه التقنيات هي أكثر ارتباطا بمخابر البحث من الجمهور العام. خلال العقد الماضي، كانت هناك العديد من المقترحات لاستخدام تقنيات هندسة البرمجيات، ولا سيما أومل لإنشاء الأنطولوجيا لكونه المعيار الهندسي البرمجيات الأكثر قبولا، لتوفير عملية تطوير الأنطولوجيا أقرب إلى عدد كبير من الممارسين.

ومع ذلك، يستند أومل على نموذج ألأشياء، وهذا يعتبر عائق في تطوير علم الأنطولوجيا. يمكن التغلب على هذه القيود باستخدام ملحقات أومل) ، فضلا عن معايير مثل أومج (مدا)، أونتولوجي تعريف ميتاموديل (مدجس).

حاليا، فإن الاستخدام المشترك لل مدا والويب الدلالي قد وصلت إلى نضج معين ونجاحا كبيرا في مجال الهندسة من أونتولوجيز.

هذه الأطروحة هي مساهمة في مجال عمليات البرمجيات .نقترح تطوير الأنطولوجيا على أساس إسو / إيك 24744 (هندسة البرمجيات ميتاموديل من أجل التنمية منهجيات-- سيمدم). يتم إنشاء الأنطولوجيا من نموذج ميتا الموصوف في المعيار باستخدام عملية التحول القائمة على مدا. وهو يصف، من بين أمور أخرى، الأنشطة والمهام التي يتعين تنفيذها، والتحف للتلاعب (إنشاء أو استخدام أو تحرير) والأشخاص المعنيين.

الكلمات المفتاحية:

هندسة الانتولوجية، طريقة الأمدا،النمادج التشكلية،تحويل النمادج،أومال،أودبليوأل، إسو / إيك 24744

1. Chapitre I Introduction générale

1.1 Contexte

L'ingénierie des procédés logiciels (Software Process Engineering) est une discipline du Génie Logiciel qui ambitionne la maîtrise du développement de logiciels en mettant l'accent sur le support et le contrôle du processus de développement. Le processus de développement logiciel joue un rôle déterminant dans la valeur des produits logiciels. L'ingénierie des procédés développés des méthodes et des techniques pour modéliser, assister, évaluer et améliorer les processus de développement.

On peut dire que l'ingénierie des procédés se consacre au développement d'un produit particulier : le procédé de développement. De la même façon que le développement du logiciel, l'ingénierie des procédés est confrontée à un double défi à savoir la qualité et la productivité. Cependant, pour l'ingénierie des procédés logiciels, ces défis sont particulièrement difficiles à relever car les procédés de développement de logiciel sont des produits fondamentalement complexes visant la gestion de la qualité, la gestion de configuration et la gestion de projet et peuvent être mis en œuvre de manière répartie, coopérative, itérative, par différents types d'agents, avec différentes contraintes de performance, et nécessitant diverses ressources matérielles ou humaines.

La description d'un procédé logiciel est appelée habituellement modèle de procédé. Un modèle de procédé ce contente de décrire les taches à effectués, mais aussi les types de produits et documents qui doivent être créés ou manipulés, les rôles des acteurs impliquées dans le développement logiciel, les outils à utiliser, etc.

Dans cette quête permanente de la qualité des procédés logiciels plusieurs modèles de procédés logiciels (software process model) ont vu le jour intégrant le changement continue de méthodes et de pratiques de développement, inspirant de nouvelles méthodes et processus tout en s'adaptant au progrès rapide des technologies et des outils.

La plupart de ces travaux sont centrés sur la capitalisation d'expériences de construction de procédés qui sont généralement représentées de façon informelle et dont la réutilisation est faite de façon manuelle. Dans un contexte aussi complexe où il existe plusieurs intervenant et

où le manque de compréhension entre les différents intervenants peut mettre en péril un projet. L'utilisation des ontologies peut s'avérer comme une bonne solution pour faire face à ces types de problèmes.

Comme l'avançait déjà Uschold dans (Uschold et Gruninger, 1996), les ontologies ont trois grandes catégories d'usages: la communication, l'interopérabilité et l'ingénierie des systèmes : où elles sont utilisées pour assister les processus de conception et de maintenance des systèmes logiciels, qu'ils soient basés sur les connaissances ou non.

Les ontologies sont intéressantes pour faire face aux problèmes d'interopérabilité et d'échange et s'avèrent être bien adaptées pour la conception et la modélisation des systèmes complexes. L'ontologie est une branche de la philosophie qui aborde l'étude de « ce qui est » (Smith, 2003.). Aristote l'employait pour décrire l'existence des êtres, d'autre part le domaine de l'intelligence artificielle aborde les modèles du monde et se l'est donc approprié en utilisant ce même terme pour décrire ce qui peut être représenté du monde réel dans un programme(Studer, 1998) afin de faciliter le partage et la réutilisation de connaissances.

Autre que la communauté des chercheurs en Intelligence artificielle, les ontologies intéressent beaucoup de domaines relatifs à la connaissance. On retrouve donc un grand intérêt aux ontologies dans les systèmes d'information (SI) et ceux à base de connaissances (KBS pour Knowledge Based Systems). Ces derniers, s'intéressant aux connaissances de domaines , à la résolution des problèmes de connaissances et trouvent dans les ontologies un moyen permettant l'analyse, la modélisation et l'implémentation des domaines de connaissances (Studer, 1998. Chandrasekaran, Josephson et al, 1999).

L'aspect consensuel d'une ontologie n'est pas anodin. Toute ontologie doit être le résultat d'un processus où tous les acteurs du domaine coopèrent et s'entendent sur le sens à donner à chacun des concepts relatifs au domaine, de sorte à ce que l'ontologie puisse être adoptée par tous et servir de base commune à la connaissance du domaine.

Dans le domaine du génie logiciel, la norme ISO/IEC 24744 (ISO/IEC, 2007) a été fortement inspirée des travaux de Henderson-Sellers sur le méta-modèle OPF (OPEN Process Framework) (Henderson-Sellers et Gonzalez-Perez, 2005). Ces travaux ont émergé suite à une forte critique des premières versions de SPEM (Software Process Engineering Metamodel) (OMG, 2008), notamment pour les mécanismes de passage d'un niveau de modélisation à l'autre (ex. de M2 vers M1 vers M0)(Atkinson and Kühne, 2001. Atkinson, Weeks et al,

2004). Pour répondre aux limites de SPEM, le méta-modèle normalisé ISO/IEC 24744 présente une nouvelle approche de modélisation basée sur trois aspects, à savoir : processus, produits et producteurs du travail. Ce dernier implique deux genres d'utilisateurs à savoir les développeurs et les ingénieurs méthode.

1.2 Problématique

Bien que la norme ISO/IEC 24744 présente plusieurs avantages, elle peut sembler intimidante vue l'adoption de nouveaux concepts qui constituent un changement de paradigme qui implique une nouvelle façon de penser et de percevoir le processus de méta-modélisation. En plus, la norme ISO/IEC24744 regroupe trois grandes familles d'utilisateurs et chaque famille contient aussi un groupe d'intervenant humain et logiciel. Ce qui pose un problème de communication entre les différents intervenants dans un projet.

Un autre problème est que la norme ISO/IEC 24744 est définie à travers son méta modèle, qui est lui même défini par des diagrammes de classe UML. Malheureusement, UML n'est qu'un langage semi-formel et ses capacités de formalisme sont très limitées d'une part, et d'autre part difficilement exploitables par des machines.

Notre thèse combine les avancées de deux domaines de recherche qui prêchent la réutilisation à large échelle à savoir : les ontologies et les procédés logiciels.

1.3 Objectifs de la recherche

Pour cette recherche nous avons fixé deux objectifs : un objectif principal et un objectif secondaire ou intermédiaire et exploratoire.

L'objectif principal de la recherche consiste à développer une ontologie de domaine, basée sur la norme ISO/IEC 24744, qui intègre les phases à suivre, les activités à exécuter, les artefacts à gérer et les acteurs impliqués dans un projet. Nous croyons que la définition d'une telle ontologie aidera à systématiser et à mieux maitriser l'utilisation du modèle de la norme ISO/24744. L'ontologie peut lever l'ambigüité terminologique sur les différents concepts de la norme et offre une représentation formelle de son vocabulaire. Aussi les techniques d'ontologie peuvent offrir un moyen d'analyse, de vérification et de validation d'un projet.

Notre approche recommande l'utilisation des techniques de méta-modélisation et de transformation de modèles. Elle propose alors de combiner à la fois les ontologies, les procédés logiciels et l'approche MDA(OMG, 2003). L'approche MDA nous permet de

modéliser l'ontologie pour l'élaboration de différents méta-modèles correspondants aux langages de représentation d'ontologies tels que OWL, RDF, RDFS.

En effet, l'association conjointe de ces trois approches nous permet effectivement :

i) d'avoir un niveau d'abstraction très élevé

ii) de prendre en compte l'aspect sémantique des différents éléments des processus (activités, tâches,… etc.) par l'intégration du concept d'ontologie

Ceci débouche sur un objectif secondaire de la recherche qui porte sur la création d'un profil UML pour le méta modèle de la norme ISO/IEC 24744. Durant notre thèse, nous avons décidé d'utiliser un processus MDA pour la création de notre ontologie. Nous nous sommes trouvés dans l'obligation de créer un profil UML qui permet de s'appuyer sur les mécanismes d'extension du formalisme UML, notamment les stéréotypes, sans remettre en cause le métamodèle UML existant, ce qui ouvre la voie au support des principaux outils UML du marché.

L'application des stéréotypes sur un modèle rend possible l'interprétation de ces stéréotypes par l'intermédiaire de programmes informatiques. Ainsi, l'implémentation de logiciels peut être dirigée par l'utilisation de stéréotypes et de notations communes à travers des outils de modélisation UML.

Ces stéréotypes permettent de distinguer et de spécialiser les éléments de modèles en leur donnant une sémantique particulière.

1.4 Plan du mémoire :

Cette recherche s'articule en cinq chapitres auxquels viennent s'ajouter cette introduction et une conclusion. En introduction nous avons déjà présenté la problématique et les contributions.

Dans les chapitres deux, trois et quatre nous introduisons les principaux concepts en relation avec le contexte scientifique dans lequel se déroule ce travail à savoir (les ontologies, les procédés logiciels et l'ingénierie dirigée par les modèles). Nous commençons par étudier en détails la notion d'ontologie, sa construction et son utilisation, ainsi que les différents langages et outils permettant sa manipulation.

Le troisième chapitre introduit les notions de base du domaine des procédés logiciels. Il détaille les concepts de base des procédés logiciels en se focalisant sur leur modélisation et

métamodélisation. Ce chapitre permet d'identifier les caractéristiques intrinsèques des modèles de procédés logiciels.

Nous présentons dans le quatrième chapitre le domaine de l'Ingénierie Dirigée par les Modèles. Nous introduisons ensuite différentes approches de l'IDM, dont l'approche Model Driven Architecture de l'OMG

La revue de littérature exposée dans le chapitre cinq vise à présenter les différents Travaux de recherches effectués dans le domaine des ontologies conçu pour les procédés logiciels.

Dans le sixième chapitre, nous présentons l'architecture MDA de notre approche dont le but est de construire une ontologie de domaine pour la norme ISO/IEC 24744 à partir de son méta modèle. De ce fait, nous commençons par décrire notre approche d'une manière globale. Ensuite, nous détaillons les différents modèles qui la construisent et la manière avec laquelle ils sont pris en compte pour concevoir et générer notre ontologie. Finalement, afin de valider cette approche et de montrer son intérêt fonctionnel, nous sommes tenus de fournir l'ensemble des outils permettant de la mettre en pratique. Pour chaque étape de notre approche, nous avons choisi les technologies et les outils existants, qui conviennent le mieux à nos besoins

Ce mémoire se termine par une conclusion portant à la fois sur le bilan de notre recherche, sur l'ensemble des contributions apportées par cette thèse et finalement ses limites qui représentent les perspectives tracées pour la suite de ce travail.

2 Chapitre 2 Les ontologies

2.1 Introduction:

L'ingénierie est la profession impliquée dans la conception, la fabrication, la constitution, et la maintenance des produits, des systèmes, et des structures. A un niveau plus élevé, il y a deux types d'ingénierie : l'ingénierie directe et la retro- ingénierie.

L'ingénierie ontologique a plusieurs buts parmi eux :

- Partager la compréhension commune de la structure d'information (Parmi des personnels et parmi des agents logiciels).
- Permettre la réutilisation de la connaissance de domaine (Pour éviter de "réinventer la roue", pour présenter des normes qui permettent l'interopérabilité).
- Rendre des suppositions de domaine explicites.
  - Faciliter le changement des suppositions sur des domaines (considérer une base de connaissance génétique)
  - Faciliter la compréhension et la mise à jour des données léguées.
- Séparer la connaissance de domaine de la connaissance opérationnelle (réutiliser le domaine et la connaissance opérationnelle)

Avant d'expliciter la notion d'ontologie à travers la description du processus conduisant à son élaboration, il est nécessaire de rappeler sa définition, quelque soit les éléments constitutifs, ses différents types et leur cycle de vie.

2.2 Définition d'une ontologie:

Le terme ontologie est issu du domaine de la philosophie, où il signifie « Explication systématique de l'existence ».

Dans le cadre de 1'intelligence artificielle, Neches et ses collègues(Neches. Fikes et al, 1991) furent les premiers à en proposer une définition, à savoir : **« une ontologie définit les termes et les relations de base du vocabulaire d'un domaine ainsi que les règles qui**

indiquent comment combiner les termes et les relations de façon à pouvoir étendre le vocabulaire ». Cette définition nous dit comment élaborer une ontologie, en nous offrant des directives relativement floues: repérer les termes de base et les relations entre les termes, identifier les règles servant à les combiner, fournir des définitions de ces termes et de ces relations.

Notons que d'après cette définition, une ontologie inclue non seulement les termes qui y sont explicitement définis, mais aussi les termes qui peuvent être créés par déduction en utilisant les règles.

En 1993, Gruber (Gruber, 1993) formule la définition suivante, à savoir « **une ontologie est une spécification explicite d'une conceptualisation »,** qui deviendra célèbre et restera la définition la plus citée dans la littérature scientifique.

En 1997, Borst (Borst, 1997) apporte une légère modification à la définition de Gruber en précisant que **« les ontologies se définissent comme une spécification formelle d'une conceptualisation commune».**

Studer et ses collègues (Studer, 1998) ont donné l'interprétation suivante de ces deux définitions :

« La conceptualisation renvoie à un modèle abstrait d'un quelconque phénomène après en avoir relève les concepts significatifs».

Par explicite, il faut entendre que le type de concepts utilises, ainsi que leurs contraintes d'utilisation, sont définis de façon explicite; quant à l'adjectif *formel*, il exprime le fait que 1'ontologie doit être lisible par ordinateur.

Comme un renvoie à l'idée qu'une ontologie rend compte d'un savoir consensuel, c'est à- dire qu'elle n'est pas 1'objet d'un individu, mais qu'elle est reconnue par un groupe.

De nombreuses définitions ont été appliquées à l'ontologie après celle de Gruber. En 1995, Guarino et Giaretta (Giaretta et Guarino, 1995) ont recueilli sept définitions dans la littérature scientifique et en ont fourni des interprétations sémantiques.

D'autres auteurs offrent des définitions fondées sur l'approche qu'ils ont adoptée pour construire leurs ontologies, d'après Gomez-Perez (Gómez-Pérez, Fernández-López et al, 2003) la définition de l'ontologie est donc la suivante:

« Une ontologie fournit les moyens de décrire de façon explicite la conceptualisation des connaissances représentées dans une base de connaissances».

2.3 Composants d'une ontologie:

Les connaissances traduites par une ontologie sont à véhiculer à l'aide des éléments suivants : (Gómez-Pérez, 1999):

1) *Concepts*; 2) *Relations*; 3) *Fonctions*; 4) *Axiomes*; 5) *Instances*.

- Les concepts, aussi appelés termes ou classes de l'ontologie, correspondent aux abstractions *pertinentes* d'un segment de la réalité (le domaine du problème), retenues en fonction des objectifs qu'on se donne et de l'application envisagée pour l'ontologie.

Un concept peut être divisé en trois parties : un terme (ou plusieurs), une notion et un ensemble d'objets.

La notion, également appelée *intension* du concept, contient la sémantique du concept, exprimée en termes de propriétés et d'attributs, de règles et de contraintes.

L'ensemble d'objets, également appelé *extension* du concept, regroupe les objets manipulés à travers le concept, qui sont appelés instances du concept.

Par exemple, le terme « table » renvoie à la fois à la notion de table comme objet de type « meuble » possédant un plateau et des pieds, et l'ensemble des objets de ce type. (Fürst, 2004)

- Les relations traduisent les associations (pertinentes) existantes entre les concepts présents dans le segment analysé de la réalité. Ces relations incluent les associations suivantes:

1- Sous-classe-de (généralisation - spécialisation) ;
2- Partie-de (agrégation ou composition) ;
3- Associée-à ;
4- Instance-de, etc. (Psyché, 2003)

Ces relations nous permettent d'apercevoir la structuration et l'interrelation des concepts, les uns par rapport aux autres. Elles sont caractérisées par un terme (voir plusieurs) et une signature qui précise le nombre d'instances de concepts que la relation lie, leurs types et 1'ordre des concepts, c'est-a-dire la façon dont la relation doit être lue. Par exemple, la relation « écrit » lie une instance du concept « personne » et une instance du concept « texte », dans cet ordre. (Fürst, 2004)

- Les fonctions constituent des cas particuliers de relations, dans lesquels un élément de la relation, le nième (extrant) est défini en fonction des n-l éléments précédents

(intrants).

- Les axiomes constituent des assertions, acceptées comme vraies, à propos des abstractions du domaine traduites par I' ontologie.
- Les instances constituent la définition extensionnelle de l'ontologie. Ces objets véhiculent les connaissances (statiques, factuelles) à propos du domaine du problème. (Psyché, 2003).

2.4 Différents types d'ontologies:

Cette section n'a pas 1'ambition de fournir une typologique exhaustive des ontologies telle que celles de van Heijst(Van Heijst, Schreiber et al. 1997) et Mizoguchi(Lee and Mizoguchi 1995).

Elle présente néanmoins les types d'ontologies les plus couramment utilisés afin de permettre au lecteur d'avoir une idée des connaissances à inclure dans chaque type d'ontologie.

En gros, on identifie les catégories suivantes : les ontologies de représentation des connaissances, les ontologies de domaine, les ontologies de tâches, les ontologies de domaine-tâche, les ontologies d'application, les ontologies d'index, les ontologies interactives, etc.

- **Les ontologies de représentation de connaissances (Van Heijst, Schreiber et al. 1997)** regroupent les primitives de représentation utilisées afin de formaliser les connaissances selon des paradigmes de représentation des connaissances. L'exemple le plus représentatif de ce type d'ontologie est la *Frame-Ontology* (Gruber 1993), qui rassemble les primitives de représentation (classes, instances, cases, facettes, etc.) utilisées dans les langages à base de frames.
- **Les ontologies générales/communes (Lee et Mizoguchi 1995)** Incluent le vocabulaire lié aux objets, aux événements, au temps, à l'espace, à la causalité, au comportement, à la fonction, etc.
- **Les ontologies de domaine (Lee and Mizoguchi 1995, Van Heijst, Schreiber et al. 1997)** sont réutilisables dans un domaine donné. Elles fournissent le vocabulaire des concepts d'un domaine (par ex., scalpel, scanner dans un domaine médical), les relations entre ces derniers et les activités de ce domaine (par ex., anesthésié,

accoucher) ainsi que les théories et principes de base de ce domaine.

- **Les ontologies de tache(Lee et Mizoguchi 1995)** fournissent un vocabulaire systématisé des termes utilises pour résoudre les problèmes associés à des tâches qui peuvent appartenir ou non à un même domaine. Ces ontologies fournissent un ensemble de termes au moyen desquels on peut décrire au niveau générique comment résoudre un type de problème. Elles incluent des noms génériques (par ex., plan, objectif, contrainte), des verbes génériques (par ex., assigner, classer, sélectionner), des adjectifs génériques (par ex., assigné) et d'autres mots qui relèvent de I' établissement d'échéances.

- **Les ontologies de domaine-tâche** sont des ontologies de tâches réutilisables dans un domaine donné. Une ontologie domaine-tâche dans le domaine médical pourrait inclure les termes liés au timing d'une intervention chirurgicale : planifier - intervention chirurgicale.

- **Les ontologies d'application (Van Heijst, Schreiber et al. 1997)** contiennent suffisamment de connaissances pour structurer un domaine particulier.

Les ontologies de domaine et les ontologies d'application saisissent les connaissances statiques indépendamment de la façon dont on résout les problèmes alors que les ontologies de tâches et les ontologies domaine-tâche sont axées sur les connaissances visant à résoudre des problèmes.

Tous ces types d'ontologie peuvent titre combinés de façon à construire une nouvelle ontologie. Si l'on applique le problème du compromis entre l'utilisabilité et la réutilisabilité au domaine de 1'ontologie, on peut affirmer que plus une ontologie est réutilisable, moins elle est utilisable, et inversement.(Gómez-Pérez, Fernández-López et al. 2003).

2.5 Principes de construction d'une ontologie

Il existe un ensemble de critères et de principes qui ont fait leurs preuves dans le développement des ontologies et qui peuvent être résumes comme suit :

- **Clarté et Objectivité** (Gruber 1993): L'ontologie doit fournir la signification des termes définis en fournissant des définitions objectives ainsi qu'une documentation en langage naturel.

- **Complétude** (Gruber 1993): Une définition exprimée par des conditions nécessaires et suffisantes est préférée à une définition partielle (définie seulement par une condition

nécessaire et suffisante).

- **Cohérence (Gruber 1993):** Une ontologie cohérente doit permettre des inférences conformes à ces définitions.
- **Extensibilité monotonique maximale** (Gruber 1993) : De nouveaux termes généraux et spécialisés devraient être inclus dans I' ontologie d'une façon qui n'exige pas la révision des définitions existantes.
- **Engagements ontologiques minimaux** (Gruber 1993): L'ontologie devrait spécifier le moins possible la signification de ses termes tout en donnant aux parties qui s'engagent dans cette ontologie la liberté de spécialiser et d'instancier l'ontologie comme elles le désirent.
- **Principe de distinction ontologique** (Borgo, Guarino et al. 1996): les classes dans une ontologie devraient être disjointes.
- **Diversification des hiérarchies** (Arpirez, Gómez-Pérez et al. 1998): Ce principe est adopté pour augmenter la puissance fournie par les mécanismes d'héritage multiple.
- **Distance sémantique minimale** (Arpirez, Gómez-Pérez et al. 1998). Il s'agit de la distance minimale entre les concepts enfants de mêmes parents.

Les concepts similaires sont groupés et représentés comme des sous-classes d'une classe, considérant que les concepts qui sont moins similaires sont représentes plus loin dans la hiérarchie.

- **Normaliser les noms** (Arpirez, Gómez-Pérez et al. 1998). Ce principe indique qu'il est préférable de normaliser les noms autant que possible.

Cet ensemble de critères et de processus est généralement accepté pour guider le processus d'ingénierie ontologique.(Psyché 2003).

2.6 Cycle de vie d'une ontologie :

Les ontologies étant destinées à titre utilisées comme des composants logiciels dans des systèmes répondant à des objectifs opérationnels différents, leur développement doit s'appuyer sur les mêmes principes que ceux appliqués en génie logiciel. En particulier, les ontologies doivent être considérées comme des objets techniques évolutifs et possédants un cycle de vie qui nécessite d'être spécifié.

Les activités liées aux ontologies sont, d'une part, des activités de gestion de projet (planification, contrôle, assurance qualité), d'autre part, des activités de développement (spécification, conceptualisation, formalisation) et s'y ajoutent des activités transversales de support telles que 1'évaluation, la documentation, la gestion de la configuration.

Un cycle de vie inspiré du génie logiciel est proposé dans(Blázquez, Fernández et al. 1998).

Il comprend une étape initiale d'évaluation des besoins, une étape de construction, une étape de diffusion et une étape d'utilisation.

Après chaque utilisation significative, I 'ontologie et les besoins sont réévalués. L'ontologie peut être étendue et, si nécessaire, en partie reconstruite.

La phase de construction d'une ontologie opérationnelle peut être décomposée en 3 parties:

1. *la conceptualisation*, qui conduit, à partir d'un corpus, à l'élaboration d'un modèle conceptuel, informel ou semi-formel, identifiant les connaissances du domaine à travers des concepts manipulés et de leur sémantique ;

2. *l'ontologisation*, qui conduit du modèle conceptuel à l'ontologie proprement dite est représentation formelle des connaissances du domaine aussi indépendante que possible des objectifs applicatifs ;

3. *l'opérationnalisation*, qui conduit de l'ontologie à une ontologie opérationnelle, représentation formelle des connaissances du domaine adaptée à une application particulière.

L'étape d'ontologisation peut être complétée d'une étape d'intégration au cours de laquelle une ou plusieurs ontologies vont être importées. Dans l'ontologie à construire (Fernández-López, Gómez-Pérez et al. 1997). M. FERNANDEZ insiste sur le fait que les activités de documentation et d'évaluation sont nécessaires à chaque étape du processus de construction et l'évaluation précoce permettant de limiter la propagation d'erreurs.

Le processus de construction peut être intégré au cycle de vie d'une ontologie comme indiqué sur la figure (2.1) suivante :

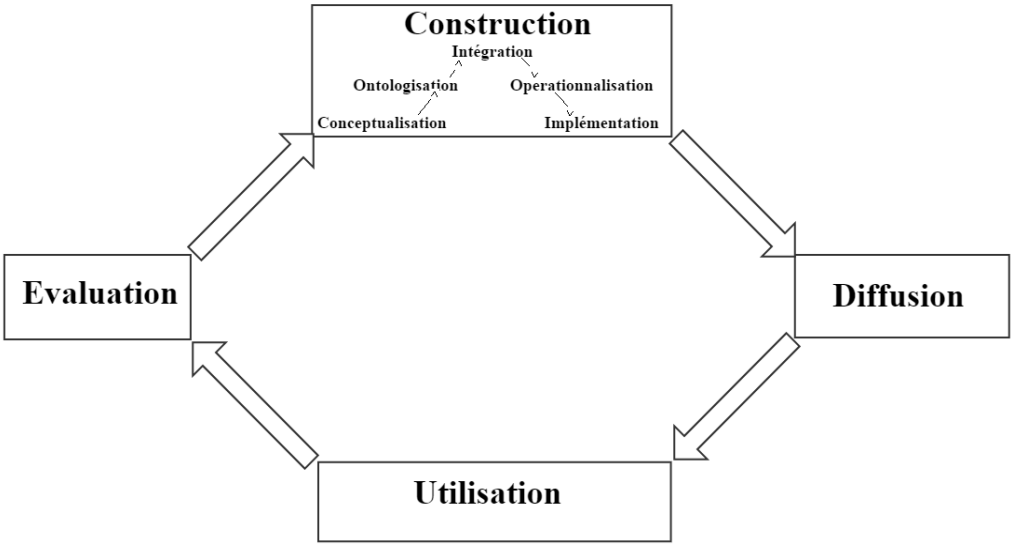

Figure 2.1 *Le cycle de vie d'une ontologie.*

Bien qu'aucune méthodologie générale n'ait pour l'instant réussi à s'imposer, de nombreux principes et critères de construction d'ontologies ont été proposés.

C'est le cas de METHONTOLOGY, élaborée en 1997 par Fernandez, qui couvre tout le cycle de vie d'une ontologie(Fernández-López, Gómez-Pérez et al. 1997). M. USCHOLD et M. KING ont également propose une méthodologie générale, inspirée de leur expérience de construction d'ontologies dans le domaine de la gestion des entreprises(Uschold and King 1995).

La méthodologie présentée par M. GRUNINGER et M.S. FOX dans(Grüninger and Fox 1995) est elle aussi issue d'une expérience de construction d'ontologie sur ce domaine.

La méthodologie ONTOSPEC de G.KASSEL(Kassel 2002) constitue une aide à la structuration des hiérarchies de concepts et de relations durant la phase d'ontologisation.

C'est également le cas des principes différentiels énoncés par B. BACHIMONT(Bachimont, Isaac et al. 2002), et des critères de classification des propriétés proposées par N. GUARINO et C. WELTY. Mais quelque soit la méthodologie adoptée, le processus de construction d'une ontologie est une collaboration qui réunit des experts du domaine de connaissance et des ingénieurs de la connaissance, voir les futurs utilisateurs de I 'ontologie (Guarino and Welty 2000).

Dans tous les cas, 1'élaboration d'une ontologie est basée sur des ressources linguistiques et cognitives constituant un corpus.

2.6.1 La constitution d'un corpus :

La construction d'une ontologie suppose au minimum que soit et délimite aussi précisément que possible le domaine de connaissance à modéliser, au besoin en le découpant en termes de connaissances du domaine, connaissances de raisonnement, connaissances de haut niveau (communes à plusieurs domaines).

Délimiter un domaine repose sur l'utilisation de ressources textuelles et/ou multimédia, constituant le corpus du domaine, et à travers lesquelles peuvent être appréhendées la terminologie du domaine et les significations des concepts.

Le point de vue actuel est qu'un domaine n 'a de sens que lorsqu'il est défini par un corpus et une application en vue de laquelle l' étude terminologique est effectuée. En effet, un domaine n'est pas seulement défini par le champ de connaissance qu'il couvre, mais aussi par le point de vue sous lequel les utilisateurs de l'ontologie considèrent ce champ de connaissance. Ainsi, le corpus constitué pour construire une ontologie des connaissances médicales ne sera pas le même si le public visé est constitué de médecins ou si le public visé n'a pas de connaissances particulières dans ce domaine.

Une fois le corpus constitué, la phase de conceptualisation du processus de construction de I' ontologie peut débuter.

2.6.2 La conceptualisation:

La conceptualisation consiste à identifier, dans un corpus, les connaissances du domaine. Cette conceptualisation peut se décomposer en deux étapes.

Tout d'abord, il faut faire le tri entre connaissances spécifiques au domaine et celles qui, bien que présentes dans le corpus, ne participent qu'a 1'expression des connaissances du domaine. La nature conceptuelle (concepts, relations, propriétés des concepts et relations, axiomes) des connaissances ainsi extraites du corpus doit ensuite être précisée. Des choix liés aux contextes d'usage de l'ontologie doivent donc être effectués dans cette étape.

La découverte des connaissances d'un domaine peut s'appuyer à la fois sur l'analyse de documents et sur l'interview d'experts du domaine.

Ces activités doivent être raffinées au fur et à mesure que la conceptualisation émerge.

L'analyse de corpus ne peut suffire à elle seule à spécifier la sémantique du domaine. En effet, l'écrit n'est qu'un support, et les connaissances qui y sont représentées ne prennent sens que lorsqu'elles sont lues par un expert, le terme expert désignant justement ici une personne pour qui le corpus fait sens. La sémantique doit donc être précisée par un expert du

domaine qui, soit participe directement à la construction de I'ontologie, soit fournit des spécifications conceptuelles permettant de contraindre la sémantique exprimée.

Dans cette optique, l'utilisation de *questions de compétence*, c'est-à-dire de questions auxquelles le système projeté est censé pouvoir répondre, est préconisée par M. GRUNINGER et M.S. FOX. Exprimées tout d'abord en langage naturel, ces questions servent à la conceptualisation, du fait que le vocabulaire du domaine et sa sémantique peuvent en être extraits. Elles peuvent ensuite être formalisées et utilisées pour valider 1'ontologie construite une fois celle-ci intégrée dans un SBC.(Grüninger and Fox 1995)

L'analyse d'un ensemble de documents, ou l'interview de plusieurs experts peut facilement mener à des différences significatives voire à des contradictions au niveau du sens prêt aux concepts ou aux relations. Il convient donc de restreindre au maximum les possibilités d'interprétation des termes et propriétés utilisés. Afin d'objectiver les connaissances, une normalisation sémantique est nécessaire, normalisation qui doit être le fruit d'un dialogue entre experts.(Bachimont, Isaac et al. 2002)

Certaines connaissances implicitement utilisées dans le domaine ne sont cependant jamais exprimées, ni dans le corpus, ni par les experts.

Une fois les concepts et relations identifiés par leurs termes, il faut en décrire la sémantique en indiquant, à priori en langage naturel, leurs instances connues, les liens qu'ils entretiennent entre eux et leurs propriétés.

Le processus de conceptualisation mène ainsi à la construction d'un modèle conceptuel (ou conceptualisation), qui décrit, au travers des éléments terminologiques et sémantiques, les connaissances du domaine. Ce modèle conceptuel n'est cependant pas formel, il peut être complètement informel, exprimé en langage naturel ou semi-formel, combinant langage naturel et propriétés formelles.

Une fois les ressources cognitives passées au travers du tamis de la conceptualisation, il convient donc, pour l'utiliser dans une machine, de formaliser le modèle conceptuel obtenu. C'est l'objet du processus d'ontologisation.

2.6.3 L'ontologisation:

L'ontologisation consiste à structurer et formaliser autant que possible la conceptualisation pour construire une ontologie spécifiant la terminologie et la sémantiques du domaine à travers un modèle doté d'une sémantique formelle (mais non opérationnelle).

Afin de respecter les objectifs généraux des ontologies, T. GRUBER (Gruber 1993) propose 5 critères permettant de guider le processus d'ontologisation :

- *La clarté et l'objectivité des définitions, qui doivent être indépendantes de tout choix d'implémentation ;*
- *La cohérence (consistance logique) des axiomes ;*
- *L'extensibilité d'une ontologie, c'est-a-dire la possibilité de l'étendre sans modification;*
- *La minimalité des postulats de formalisation, ce qui assure une bonne portabilité ;*
- *La minimalité du vocabulaire, c'est-a-dire I' expressivité maximum de chaque terme.*

M. USCHOLD (Uschold and Gruninger 1996) préconise de bâtir ces hiérarchies de bas en haut, c'est-à-dire en identifiant les concepts de bas puis en regroupant ces concepts à l'aide de concepts plus généraux. On donne ainsi la priorité aux concepts de bas niveau réellement utilisés dans le domaine, par rapport aux concepts qui ne sont souvent qu'ajouter artificiellement pour bâtir la hiérarchie. De plus, il faut bien voir que l'ontologisation est une traduction dans un certain formalisme de connaissances exprimées à priori en langage naturel.

Le respect de la sémantique du domaine doit être assuré par un engagement ontologique, notion proposée initialement par T. GRUBER comme un critère pour utiliser une spécification partagée d'un vocabulaire : **« We say that an agent commits to a knowledge level specification if its observable actions are logically consistent with the specification »** Pour T.GRUBER(Gruber 1993), un engagement ontologique est une garantie de cohérence entre une ontologie et un domaine, mais pas une garantie de complétude de l'ontologie.

B. BACHIMONT, dans (Bachimont 2000), distingue l'engagement sémantique qui, à travers des principes différentiels, permet de préciser le sens des concepts (concepts sémantiques) de manière non ambiguë, et l'engagement ontologique qui associe des extensions à des concepts (concepts formels).

Les quatre principes différentiels proposés par B. BACHIMONT mènent à une hiérarchie des concepts différentiels sous forme d'un arbre. En sus d'en guider la construction, ces principes permettent de vérifier la cohérence de la hiérarchie. Une fois bâtie l'ontologie différentielle, nous ajoutons l'ontologie référentielle, c'est-à-dire les extensions des concepts. Dans le cadre référentiel, une instance peut faire partie de l'extension de plusieurs concepts. Les arbres de concepts et de relations obtenus sont ensuite étiquetés avec les propriétés et attributs des primitives, ce qui permet de présenter de manière synthétique et structurée l'ensemble des connaissances identifiées (Guarino and Welty 2000).

Une fois le modèle conceptuel structuré, il faut l'exprimer dans un langage formel de représentation d'ontologies.

Le travail de structuration peut d'ailleurs être mené dans le cadre de tels langages, car ils offrent tous la possibilité de représenter des hiérarchies de concepts et de relations ou d'attributs. La sémantique de la subsomption n'est cependant pas toujours Conforme aux principes différentiels cités précédemment.

L'étape d'ontologisation conduit ainsi à la construction d'une ontologie mais celle-ci ne peut être utilisée telle quelle dans un SBC, du fait de 1'absence de sémantiques opérationnelles des représentations qu'elle contient.

2.6.4 L'opérationnalisation :

Décrire les connaissances en termes de concepts, de relations et de propriétés sur ces concepts et relations ne suffisent généralement pas pour atteindre 1'objectif opérationnel d'un SBC. *« De façon un peu caricaturale, on pourrait dire que ce ne sont pas les connaissances en elles--mêmes qui sont intéressantes [..] En fait. ce qui doit être le centre d'intérêt pour le modélisateur, c'est davantage leur mise en œuvre, leur concrétisation dans une action pour atteindre un but car c'est bien 1'activité qu'on cherche à assister et non les connaissances en elles-mêmes ». (Fürst 2004)*

En sus des paradigmes différentiels et référentiels, B. BACHIMONT définit le paradigme opérationnel d'une ontologie.

Au niveau opérationnel, les concepts et relations sont définis par les opérations pour qu'il soit possible de les appliquer. Dans cette optique, nous considérons qu'une représentation de connaissance n'est opérationnelle que si la façon dont cette représentation est utilisée pour raisonner est fixée.

La sémantique opérationnelle est plus spécifique que la sémantique formelle, car cette dernière ne fait que restreindre les interprétations possibles d'une représentation, alors que la première fixe, non seulement l'interprétation de la représentation, mais également l'usage qui est fait de cette représentation pour raisonner,

Nous utilisons donc ici le terme *opérationnel* à la fois pour caractériser un modèle de représentation doté de mécanismes de raisonnement et d'une sémantique opérationnelle, et pour caractériser tout langage exécutable qui implémente un tel modèle.

Une représentation opérationnelle est donc ici une représentation qui fixe les opérations qu'elle permet d'effectuer, qu'elle soit exprimée dans un langage exécutable sur machine ou pas.

L'utilisation opérationnelle d'une ontologie va donc nécessiter le passage par un processus d'opérationnalisation qui permettra, au delà de la spécification des connaissances au niveau conceptuel, de préciser la sémantique opérationnelle de la forme de l'ontologie qui sera utilisée dans un SBC. (Fürst 2004)

L'opérationnalisation consiste à outiller une ontologie pour permettre à une machine, via cette ontologie, de manipuler des connaissances du domaine. La machine doit donc pouvoir utiliser des mécanismes opérant sur les représentations de 1'ontologie.

Dans le cas où le langage d'ontologisation n'est pas opérationnel, il est nécessaire, soit d'outiller ce langage, dans la mesure du possible, soit de transcrire l'ontologie dans un langage opérationnel. Mais certains langages offrent des possibilités de raisonnements limites qui peuvent convenir à certaines applications limitées.

Finalement, l'ontologie opérationnalisée est intégrée en machine au sein d'un système manipulant le modèle de connaissances utilisé via le langage opérationnel choisi. Mais avant d'être livrée aux utilisateurs, 1'ontologie doit bien sûr être testée par rapport au contexte d'usage pour lequel elle a été bâtie. (Fürst 2002) .

2.6.5 L'évaluation d'une ontologie :

Deux niveaux peuvent être distingués dans 1'évaluation d'une ontologie :

- **La vérification,** qui consiste à s'assurer que 1'ontologie est conforme à un modèle formel de représentation de connaissances. Cette vérification porte sur des propriétés formelles qui ne peuvent être violées par 1'ontologie, sous peine de perdre son expressivité.
- **La validation**, qui consiste à s'assurer de la conformité sémantique de l'ontologie à un domaine de connaissance, c'est-à-dire que la sémantique exprimée dans 1'ontologie doit être celle du domaine considéré.

En d'autres termes, la vérification correspond à 1'exigence « *building the system right »*, relative à un modèle formel et à la validation correspondante à l'exigence « *building the right system »*, relative au domaine de connaissance modélise.

De plus, l'évaluation de 1'ontologie en amont de son opérationnalisation est souhaitable pour éviter de propager des erreurs, même si l'opérationnalisation peut être nécessaire pour

mener certaines activités d'évaluation. Ainsi, utiliser 1'ontologie pour répondre à des questions de compétence nécessite d'avoir opérationnalisé l'ontologie; le test de la cohérence d'une ontologie peut nécessiter des déductions pour mettre en évidence des contradictions logiques entre axiomes.

Cependant, la validité des hiérarchies de concepts et/ou de relations doit titre testée dès la phase d'ontologisation, aussi bien du point de vue formel que du point de vue sémantique.

2- La validation :

La validation d'une ontologie consiste à tester sa fidélité à la sémantique du domaine de connaissance considéré. La validation permet de tester la complétude de l'ontologie et la conformité de l'ontologie par rapport au domaine.

La complétude de 1'ontologie par rapport au domaine est assurée si toutes les connaissances du domaine sont présentes dans l'ontologie ;

La conformité de I'ontologie par rapport au domaine est assurée si les connaissances représentées dans I'ontologie correspondent exactement à la sémantique du domaine.

Le test d'une ontologie doit bien entendu précéder son utilisation au sein d'un SBC, mais est également utile à tous les niveaux de l'évolution d'une ontologie.

En particulier, si on désire augmenter la taille du domaine de connaissance modélisé, donc ajouter de nouvelles connaissances à l'ontologie, il faut s'assurer que les représentations déjà construites sont valides, d'autant plus qu'une ontologie va croître par agrégation de connaissances dans toutes les directions, et pas par ajout d'une couche de connaissances.

2.6.6 L'alignement et la fusion d'ontologies:

Utiliser les connaissances de différents domaines au niveau ontologique passe donc par l'utilisation de plusieurs ontologies couvrant chacune un des domaines visés, domaines à priori non disjoints puisque utilisés au sein du même système d'information. Les représentations considérées sont donc connexes et les utiliser de façon cohérente nécessite de déterminer les parties communes aux différentes ontologies. L'alignement de deux (ou plus) ontologies est nécessaire quand elles interviennent toutes deux dans un même SBC(Noy and Musen 2002). **L'alignement d'ontologies** consiste à trouver des correspondances entre les connaissances spécifiées dans les deux ontologies, de manière à pouvoir les exploiter conjointement dans le même système. En pratique, il s'agit d'identifier des concepts (ou des relations) de la première ontologie avec des concepts (ou des relations) de la seconde, ou de

trouver des liens conceptuels (subsomption, etc.) entre eux. Contrairement à l'alignement ou les deux ontologies de départ reste intactes, la **fusion** d'ontologies consiste, à partir de deux ontologies, à en créer une troisième qui intègre les connaissances spécifiées dans les deux premières.

Dans les deux cas, la connexité des deux domaines de connaissance modélisés par les ontologies est requise, sans quoi aucun lien ne peut être établi entre concepts. De plus, les formalismes de représentation d'ontologie utilisés doivent être aux mains compatibles, ainsi que les paradigmes conceptuels.

En pratique, l'alignement de deux ontologies se fera dans le cadre du même langage de représentation et pourra donc nécessiter des transcriptions préalables d'un langage à 1'autre.

L'uniformisation des modèles et formalismes de représentation est également nécessaire à la fusion d'ontologies. Préalablement à la fusion, il convient de déterminer quelle est l'ontologie la plus générale, ou celle qui est la plus étendue, c'est-à-dire celle qui ne sera pas modifiée. Les autres devront être alignées sémantiquement et syntaxiquement sur l'ontologie la plus générale. Le problème se ramène alors à l'intégration d'une ontologie dans une autre. Une fois les deux ontologies exprimées dans le même formalisme et à travers le même modèle cognitif, ces entités communes aux deux ontologies doivent être identifiées.

Les méthodes appliquées pour repérer les similarités entre concepts et/ou relations sont :

- Les méthodes terminologiques qui comparent les **labels** désignant deux concepts ou deux relations ;
- Les méthodes qui comparent les **propriétés internes** des concepts et relations (attribut des concepts, portée d'une relation, etc) ;
- Les méthodes qui comparent les propriétés externes des concepts et relations (subsomptions, relations entre concepts, etc) ;
- Les méthodes qui comparent les **extensions** des concepts et relations, c'est-à-dire leurs instances;
- Les méthodes qui comparent la **sémantique** des concepts et relations.

Ces méthodes peuvent bien entendu être combinées entre-elles. Elles peuvent parfois recourir à des ressources extérieures aux ontologies à aligner. Ainsi, les méthodes terminologiques peuvent faire appel à des ressources linguistiques (tables de synonymes ou/et d'hyperonymes) pour faciliter l'identification de liens entre concepts (c'est le cas dans l'outil de fusion d'ontologie ODEMerge intégré à l'éditeur d'ontologie ODE). Les méthodes

comparant les extensions des concepts et relations peuvent utiliser des corpus ou apparaissent les instances recherchées (c'est le cas de 1'outil FCA-Merge). Les correspondances ainsi établies entre entités conceptuelles ne sont pas forcément bijectives. Des conflits peuvent naître lors de cette « traduction au niveau sémantique », qui ne peuvent être résolus automatiquement. Si le degré de similarité ne permet pas de trancher entre deux correspondances possibles, l'intervention humaine est indispensable. D'autre part, si certaines entités de I'ontologie à intégré n'offrent de similarité avec aucune entité de 1'ontologie cible, il est tout de même nécessaire de leur trouver une entité subsumant dans l'ontologie cible. La différence de granularité entre les deux ontologies peut de plus entraîner la suppression de certaines entités, ou plus précisément leur agrégation au sein d'une même entité cible.

Plusieurs outils existants se basent sur une approche terminologique pour la mise en évidence d'analogies entre primitives conceptuelles, analogies qui sont ensuite raffinées par la comparaison de certaines propriétés structurant I'ontologie.

C'est par exemple le cas des outils SMART et PROMPT, développés pour l'atelier Protége2000, et qui comparent les attributs des concepts (slots et facets) pour affiner les analogies. Les analogies ainsi découvertes sont soumises à 1'utilisateur qui tranche entre les différentes possibilités. Chimaera est un outil d'alignement lié à 1'éditeur d'ontologie Ontolingua. Chimaera utilise à la fois les termes des concepts et certaines de leurs propriétés pour proposer à l'utilisateur des analogies que celui-ci doit valider ou rejeter.

Une étude détaillée des différents outils d'alignement d'ontologie montre cependant qu'aucun outil ne propose pour le moment une méthode d'alignement principalement basée sur les axiomes des ontologies et comparant la sémantique des concepts et relations pour déterminer des analogies entre eux.

2.7 Langages de spécification d'ontologies:

Plusieurs langages de spécification d'ontologies (ou langage d'ontologies) ont été développés pendant les dernières années, et ils deviendront surement des langages d'ontologie dans le contexte du Web sémantique. Certains d'entre eux sont basés sur la syntaxe de XML, tels que XOL (Ontology Exchange Language), SHOE (Simple HTML Ontology Extension - qui a ad précédemment base sur le HTML), OML (Ontology Markup Language), RDF (Resource Description Framework), RDF Schéma. Les 2 derniers sont des langages crées par des groupes de travail du World Wide Web Consortium (W3C). En conclusion, trois langages additionnels sont établis sur RDF(S) pour améliorer ses caractéristiques: OIL (Ontology

Inférence Layer), DAML+OIL et OWL (Web Ontology Language). La figure (2.2) ci-dessous présente des langages de spécification d'ontologie, qui ont été développés. Elle représente les rapports principaux entre tous ces langages sous la forme d'une pyramide des langages du Web sémantique.

RDF, langage dédie à l'expression d'assertions sur les relations entre objets, s'est heurté à la nécessité de définir les propriétés des classes dont ces objets sont en instances. Cependant, l'extension à RDFS ne fournit que des mécanismes très basiques pour spécifier ces classes, et il est donc moins expressif. Pour cela les chercheurs ont développé une série de langages pour la définition des ontologies comme le montre la figure 2.2, oú le OWL représente l'aboutissement de tous ces langages.

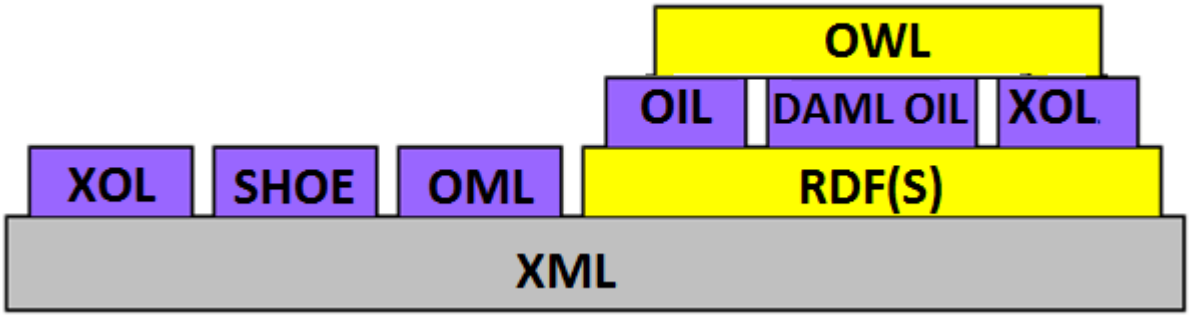

Figure 2.2 La pyramide des langages bases Web

OWL (Web Ontology Language) :

Le langage OWL, quant à lui, est dédié aux définitions de classes et de types de propriétés, et donc à la définition d'ontologies. Inspiré des logiques de descriptions (et successeur de DAML+OIL), il fournit un grand nombre de constructeurs permettant d'exprimer de façon très fine les propriétés des classes définies.

OWL est un langage formel qui reprend la syntaxe du RDF, ce dernier représente les connaissances sous forme de triplets (sujet, prédicat, objet) et chaque triplet sera considéré comme une déclaration RDF. RDF dissocie trois types d'objets: une "ressource" (resource) est définie par des "propriétés" (*properties*) ; L'association d'une ressource à une propriété par une valeur de propriété est une "declaration" (*statement*) (MCGUINNESS, Schreiber et al. 2004) .

Le rôle de OWL est d'augmenter l'expressivité de RDF en intégrant de nouveaux constructeurs exprimant des concepts ontologiques tels que les classes, les axiomes les contraintes, les relations de subsomption.

Mais cette expressivité n'est pas sans prix, puisqu'on perd en contre partie dans la complexité et la décidabilité les mécanismes d'inférences traitant le langage OWL, c'est pour cette raison qu'on a subdivisé le langage OWL en 03 sous langages qui différent dans la complexité et l'expressivité (Charlet, Laublet et al. 2003).

OWL LITE ne contient qu'un sous-ensemble réduit des constructeurs disponibles, mais son utilisation n'assure que la comparaison de types qui pourra être calculée (un problème de NP, donc « simple » en représentation de connaissances).

OWL DL contient l'ensemble des constructeurs, mais avec des contraintes particulières sur leur utilisation qui assurent la décidabilité de la comparaison de types. Par contre, la grande complexité de ce langage. OWL FULL, sans aucune contrainte, pour lequel le problème de comparaison de types est vraisemblablement indécidable.

La syntaxe d'un document OWL est donnée par celle des différents constructeurs utilisés dans ce document. Elle est le plus souvent donnée sous la forme de triplets RDF. A chaque constructeur est associée une sémantique, en théorie des modèles. Elle est directement issue des logiques de descriptions. La sémantique associée aux mots clés de OWL est plus précise que celle associée au document RDF représentant une ontologie OWL (elle permet plus de déductions).

2.8 Editeurs d'ontologies:

De nombreux outils permettent aujourd'hui de construire des ontologies. Parmi ceux-ci, quelques uns essaient de guider leur utilisateur dans l'élaboration de l'ontologie en suivant une méthodologie de conception plus ou moins complète, en respectant des principes de cycle de vie et validation logiciels et les principes de L'IC. Dans tous les cas, force est de constater qu'aucun de ces outils n'a réussi à s'imposer et la réflexion sur l'outillage de la construction des ontologies reste donc ouverte. Les outils proposés peuvent se regrouper grossièrement en deux catégories. Dans la première, on trouve les plus anciens historiquement, qui permettent de spécifier les ontologies au niveau symbolique (voir, par exemple, le serveur ONTOLINGUA(Farquhar, Fikes et al. 1995) qui permet la construction d'ontologies par réutilisation). Dans la seconde catégorie, les outils prennent mieux en compte l'importance du niveau des connaissances : ils proposent à leur utilisateur de créer l'ontologie de manière relativement indépendante de tout langage implémenté et prennent ensuite automatiquement en charge l'opérationnalisation de l'ontologie, en la transposant dans divers langages. Cette évolution tend à rapprocher les ontologies de leur but original : il semble en effet naturel de

chercher à s'abstraire - dans un premier temps - du niveau symbolique si on veut obtenir une ontologie permettant un réel partage d'une compréhension. Cette dernière catégorie regroupe les outils principalement utilisés aujourd'hui.

2.8.1 PROTEGE-2000:

PROTEGE-2000 (Noy and Musen 2000) est un environnement graphique de développement d'ontologies développé par le SMI de Stanford. Dans le modèle des connaissances de PROTEGE, les ontologies consistent en une hiérarchie de classes qui ont des attributs *(slots)*, qui peuvent eux-mêmes avoir certaines propriétés *(facets)*. L'édition des listes de ces trois types d'objets se fait par l'intermédiaire de l'interface graphique, sans avoir besoin d'exprimer ce que l'on a à spécifier dans un langage formel : il suffit juste de remplir les différents formulaires correspondant à ce que l'on veut spécifier.

PROTEGE2000 autorise la définition de méta-classes, dont les instances sont des classes, ('extraction d'ontologies à partir de sources textuelles, et la fusion semi-automatique d'ontologies. Il regroupe une communauté d'utilisateurs assez importante et constitue une référence pour beaucoup d'autres outils.

2.8.2 OILEd:

OILEd (Bechhofer, Horrocks et al. 2001), développé sous la responsabilité de I 'université de Manchester, a été conçu pour éditer des ontologies dans le langage de représentation OIL, un des précurseurs du langage OWL *(Ontology Web Language)* qui est aujourd'hui en voie d'être une recommandation W3C. Officiellement, il n'a pas d'autre ambition que de construire des exemples montrant les vertus du langage pour lequel il a été créé. A ce titre, OILEd est souvent considéré comme une simple interface d'un langage appartenant aux logiques de description. Néanmoins, il offre la plus grande partie de ce que l'on peut attendre d'un éditeur d'ontologies. On peut créer des hiérarchies de classes et spécialiser les rôles, et utiliser avec l'interface les types d'axiomes les plus courants. Cet éditeur offre également les services d'un raisonneur, FaCT, qui permet de tester la satisfiabilité des définitions de classes et de découvrir des subsomptions restées implicites dans l'ontologie.

2.8.3 OntoEdit :

Contrairement aux deux outils précédents, ONTOEDIT(Sure, Erdmann et al. 2002) n'est pas disponible gratuitement dans sa version complète. Il présente les fonctionnalités essentielles communes aux autres éditeurs (hiérarchie de concepts, expression d'axiomes, export de l'ontologie dans des langages divers) et a le mérite de s'appuyer sur une réflexion

méthodologique significative. La modélisation des axiomes a fait l'attention de soins particuliers pour pouvoir être effectuée- en tout cas pour les types les plus répandus - indépendamment d'un formalisme privilégié et cela pour faciliter la traduction d'un langage de représentation à un autre. Il propose également une gestion originale des *questionnaires de compétences*. Un programme appelé ontokick permet la spécification des questions de compétence pour lesquelles l'ontologie doit fournir des réponses.

2.8.4 Web ODE :

WebODE pour web Ontology Design Environment(Arpírez, Corcho et al. 2001), développé par le LAI de Madrid, est une plateforme de conception d'ontologies fonctionnant en ligne. D'un point de vue méthodologique, l'outil fait suite à ODE, ce dernier construit son ontologie dans un modèle de type frame, en spécifiant les concepts du domaine, les termes associés, les attributs et leurs valeurs et les relations de subsomption. L'ontologie opérationnelle est alors générée en utilisant les formalismes ONTOLINGUA ou FLOGIC.

Ces éditeurs assurent fidèlement le support de la méthodologie maison, en plus ils mettent L'accent sur la possibilité d'un travail collaboratif ou sur la mise à disposition d'outils complémentaires, comme un moteur d'inférences.

2.8.5 DOE :

Le dernier outil présenté ici est DOE pour Differential Ontology Editor(Bachimont, Isaac et al. 2002) Cet outil n'a pas pour ambition de concurrencer les grands environnements existants, mais plutôt de fournir un début d'implémentation à la méthodologie de structuration différentielle proposée par B. Bachimont. A l'instar des autres éditeurs, il offre une représentation graphique des arbres de concepts et des relations de l'ontologie et permet d'interagir avec les hiérarchies. L'outil assiste également la saisie des principes différentiels issus de la méthodologie en automatisant partiellement cette tâche. Le modèle de représentation de l'ontologie est finalement proche de celui du langage RDFS, à ceci prés qu'il autorise la modélisation de relations n-aires. Au niveau formel, l'éditeur est capable de faire quelques inférences en vérifiant la consistance de 1'ontologie (propagation de l’arité le long de la hiérarchie des relations et héritage des domaines par exemple).

2.9 Différentes applications de l'ontologie :

La notion d'ontologie, qui précède largement l'utilisation du mot, ne semble pas prête à disparaitre. Au contraire, le spectre d'applications et de domaines s'intéressant aux ontologies ne cesse de s'élargir, en plus du fait qu'elles peuvent être utilisées dans n'importe quelle application où un domaine doit être conceptualisé.

Anciennement réservée aux systèmes experts simulant des raisonnements humains dans des domaines spécifiques, l'ontologie se retrouve maintenant dans une large famille de systèmes d'information. Elle est utilisée pour décrire et traiter des ressources multimédia, assurer 1'interopérabilité d'applications en réseaux, piloter des traitements automatiques de la langue naturelle, construire des solutions multilingues et interculturelles, permettre l'intégration de sources hétérogènes d'information, décrire des protocoles d'interactions complexes ; vérifier la cohérence de modèles, permettre les raisonnements temporel et spatial , faire des approximations logiques, etc.

Ces utilisations des ontologies se retrouvent dans de nombreux domaines d'application : intégration d'informations géographiques, gestion de ressources humaines, aide à l'analyse en biologie, commerce électronique, enseignement assisté par ordinateur, bibliothèques numériques, échanges commerciaux entre partenaires industriels, suivi médical informatisé, etc.

Un courant particulièrement prometteur pour l'expansion des systèmes à base d'ontologies est celui du web sémantique. Il s'agit d'une extension du Web actuel, dans laquelle l'information se voit associée à un sens bien défini, améliorant la capacité des logiciels à traiter l'information disponible sur le web. L'annotation de ces ressources d'information du web repose sur des ontologies, elles aussi disponibles et échangées sur le web. Grâce au web sémantique, l'ontologie a trouvé un formalisme standard à l'échelle mondiale et s'intègre de plus en plus dans des applications web, sans même que les utilisateurs ne le sachent. Cela se fait au profit des logiciels qui, à travers les ontologies et les descriptions qu'elles permettent, peuvent proposer de nouvelles fonctionnalités.

De ce fait, de plus en plus d'ontologies de domaines sont disponibles : ontologie de la génétique, ontologie de la géométrie, ontologie pour les musées, ontologie médicale, ontologie pour 1'enseignement, ontologie pour le bâtiment, ontologie de systèmes documentaires, ontologie pour la gestion, ontologie dans le secteur automobile, etc.

A l'heure où 1'ontologie se dote d'une ingénierie, cette expansion est loin d'être finie. Parmi ses dernières évolutions, 1'ontologie qui s'appliquait essentiellement à des données (documents, images, vidéos) est maintenant utilisée pour décrire des logiciels (par exemple, des services web), leurs caractéristiques fonctionnelles (types d'entrées, types de sorties) et non fonctionnelles (coût, qualité). Elle pourrait ainsi permettre l'identification, l'invocation et la composition dynamique d'applications à l'échelle du web.

De même, l'ontologie commençait déjà à être utilisée pour décrire les utilisateurs et s'étend maintenant à la description du contexte d'interaction, pour doter les applications de ce que l'on appelle une conscience du contexte. Cela concerne les préférences de l'utilisateur (langue, gouts, droits, etc.), les caractéristiques du terminal (mobile, vocal, etc.), la situation géographique (à l'étranger, dans une salle avec imprimante, etc.), l'activité en tours (au volant, en présentation, etc.), l'historique d'utilisation...etc.

L'introduction d'une ontologie dans un système d'information vise à réduire, voir éliminer, la confusion conceptuelle et terminologique et à tendre vers une compréhension partagée pour améliorer la communication, le partage, l'interopérabilité et le degré de réutilisation possible. Une ontologie informatique offre un cadre unificateur et fournit des « des primitives », des éléments de base pour améliorer la communication entre les personnes, entre les personnes et les systèmes, et entre les systèmes.

Intégrer une ontologie à un système d'information permet donc de déclarer formellement un certain nombre de connaissances utilisées pour caractériser les informations gérées par le système et de se baser sur ces caractérisations et la formalisation de leur signification pour automatiser des tâches de traitement de l'information.

Dans un moteur de recherche c'est, par exemple, pouvoir améliorer la précision de la recherche d'information, en évitant des ambigüités au niveau terminologique (provenant de l'homonymie) ; le taux rappel de cette recherche d'information, en intégrant des notions plus précises ou équivalentes (en utilisant la synonymie, l'hyponymie) ou en déduisant des connaissances implicites (par exemple, des règles d'inférence) ; en relaxant des contraintes trop strictes en cas d'échec de la requête (par généralisation) ; en regroupant des résultats trop nombreux selon leur similarité pour les présenter de façon plus conviviale (regroupement ou clustering conceptuel).

Dans le domaine médical, une décision thérapeutique peut impliquer plusieurs experts médicaux et peut aussi nécessiter une connaissance détaillée des antécédents du patient. Le système Ligne de vie repose sur une ontologie pour décrire les symptômes, les diagnostics, les options et les choix thérapeutiques, intégré différentes contributions au dossier d'un patient et permettre une collaboration non ambigüe des personnes et des systèmes. L'ontologie est utilisée ici comme référentiel commun dans une activité collaborative.

Enfin, si 1'ontologie est actuellement utilisée pour faciliter l'accès à des informations et des applications, on pressent aussi son utilisation dans la description et 1'application de

règles de sécurité et de confidentialité décrites à de hauts niveaux d'abstraction, permettant de restreindre les accès avec une grande flexibilité. Ainsi, dans un système d'information à base d'ontologies, la confidentialité et ses règles reposent aussi sur la sémantique des ontologies et les inférences qu'elle permet pour contrôler l'accès à l'information et la précision de l'information diffusée.

2.10 Conclusion

Nous venons d'avoir une vue générale sur les concepts de base liés à l'ontologie, en l'occurrence, ses définitions, ses composants, ses différents types, son cycle de vie, les langages et les éditeurs d'ontologie, et pour finir, nous avons présenté un certain nombre d'éditeur d'ontologie. Actuellement les ontologies sont utilisées dans tous les domaines de l'informatique. L'un des domaines qui suscite un grand intérêt a l'utilisation des ontologies est le domaine des procédés logiciels où cette dernière nous permettra le partage des connaissances capitalisées. Elle offrira non seulement la possibilité de gérer l'hétérogénéité de la terminologie utilisée mais aussi un outil de validation pour ces procédés. Dans le chapitre qui suit nous allons voir la notion des procédés logiciels (PLs), L'objectif est de cerner les connaissances liées à ce domaine, mais aussi de comprendre les liens et les interactions entre les différents aspects du domaine étudié.

3 Chapitre III LES PROCEDES LOGICIELS

3.1 Introduction

Le point traité dans ce chapitre relève de l'ingénierie des procédés logiciels (PLs). Notre travail s'articule au tour des metamodels des procédés logiciels principalement à la création d'ontologie pour des PLs. Par conséquent, la mise au point sur certain concept du domaine des PLs s'avère indispensable.

L'objectif de ce chapitre est de clarifier la notion de PL ; en effet, différents termes sont utilisés dans ce contexte tel que cycles de vie de logiciels, processus logiciels et procédés logiciels peuvent introduire des ambigüités et prêter à confusion. La modélisation et la méta-modélisation des PLs est le sujet principal de ce chapitre. Comme exemple de métamodèle de PL nous introduisons le métamodèle SEMDM (System and Software Process Engineering Metamodel) qui est exploité par notre approche de création d'ontologie à base de metamodele. Nous avons aussi détaillé les caractéristiques particuliers aux PLs que nous avons jugé importantes. Par ailleurs, étant donné que nous avons dédié le chapitre 4 à l'approches MDA, dans ce chapitre nous citons seulement les principes et concepts de base de MDA.

3.2 Définition des procédés logiciels

L'origine de cette appellation vient de la composition du mot procédé et logiciel. Le mot procédé est défini par Osterweil (Osterweil 1987) comme une approche systématique pour la réalisation d'un objectif. Le terme systématique implique la séparation entre l'exécution réelle des tâches et l'idée abstraite sur la façon dont les taches doivent être ou sont exécutées. Quand au mot logiciel qui fait allusion au géni logiciel qui veut dire une approche systématique et disciplinée pour la construction de systèmes logiciels. A partir de là, le lien entre génie logiciel et procédés est donc naturel comme le définit Camargo (Camargo 2006) pouvoir aborder la construction des logiciels de façon systématique, « Une idée précise ou même formelle de la procédure à suivre est nécessaire ». Une autre définition est celle de Feiler (Feiler and Humphrey 1993) « un procédé logiciel (software process) est, en général, défini comme un ensemble d'activités aussi bien techniques qu'administratives dédiées au

développement et à la maintenance d'un produit logiciel ». Durant les années 80 le domaine de l'ingénierie des procédés logiciels a donné naissance à un nombre considérable de travaux. Malgré le large éventail de langages, d'environnements et d'approches proposés pour leurs modélisations et leurs exécutions, les travaux dans ce domaine sont toujours d'actualité, et plusieurs nouvelles orientations sont actuellement explorées pour améliorer la qualité du PL.

3.3 Le Modèle de procédé

Un modèle de procédés logiciels, ou modèle de PLs, est la représentation plus ou moins formalisée d'un ensemble de PLs. Il explicite les propriétés et les contraintes qui régissent le monde réel du développement en prenant en charge, entre autres, l'évolution prévue et imprévue de la réalité du développement(Tankoano, Derniame et al. 1994)

Différents modèles de PLs peuvent décrire différents angles de développement logiciel (LORSZ, MARTA et al. 2000) , se focalisant sur un aspect particulier du développement et reléguant d'autres aspects au second plan. Ainsi, différents langages de modélisation de PLs existent, suivant les buts recherchés de simulation, d'exécution, etc. Plusieurs classifications de modèles de PLs ont alors été définies, relativement aux critères suivants :

• Le niveau de formalisation du modèle de PL : non formel, semi formel, (ESTUBLIER 2005).

• La nature centrale du modèle de PL : centrée activité, centrée rôle, centrée artefact, etc.(Charlotte 2009).

• L'orientation du modèle de PL : orientée gestion de configuration, orientée décision, orientée stratégie(Charlotte 2009)

- Le type du langage de modélisation de PL: orienté objet, réseau de pétri, à base de règles(Charlotte 2009) ,(Zamli, Isa et al. 2004), (Bendraou, Gervais et al. 2008)

• Le type d'exécution supporté : exécution distribuée, simulation, etc. (LORSZ, MARTA. et al. 2000). (Zamli and Lee 2001) ,(Bendraou, Gervais et al. 2008).

Les concepts des modèles de PLs varient selon les orientations adoptées. Il existe autant de métamodèles que d'orientations de (Charlotte 2009). Cependant, tous les PLs sont conformes au même noyau conceptuel (figure 3.1) : Le PL est un enchainement d'activités, chacune nécessitant des produits «produit de travail» en entrée (inputs) pour fournir des produits en

sortie (outputs). Le PL étant centré humain, une activité est sous la responsabilité d'un rôle «Role».

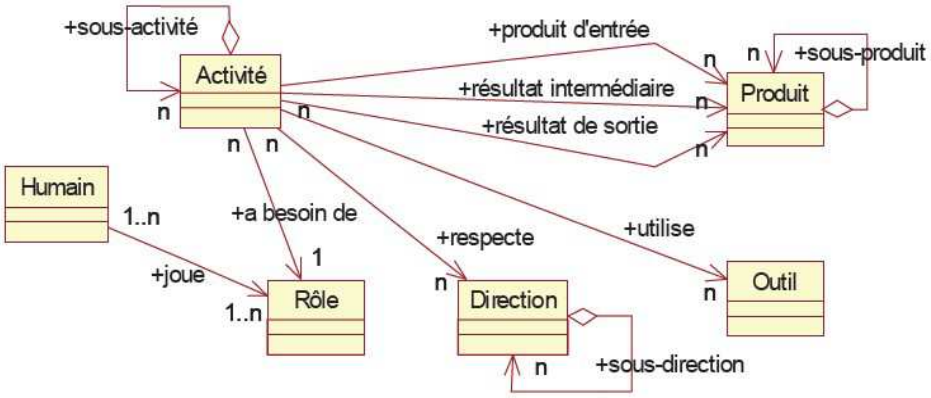

Figure 3.1 noyau conceptuel des modèles de PLs(Montangero 1999)

Ainsi, les concepts primaires de tout modèle de PL sont : Activité, Produit et Rôle. Il est clair que d'autres concepts secondaires tels que stratégie, organisation ou outil peuvent être présents dans un métamodèle de PL, dépendant de l'orientation et de la spécialisation des modèles de PLs que l'on veut décrire.

Éléments primaires : ce sont les éléments principaux qui reflètent l'essence des procédés sans prendre en compte les aspects de planification et d'exécution de procédé. Ce sont :

− Les activités : une activité est un ensemble d'actions à réaliser pour accomplir un objectif de développement.

− Les produits : un produit est un artéfact utilisé ou élaboré par les activités durant le développement.

− Les rôles : un rôle est un concept abstrait regroupant un ensemble de responsabilités et de compétences nécessaires pour réaliser certaines activités de développement.

Éléments secondaires : ce sont les éléments qui fournissent des informations supplémentaires pour mettre en œuvre un procédé. Ce sont :

− Les agents : un agent est une personne participant au développement. Un agent n'est pas un rôle, il peut jouer plusieurs rôles en même temps. Dans le sens inverse, un rôle n'est pas forcement associé à un agent, mais peut être associé à un groupe d'agents. Le concept d'agent est nécessaire pour la gestion de processus.

− Les ressources : une ressource est un élément qui facilite l'exécution d'une activité, par exemple un outil, un guidage, etc. De telles informations sont utiles pour l'assistance au cours d'un processus.

− Les informations qualitatives : elles permettent d'évaluer la performance et la qualité de procédés, par exemple des métriques, les résultats de révision ou test, etc.

− Les informations organisationnelles : elles facilitent l'exécution d'un procédé pour un projet spécifique. Généralement, elles concernent le contexte de travail, l'organisation de ressources, de coopérations et de communications. Les modèles de procédé ne décrivent pas seulement.

3.4 Modélisation et Méta Modélisation des procédés logiciels

Afin d'organiser et de structurer les modèles, l'OMG (Object Managment Group) a défini une architecture de modélisation à plusieurs niveaux appelée : Architecture multi niveaux des modèles (figure-3.2-). Cette architecture fournit un support d'abstraction basé sur les modèles qui permet de maîtriser la complexité croissante des logiciels.

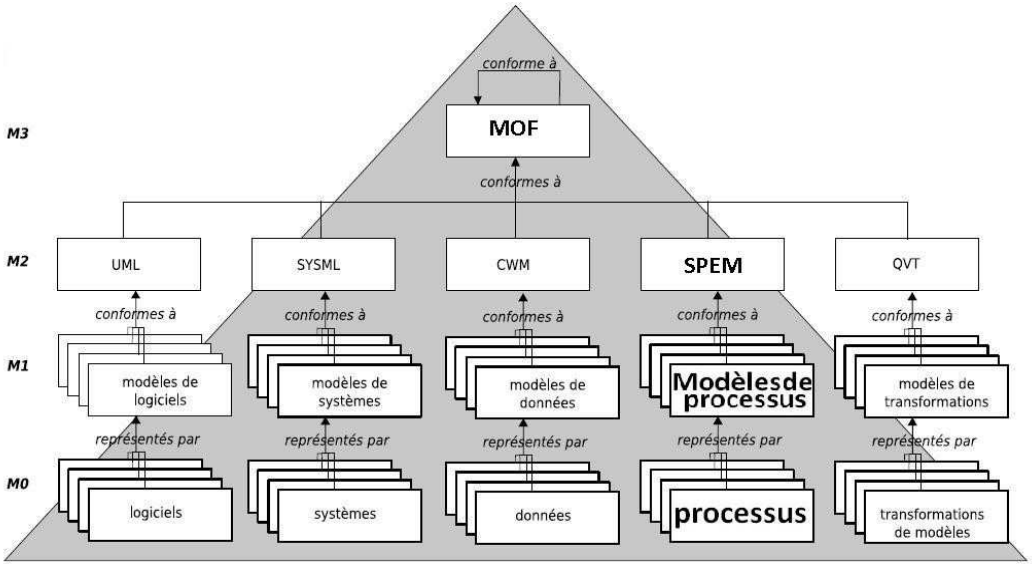

Figure 3.2 Architecture multi niveaux des modèles

Les PLs étant régis par les modèles, ils respectent eux aussi cette architecture, ainsi, la méta modélisation des PLs est organisée comme suit (figure-1.3-) :

– Le niveau M0 décrit les PLs.

– Le niveau M1 décrit les modèles de PLs.

– Le niveau M2 décrit les métamodèles de PLs.

– Le niveau M3 décrit les méta-métamodèles de PLs.

3.5 Objectifs et apports de la modélisation

L'objectif principal de la modélisation de procédés est d'expliciter les pratiques de développement pour pouvoir les étudier, les améliorer et les utiliser de manière répétable, gérable et éventuellement automatisable(Finkelsteiin, Kramer et al. 1994, Rolland 1998). Plus spécifiquement, on attribue généralement trois finalités à la modélisation de procédés (Rolland 1998) : descriptive, prescriptive et explicative. La modélisation descriptive décrit un procédé tel qu'il est (as-is process) dans le but de le comprendre et l'évaluer. La modélisation prescriptive définit un procédé souhaité en décrivant comment il devra être (to-be process). La finalité prescriptive vise à guider les intervenants et à assister l'exécution de procédé par l'intermédiaire d'outils. La finalité explicative vise à modéliser un procédé en fournissant la logique du procédé. Les différents travaux du domaine des procédés (Curtis, Kellner et al. 1992, Derniame and Kaba 1999) s'accordent sur les avantages de la représentation explicite de procédé :

− faciliter la compréhension des procédés : en représentant explicitement un procédé, les connaissances de procédé sont concrétisées et deviennent compréhensives, communicables. Un modèle de procédé est donc un bon moyen pour faciliter la communication entre les intervenants d'un procédé.

− faciliter la gestion et l'exécution de procédé : si un modèle de procédé est défini de manière assez détaillée, et avec une sémantique claire, on peut développer des outils d'assistance pour la planification, le contrôle et la surveillance de procédé en suivant le modèle du procédé. Si le modèle est décrit formellement, on peut même l'exécuter automatiquement. C'est l'objectif des Ateliers de Génie Logiciel centrés-Procédé (AGL-P).

− faciliter l'analyse et l'amélioration de procédés : un modèle de procédé fournit l'information pour analyser et simuler le procédé. De plus, l'existence d'un modèle de procédé permet la réutilisation de procédés bien définis, ce qui est important pour leur amélioration. Les caractéristiques requises d'un modèle de procédé dépendent bien entendu de la ou les finalités pour lesquelles il est développé.

3.6 Classification des procédés (domaine d'utilisation):

Les procédés sont apparus dans le monde de l'informatique dans les années 80, lorsque les informaticiens se sont aperçus de la répétitivité de certaines suites de tâches, et ont cherché à les formaliser et/ou les automatiser. Les informaticiens se sont intéressés en premier à la formalisation des tâches du développement logiciel, ce qu'ils ont appelé le "Procédé

Logiciel". Les procédés informatiques sont utilisés de manière différente dans les différents domaines, on cite:

Les "Procédés Logiciels" (Software Process) dans le domaine de conception logicielle, les procédés servent à gérer et assister le développement logiciel. Il existe plusieurs sous-classes de Procédés Logiciels :

3.6.1 Les Procédés Exécutables :

ils assistent le développement logiciel de manière exécutable (ESTUBLIER 2005), Le Procédé Logiciel Exécutable définit la manière dont le développement logiciel est organisé, géré, mesuré, assisté, et amélioré (indépendamment du type de support technologique choisi pour le développement) (Derniame and Kaba 1999).

3.6.2 Les Procédés Semi-Formels

dans cette catégorie, nous citons :

- Le "Processus Unifié" (RUP : Rational Unified Process).
- GQM (Goal/Question/Metric),
- QIP (Quality Improvement Paradigm),
- TSP (Team Software Process),
- PSP (Personal Software Processs)

3.6.3 Les Procédés Organisationnels :

Comme exemple de procédés organisationnels nous avons :

- CMM (Capability Maturity Model) ;
- CMMI (Capability Maturity Model Integration) amelioration de CMM;
- SPICE (Software Process Improvement and Capability dEtermination).

3.6.4 Les Procédés de Gestion de Configuration (SCM : Software Configuration Management) :

Dans ce domaine, les procédés sont utilisés pour contrôler et assister l'évolution du logiciel,

3.6.5 Les Procédés Logiciels récents ou descriptive :

Ce genre de prodécés s'occupe de la description et non pas de l'exécution de procédé nous citons : OOSPICE, OPEN Process Framework, la norme de l'OMG, SPEM 1.1 (System and Software Process Engineering Metamodel) et sa nouvelle version SPEM 2.0 (SPEM), ainsi que la norme ISO 24744 (ISO/IEC (2007)). Dans la section suivante, nous nous intéressons

au métamodèle de la norme ISO 24744 définit par ISO comme standard pour la modélisation des PLs, il sera exploité dans notre travail.

3.7 Les langages de modélisation des procédés :

Il existe de nombreux langages utilisés par la technologie des procédés. Ces langages sont regroupés sous le terme de langage de modélisation de procédés, les PMLs ("Process Modeling Languages"). Chaque étape (ou phase) du méta procédé est elle-même un procédé et fait appel à un modèle de procédé. Comme un PML a pour fonction d'être utilisé lors de ces différentes étapes, il doit répondre à un certain nombre d'exigences parfois différentes (voir contradictoires) selon les phases. Par exemple, le langage doit offrir une description suffisamment détaillée du procédé pour que celui-ci soit exécuté et il doit être suffisamment compréhensible par un humain lors de la phase de conception, etc. (Kabbaj 2009).

3.7.1 Les principales caractéristiques des PMLs:

- **Formalisation**: c'est le niveau de formalisation de la syntaxe et de la sémantique du PML (formel,semi formel, informel)(Kabbaj 2009) ;
- **Expressivité**: c'est la capacité du PML de représenter tous les éléments du procédé par des dispositifs du PML (Kabbaj 2009) ;
- **Facilité de compréhension**: c'est le degré de facilité avec lequel on arrive à comprendre le modèle décrit (même par des non informaticiens) à travers les notations du PML (texte ou graphique).
- **Abstraction et modularité**: la capacité du PML est de supporter les mécanismes d'abstraction et de modularité afin de structurer le modèle du procédé (Kabbaj 2009).
- **Exécutabilité**: c'est la capacité du PML de définir des modèles opérationnels (exécutables).
- **Facilité d'analyse**: c'est la capacité du PML de définir des modèles descriptifs et faciles à analyser
- **Evolutivité**: c'est le degré de réflexivité du PML, qui garantit un grand soutien à l'évolution du modèle du procédé (Kabbaj 2009).
- **Multi-vues**: c'est la capacité du PML de définir différentes vues (le modèle d'activité, le modèle de produit, le modèle de ressource, le modèle de rôle) dans un modèle commun et global du procédé (Kabbaj 2009).

Il existe un débat au sein de la communauté procédé sur le fait que l'on doit utiliser un ou plusieurs PMLs, c'est à dire utiliser un langage par phase ou bien concevoir un langage "universel" utilisable par toutes les phases. Il apparaît, cependant, difficile de concevoir un tel langage sachant que les caractéristiques suivantes (issues des besoins des différents acteurs du procédé) doivent être respectées : plusieurs niveaux de formalisme, d'expression, de compréhension ou d'exécution (Kabbaj 2009).

3.7.2 Catégories des Langages de Modélisation de Procédé :

Les Langages de Modélisation de Procédé et leurs Environnements de Génie Logiciel Sensibles au Procédé (PSEEs) peuvent être classés en trois catégories suivant l'élément central du modèle de procédé. Ainsi, il y a eu des PMLs centrés Produits, Activités, et Rôles(ESTUBLIER 2005):

- **Les PMLs centrés Produits :** Les PMLs centrés Produits se concentrent surtout sur les données échangées, et ont développé d'importantes propriétés concernant ces données (données orientées objet, transactions, persistance, versionnement,etc). Malgré les difficultés que ces PMLs ont rencontrés, ils ont quand même eu beaucoup de succès, surtout dans le domaine de la gestion de configuration. Cette classe de PML a été fortement utilisée dans l'industrie, ce qui n'a pas été le cas des deux autres classes de PML

- **Les PMLs centrés Activités :** Ils sont conçus pour fournir un support pour la description d'activités. Dans les PMLs centrés Activités, ce sont les activités, représentant les tâches à réaliser, qui forment le concept central. Cette classe de PMLs n'a eu du succès que dans le domaine de la recherche. Dans les années 90, beaucoup de travaux de recherches ont été effectués sur cette classe, c'était d'ailleurs la classe favorite des chercheurs. Ces travaux de recherches ont permis de bien spécifier les concepts des PMLs centrés Activités, mais ils n'ont donné lieu à aucune implémentation commercialisée

- **Les PMLs centrés Rôles:** Dans les PMLs centrés Rôles, l'accent est mis sur les rôles et sur la communication et la collaboration entre rôles. Cette classe de PML n'a pas eu beaucoup de succès dans le domaine des procédés logiciels. Par contre, elle a été utilisée dans le domaine du travail coopératif. Un exemple de

ces PMLs: PWI, et les systèmes de groupware ou de speechact (ESTUBLIER 2005) .

Il existe d'autres classifications des PMLs, comme par exemple celles basées sur les paradigmes du langage(Armenise, Bandinelli et al. 1993, Ambriola, Conradi et al. 1997), sur le degré de formalisation du langage (informels, semi-formels et formels) (Huff 1996, Zamli, Isa et al. 2004) ou encore les classifications basée sur la finalité du langage(LORSZ, MARTA et al. 2000). On retrouve aussi une classification faite dans (Garcia Camargo 2006), dont le but est de comprendre les traits communs et les différences entre les langages.

3.8 La norme ISO/IEC 24744

3.8.1 Définition

la norme ISO/IEC 24744 appelée aussi SEMDM (Software Engineering-Metamodel for Development Methodologies) définit un métamodèle très générique mais qui permet tout de même de créer des méthodes riches et expressives, aussi elle était défini dans le but d'être utilisée dans trois contextes différents. Chacun de ces contextes définit un domaine d'expertise bien déterminé qui correspond à un certain niveau d'abstraction dans le processus de développement des méthodes(ISO/IEC (2007)) Ces trois domaines d'expertise sont : 1) le domaine métamodèle qui comporte le SEMDM et ses éventuelles extensions, 2) le domaine méthode dans lequel les ingénieurs de méthodes définissent leurs méthodes, et 3) le domaine d'application (Endeavour Domain) qui représente le domaine ou ces méthodes sont utilisées. La Figure 3.3 montre les relations entre ces trois niveaux.

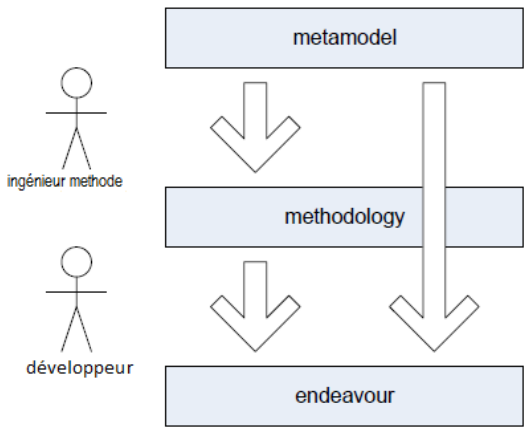

Figure 3.3 Contextes (domaines) définis par la norme ISO/IEC 24744(Henderson-Sellers and Gonzalez-Perez 2005).

Une relation d'utilisation relie ces trois niveaux. Ainsi, les ingénieurs de méthodes instancient les éléments du domaine métamodèle pour pouvoir les utiliser de même que les développeurs utilisent, dans le contexte de leurs projets, les éléments définis dans les domaines méthode et/ou métamodèle.

Un autre avantage est que la norme ISO/IEC 24744 introduit deux concepts clés afin de pallier aux inconvénients des architectures de métamodélisation conventionnelles (Atkinson 1997) (Atkinson and Kühne 2001), notamment le problème de dualité posée par l'architecture adoptée par l'OMG à quatre couches (Four-layer Architecture). Le problème de dualité fait surface lorsque les éléments définis dans les couches intermédiaires, agissent à la fois comme des objets (lorsqu'ils sont le résultat d'une instanciation des éléments de la couche supérieure) et comme des classes (lorsqu'ils font l'objet d'une instanciation). Les deux concepts permettant de contourner ce problème sont le powertype et le clabject. Ces Deux nouveaux concepts ont été introduits pour modéliser une méthodologie.

Les « Powertypes » : introduits d'abord par (Odell 1994) puis adaptés pour le contexte des processus par(Henderson-Sellers 1997), ont l'avantage de fournir une classification dynamique en permettant aux sous-types d'une classe d'être définis comme des instances d'une autre classe en utilisant un discriminant, ce qui est particulièrement utile quand on a des éléments qui appartiennent à deux niveaux d'abstraction différents.

Les « Clabjects » ont l'avantage de fournir un mécanisme d'instanciation profonde. Ainsi, ils permettent de spécifier qu'une entité du niveau méthodologique est à la fois une instance d'une entité du méta-modèle et une classe pour les éléments du niveau « Projet »(Atkinson and Kühne 2001) .

3.8.2 Structure de la Norme ISO/IEC 24744 SEMDM

les composants (éléments) d'une méthodologie SEMDM sont répertoriés en trois catégories majeurs: les éléments Endeavour créés par le développeur durant un projet, les éléments Template qui représentent tout élément créé par l'ingénieur de méthode pendant la construction d'une méthodologie, et les éléments Ressource qui représentent les éléments de la méthodologie utilisés directement au niveau du projet sans instanciation (ex. : références, directives).

La figure 3.4 donne une vue globale du metamodele et les figure 3.5 et 3.6 donne respectivement une vue partielle de la partie template et endeavour.

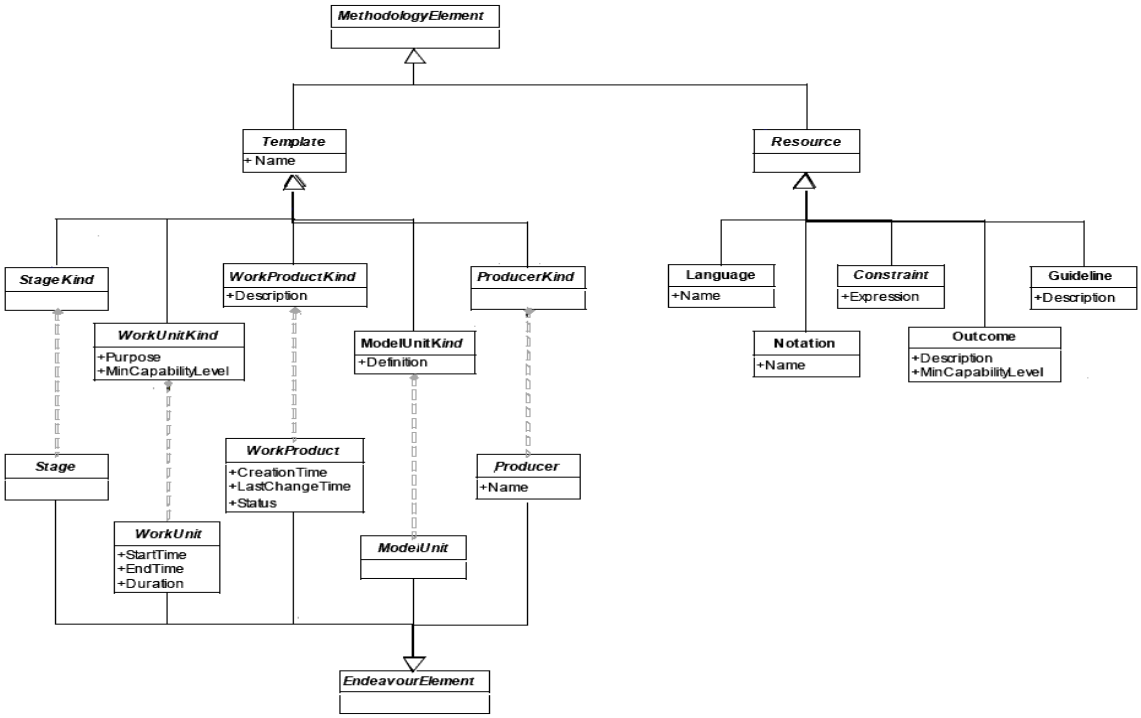

Figure 3.4 vue globale du metamodele de la Norme ISO/IEC 24744

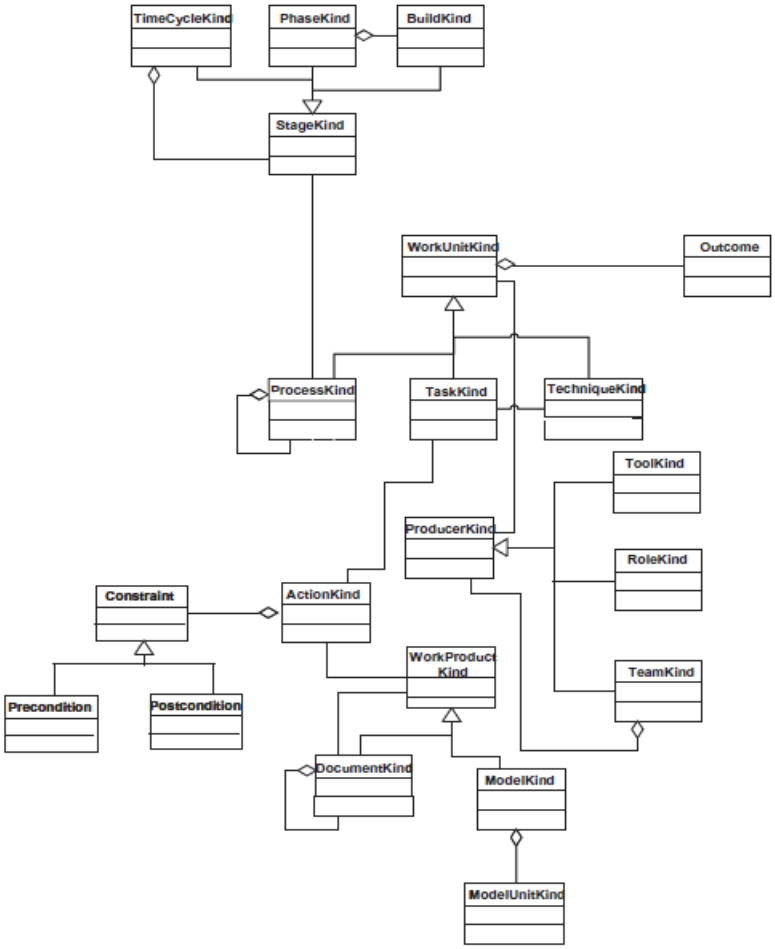

Figure 3.5 vue partiel de la partie Template

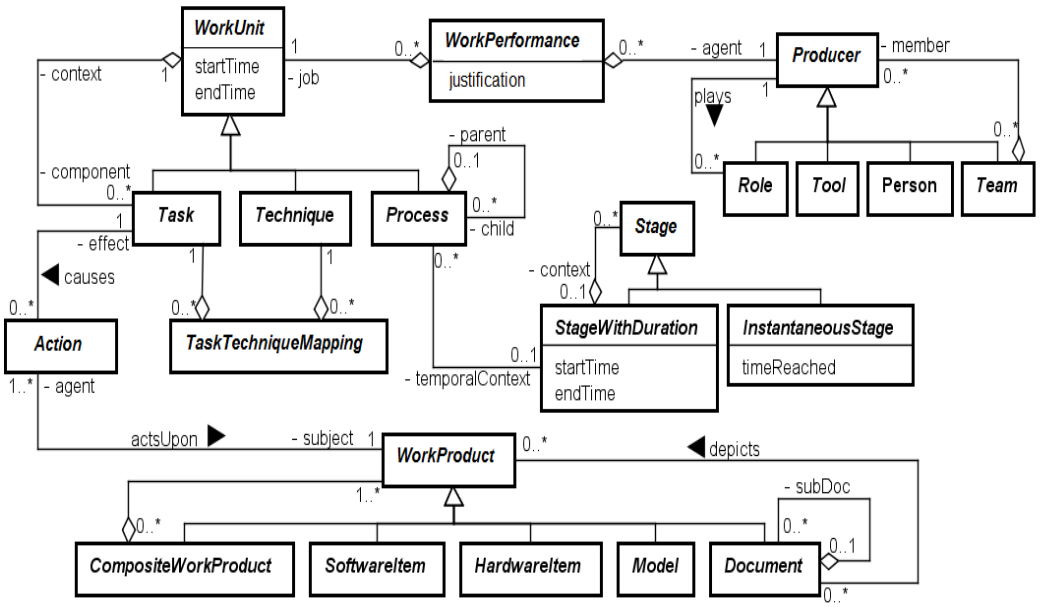

Figure 3.6 vue partiel de la partie Endeavour

Ces trois types d'éléments donnent l'ensemble des classes nécessaires pour s'y prendre avec les trois aspects des méthodologies : le processus, les produits et les producteurs comme le montre la figure 3.7

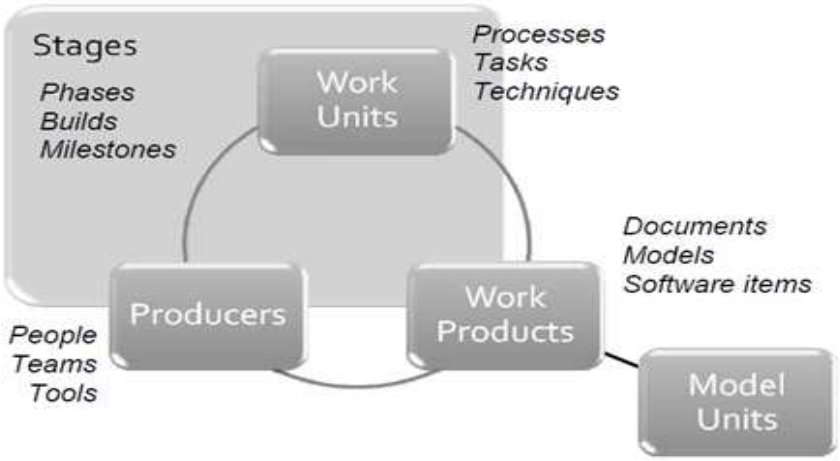

Figure 3.7 les trois aspects des méthodologies : le processus, les produits et les producteurs(Gonzalez-Perez 2007)

3.8.2.1 Processus

la norme ISO/IEC 24744 définit l'aspect processus par les powertypes WorkUnit/*Kind et Stage/*Kind. Le powertype WorkUnit/*Kind permet de représenter "l'effort" du processus qui se concentre sur le travail à réaliser. La classe WorkUnitKind spécifie ce qui doit être fait

sans faire référence au cadre temporel dans lequel s'inscrivent les efforts du processus. Elle représente les types d'unités de travail définis par la méthode. Chacun de ces types est caractérisé par un objectif dans le projet et un niveau de capacité.

D'autre part le powertype Stage/*Kind permet de spécifier la structure temporelle de la méthodologie en déterminant les blocs de temps (intervalles de temps ou moments instantanés (point in time)) gérés dans un projet. La classe StageKind symbolise les types d'étapes (Stages) définis par la méthode. Chacun de ces types est précisé par son niveau d'abstraction et le résultat produit.

Le lien entre le coté effort et le coté temporel est assuré par une connexion qui lie Process/*Kind et StageWithDuration/*Kind. La connexion décrit le fait qu'un processus peut-être exécuté à l'intérieur d'une étape avec durée.

3.8.2.2 Producteur

La norme ISO/IEC 24744 défini l'aspect Producteur par le powertype Producer/*Kind. Ce dernier est relié au powertype WorkUnit/*Kind via le powertype WorkPerformance/*Kind afin de mettre en relation les producteurs et les unités de travail qui leur sont assignées.

La classe ProducerKind symbolise un type particulier de producteur (personnes, groupes de personnes ou outils) dont la responsabilité est d'exécuter une ou plusieurs unités de travail. Un producteur peut jouer plusieurs rôles et peut être engagé dans diverses unités de travail.

3.8.2.3 Produit

La norme ISO/IEC 24744 définit l'aspect produit par les powertypes WorkProduct/*Kind, ModelUnit/*Kind et ModelUnitUsage/*Kind. Les deux derniers powertypes représentent, spécifiquement, l'aspect modélisation des produits.

La classe WorkProductKind symbolise un type particulier de produits (documents, matériels ou collections d'informations) défini par la nature de son contenu et l'idée d'utilisation. Un produit fait toujours l'objet d'une ou de plusieurs actions.

La classe ModelUnitKind représente un type d'unité de modèle (ModelUnit). Un ModelUnit est un élément atomique d'un modèle représentant un fragment cohésif d'information dans le sujet modélisé. Chaque ModelUnitKind est caractérisé par la nature de l'information qu'il représente et par l'intention d'utilisation d'une telle représentation. Le powertype ModelUnitUsage/*Kind représente l'usage d'une unité de modèle par un modèle donné. Un

ModelUnitUsage/*Kind appartient toujours à un modèle qui représente son contexte et se réfère à une unité de modèle qui constitue sa cible.

3.8.3 Utilisation du SEMDM

L'incorporation du concept de powertype rend la norme ISO/IEC 24744 différente des autres approches de métamodélisation. Ceci se répercute par des ajustements dans la façon d'instancier une méthode à partir de ce métamodèle, et plus précisément l'instanciation du patron powertype. L'instance d'un patron powertype est un clabject dont la facette objet est une instance de la classe powertype et la facette classe est un sous type du type partitionné.

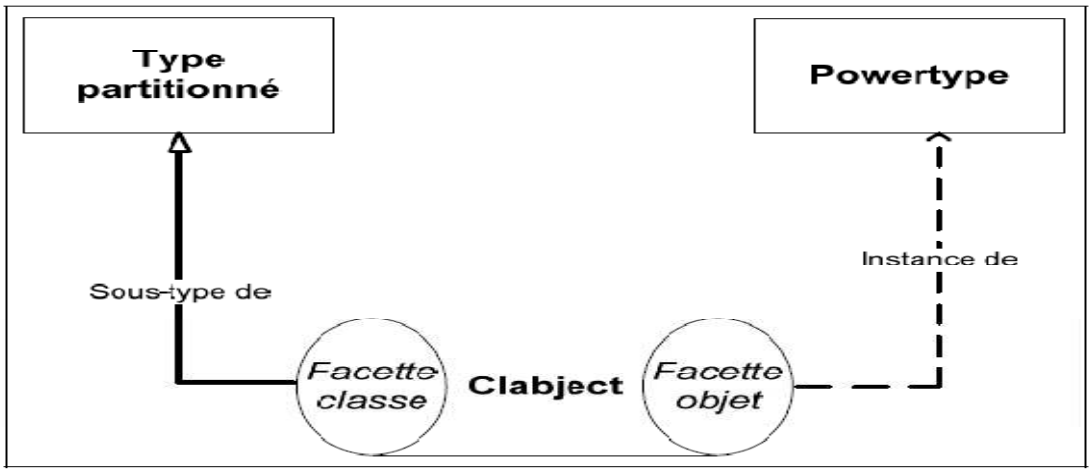

Figure 3.8 Composition d'un clabject

Les classes du métamodèle qui ne font pas partie d'un patron de powertype sont instanciées selon le mécanisme d'instanciation ordinaire et sont utilisées directement au niveau application (Endeavour), comme la classe « Resource ».

La norme ISO/IEC 24744 est utilisée essentiellement par les ingénieurs de méthodes. Ces derniers instancient les éléments de la norme ISO/IEC24744 par la création des clabjects et d'objets comme le montre la Figure 3.8. Les clabjects sont créés à partir des patrons de powertype, c'est-à-dire à partir des classes dérivées des classes Template et Endeavour comme le montre la figure 3.8. Les développeurs utilisent ces méthodes en créant des objets à partir des facettes classes des clabjects. Les facettes objets et les objets Ressources sont utilisés en tant que références. La figure 3.9 montre un exemple d'interactions entres les trois aspects du SEMDM

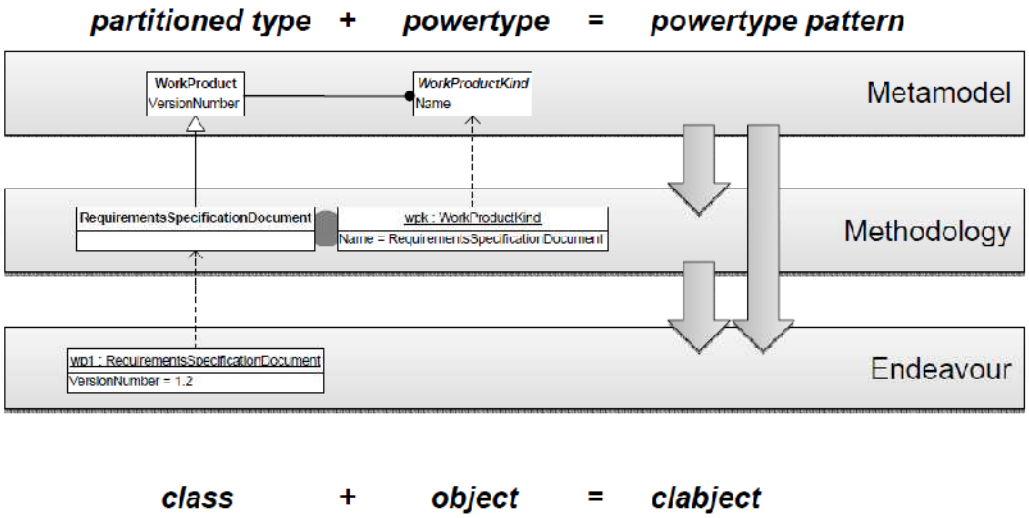

Figure 3.9 Interactions entres les trois aspects du SEMDM

3.8.4 Extension du SEMDM

La norme ISO/IEC 24744 vise à fournir un métamodèle générique et flexible afin de permettre la création de méthodes riches et expressives. Cette flexibilité est accomplie via le mécanisme d'extension qui consiste à créer des extensions (collections d'éléments du niveau métamodèle) en étendant le métamodèle standard. Ces extensions sont créées tout en créant des sous types des powertypes, des classes et des types énumérés standards ou en ajoutant des attributs et des associations aux classes existantes.

3.8.5 Notation pour la norme ISO/IEC 24744

La notation couvre les trois aspects fondamentaux du SEMDM : processus, produit et producteur ainsi que les relations, les contraintes et les concepts auxiliaires définis par le métamodèle.

Les quatre types de diagrammes définis SEMDM sont :

Diagramme de cycle de vie : représente la structure globale d'une méthode en décrivant son aspect temporel (temporal aspect) et son aspect de contenu (content aspect). Les éléments qui sont représentés par ce diagramme sont alors ProcessKind et StageKind (y compris TimeCycleKind, PhaseKind, BuildKind et MileStoneKind). La structure temporelle consiste en un ensemble de StageKind liés entre eux généralement par des liens génériques afin d'exposer la séquence des étapes du cycle de vie. Les StageKind peuvent aussi contenir d'autres StageKind afin de représenter les compositions d'éléments (relations d'agrégation entre StageKind) ou l'aspect itératif d'un élément. Les processus à exécuter au sein des

StageKind sont présentés, dans la structure de contenu, en incluant les symboles ProcessKind à l'intérieur des symboles StageKind.

Diagramme de processus : décrit les détails des processus utilisés dans une méthode en se concentrant sur ce qui doit être fait (le quoi) et qui en est responsable (le qui). Ce diagramme représente, entre autres, les relations entre les types de processus et les liens reliant ces derniers aux types de tâches (TaskKind) et aux types de producteurs (ProducerKind).

Diagramme d'induction : représente une mise en application (Enactment) spécifique d'une méthode, ou d'une partie de cette méthode. Ce diagramme relie les symboles du diagramme de cycle de vie à un diagramme de Gantt permettant ainsi de visualiser dans le temps les divers éléments ProcessKind composant un StageKind.

Diagramme d'action : représente les liens entre la partie processus et la partie produit en montrant les interactions entre les types de tâches et les types d'unités de travail de la méthode. Les éléments représentés dans un diagramme d'actions sont les TaskKind, les WorkProductKind et les ActionKind.

3.9 Conclusion

Après avoir passé en revue la notion de procédé logiciel et de modèle de procédé logiciel, nous nous sommes concentrés sur l'étude des langages de modélisation de procédés ainsi que sur leurs concepts. Aussi, nous avons présenté le métamodèle pour les méthodes de développement (SEMDM) défini par la norme ISO/IEC 24744. La particularité de ce métamodèle réside dans son adoption d'une nouvelle approche basée sur le concept de powertype afin de remédier aux lacunes des approches de métamodélisation conventionnelles comme l'approche de métamodélisation stricte adoptée par le groupe OMG. SEMDM est caractérisé également par une sémantique riche permettant de modéliser les interfaces entre l'aspect processus et l'aspect produit d'une méthode. Ajoutons à cela l'adoption du concept de niveau de capacité qui permet aux ingénieurs méthodes d'établir le niveau de capacité minimale auquel chaque type d'unité de travail peut être effectué. Dans le chapitre suivant nous allons voir comment on peut exploiter automatiquement l'information contenue dans des modèles, dans le but d'automatiser des tâches de développement à travers l'Ingénierie Dirigée par les Modèles (IDM).

4 Chapitre VI L'Ingénierie Dirigée par les Modèles

4.1 Introduction

La philosophie de l'Ingénierie Dirigée par les Modèles (IDM), en anglais MDE (Model Driven Engineering) se base sur l'idée qu'on peut exploiter automatiquement l'information contenue dans des modèles, dans le but d'automatiser des tâches de développement. En effet, les approches dirigées par les modèles tendent à capturer l'intention du développeur, souvent exprimée informellement en langage naturel ou bien avec des diagrammes ad-hoc, comme étant des spécifications formelles qui décrivent les propriétés du logiciel. Ces approches essayent ensuite d'automatiser l'implémentation de cette intention, en compilant les spécifications pour générer du code.

Dans ce chapitre, nous proposons dans un premier temps une présentation des concepts clés de l'IDM et des principales approches afférentes (section 4.2) puis nous focaliserons notre discussion sur l'approche MDA et les travaux de normalisation de l'OMG (section 4.3). Nous détaillons les deux premiers axes principaux de l'IDM selon l'approche MDA et sur lesquels portent nos travaux de recherche : le processus de développement et la modélisation (métamodélisation). La section (4.3.2) présente la transformation de modèles qui est le troisième axe clé de MDA. La section (4.3.5) présente les technique de mis en œuvre entre la spécification des correspondances entre métamodèles et la génération des règles de transformations. Nous concluons enfin par la section (4.4) avec une discussion sur les limites actuelles de l'IDM et les travaux de recherche en cours.

4.2 Concepts de bases de l'IDM

Dans cette section nous faisons le tour sur les concepts de base de l'ingénierie dirigée par les modèles et les relations qui lient ces concepts. Les définitions présentées dans cette section sont la porte d'entrée dans le domaine de L'IDM , les quatre concepts de bases sont : système, modèle, métamodèle et langage.

4.2.1 Modèle et système

La pierre angulaire de l'IDM est la notion de modèle. À ce jour, il n'existe pas une définition consensuelle pour ce qu'est un modèle. Néanmoins, à partir de différentes définitions proposées dans la littérature(Seidewitz 2003, Bezivin 2004), un grand nombre d'auteur utilisent la définition proposée dans (Bézivin and Gerbé 2001) et la considère très représentative, Bézivin et al définissent un modèle comme suit:

"A model is a simplification of a system build with an intended goal in mind. The model should be able to answer questions in place of the actual system ".

Ils définissent un modèle comme étant une abstraction d'un système construit dans un but précis. De ce fait, le modèle contient un ensemble restreint d'informations sur un système et cet ensemble d'informations est choisi pour être pertinent vis a vis de l'utilisation qui sera faite du modèle. On dit alors que le modèle représente le système.

4.2.2 Métamodèle : langage de modélisation

La définition suivante d'OMG du métamodéle est la plus répondue dans la littérature et assez consensuelle:

"A meta-model is a model that defines the language for expressing a model"

Ainsi, un métamodèle représente les concepts du langage de métamodelisation utilisé et la sémantique qui leur est associée. En d'autres termes, le métamodèle décrit le lien existant entre un modèle et le système qu'il représente. Cette notion de métamodèle conduit à l'identification d'une seconde relation, liant le modèle et le langage utilisé pour le construire, désignée par conformsTo dans(Bézivin 2004, Favre 2004) et stipule en fait qu'un modèle est conforme à son métamodèle. En poursuivant l'analogie avec le monde de la programmation, un programme est écrit ou est exprimé dans un langage de programmation. Un modèle est écrit ou est exprimé dans un langage de modélisation. Chacun des langages (programmation ou modélisation) définit les concepts et les règles à suivre (syntaxe) pour écrire un programme source conforme au langage de programmation, ou spécifier un modèle conforme au langage de modélisation (métamodèle).

Un autre point lié à la notion de métamodèle c'est la distinction entre langage et métamodèle qui sont souvent confondus dans la littérature. Bien que ces deux concepts soient proches (voir la définition ci-dessus), ils sont néanmoins différents (Bézivin 2005). Un langage est un système abstrait alors qu'un métamodèle est une définition explicite de ce langage.

4.2.3 Métamétamodèle : langage de métamodélisation

Les concepts de système, modèle et métamodèle ainsi que les relations qui les lient, représentent les piliers sur lesquels s'appuie l'IDM. D'autres concepts non moins importants peuvent être déduits afin d'assurer une complétude et une cohérence du cadre conceptuel de l'IDM. Ainsi, de la même manière qu'il soit nécessaire d'avoir un métamodèle pour interpréter un modèle, pour pouvoir interpréter un métamodèle il faut disposer d'une description du langage dans lequel il est écrit : un métamodèle pour les métamodèles. Le terme métamétamodèle est utilisé tout naturellement pour désigner ce métamodèle en particulier. Un métamétamodèle définit les notions de base permettant l'expression des métamodèles et des modèles. Il permet d'exprimer les règles de conformités qui lient les entités du niveau modèle à celle du niveau métamodèle. Afin de limiter le nombre de niveaux d'abstraction (et ainsi éviter d'avoir à définir un métamétamétamodèle), un métamétamodèle est conçu avec la propriété de métacircularité, câd la capacité de se décrire lui-même.

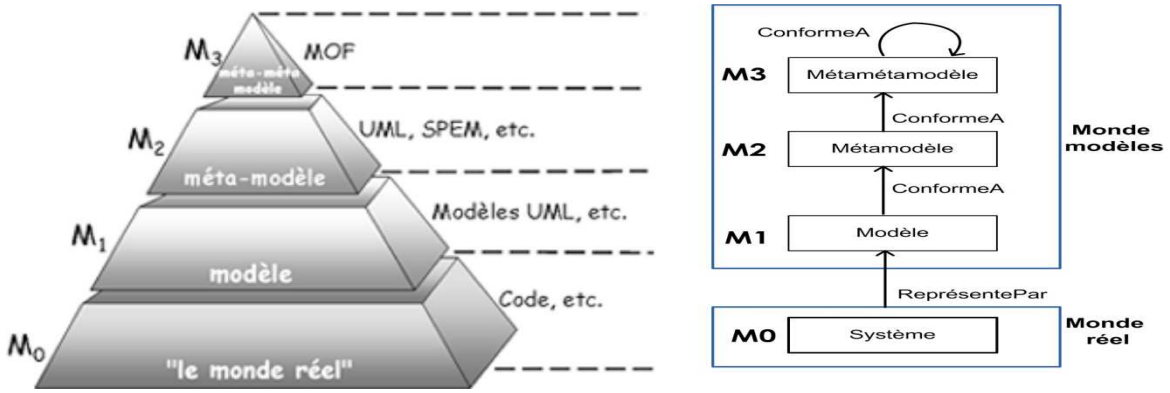

Figure 4.1 Pyramide de modélisation à 4 niveaux

Ainsi, l'IDM est fondée sur une architecture à quatre niveaux (cf. figure 4.1), initialement proposée dans le cadre du MDA(Soley 2000), et qui fait maintenant l'objet d'un large consensus en IDM(Seidewitz 2003, Bézivin 2005). Cette approche consiste à considérer une hiérarchie de métamodèles qui n'est pas propre à l'OMG, ni même à l'IDM, puisqu'elle est utilisée depuis longtemps dans différents domaines de l'informatique. Nos travaux portent sur l'espace technique des modèles à travers l'approche MDA de l'OMG qui sera décrite en détail dans la section 4.2.1.

C'est sur les principes présentés dans cette section que se base l'organisation de la modélisation en IDM généralement décrite sous forme pyramidale (cf. figure 4.1). Le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les métamodèles permettant la définition de ces modèles

(p. ex. UML) constituent le niveau M2. Enfin, le métamétamodèle, unique et métacirculaire, est représenté au sommet de la pyramide (niveau M3).

4.2.4 IDM, et ses principales approches.

Dans le domaine des langages spécifiques aux domaines et de la programmation générative, un certain nombre d'approches basées sur les modèles se développent depuis une décennie. Les trois principales approches parmi cette tendance sont : le Model-Integrated Computing (MIC), le Model-Driven Architecture (MDA) et les Software Factories. Nos travaux étant centré principalement sur l'approche MDA, qui est présentée plus en détail dans la section suivante.

4.3 L'approche MDA

L'Architecture Dirigée par les Modèles (Model Driven Architecture – MDA) est une approche proposée et soutenue par l'Object Management Group (OMG). Cette approche peut être considérée comme une variante de l'IDM pour le génie logiciel. Son objectif est de faire évoluer les pratiques de conception du logiciel vers une approche centrée sur le modèle et non plus sur le code. La figure 4.2 illustre la philosophie, les technologies standards ainsi que l'architecture de modélisation de MDA. La figure de gauche illustre le LOGO de l'OMG concernant l'approche MDA. Celle-ci repose sur un ensemble de standards à partir desquels les plateformes, les applications et les services transversaux sont conçus. La figure de droite illustre trois principaux standards de MDA dans l'architecture à 4 niveaux.

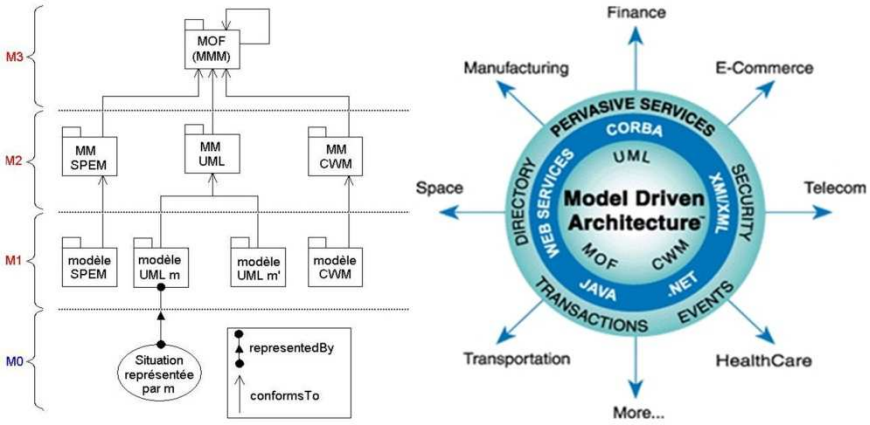

Figure 4.2 Principes, technologies et architecture MDA(OMG 2003)

MDA offre une panoplie de formalismes et standards de modélisation, notamment UML, MOF et XMI, afin de promouvoir les qualités indispensables des modèles, telles que pérennité, productivité et prise en compte des plateformes d'exécution. La section 4.3 est un tour d'horizon sur les principaux standards de MDA et leur place dans le processus de

modélisation et métamodélisation. La clé de MDA consiste en l'utilisation de modèles aux différentes phases de développement d'une application en s'appuyant sur le standard UML. Plus précisément, MDA lesquelles préconise l'élaboration des modèles :

– d'exigences (Computation Independant Model - CIM), dans lesquels aucune considération informatique n'apparait,

– d'analyse et de conception (Platform Independant Model - PIM)

– de code (Platform Specific Model - PSM).

Dans la section 4.3.5 nous discuterons ces trois types de modèles et leur place dans le processus de développement MDA. L'objectif principal de MDA(Blanc 2005), est l'établissement de modèles durables (PIM), sans faire allusion aux détails techniques des plateformes d'exécution (J2EE, .Net, PHP, Oracle, ect.), dans le but de permettre la génération automatique de la globalité des modèles de code (PSM) . Cette génération automatique est réalisable grâce à l'exécution des transformations de modèles. On saisit pourquoi le succés de MDA et de l'IDM en général, repose en grande partie sur la résolution de problème de transformation de modèle. Cette problématique a donnée lieu ces dernières années à plusieurs travaux scientifiques(Jézéquel, Defour et al. 2004) (Jouault 2006) et de normalisation (OMG 2003)

4.3.1 Processus de développement dans MDA

Nous avons insisté sur la place centrale qu'occupe les modèles dans l'IDM et dans l'approche MDA. C'est ce qui la distingue principalement des approches habituelles du génie. Nous avons vu aussi que MDA recommandait l'élaboration de différents modèles : modèle d'exigences CIM, modèle d'analyse et de conception abstraite PIM et modèle de code et de conception concrète PSM. Ces modèles de base seront présentés plus en détail dans les sous sections suivantes. Au dela de ces modèles de base, MDA préconise de modéliser n'importe quel artefact nécessaire au cycle de développement des applications. Nous pouvons donc trouver des modèles d'éxigence, de test, de sécurité, etc. Cependant, toute ces notions de modèles et de formalisme de modélisation ne sont pas suffisantes pour mettre en œuvre MDA. Un processus de développement (appelé aussi méthode de développement) est plus que nécessaire pour MDA. Une méthode définit une démarche reproductible qui procure des résultats crédibles. Tous les domaines de la connaissance utilisent des méthodes plus ou moins compliquées et plus ou moins formalisées.

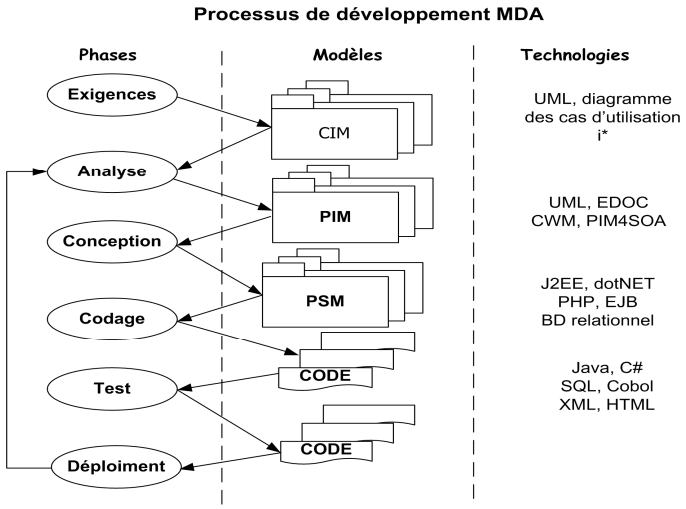

Figure 4.3 Processus de développement selon l'approche MDA

Une méthode d'élaboration de logiciels selon l'approche MDA doit décrire comment construire des systèmes logiciels de manière fiable et reproductible en utilisant les différents types de modèles. La figure 4.3 illustre le processus de développement selon l'approche MDA. Cette figure en trois couloirs est une extension d'une première ébauche présentée dans (Kleppe, Warmer et al. 2003). Le couloir de gauche présente les activités du processus classique de développement de logiciel. Le couloir central présente les différents modèles impliqués dans chaque activité et les liens entre les différents modèles. Enfin, le couloir de droite présente les principaux formalismes préconisés à chaque étape et pour chaque type de modèle. Comme il a été remarqué dans (Kleppe, Warmer et al. 2003), les phases de développement dans MDA sont simillaires au processus habituel de développement de logiciels. La principale distinction demeure dans la nature des artefacts qui sont génères pendant le processus de développement. Les artefacts dans le cadre de MDA sont des modèles formels interprétables et manipulables par une machine.

4.3.1.1 Le modèle d'exigences CIM

Un modèle d'exigence doit indiquer l'application dans son milieu afin d'expliquer quels sont les services offerts par l'application et quelles sont les autres entités avec lesquelles il inter-agit. Le modèle d'exigence explique les besoins du client et symbolise ainsi la position de départ dans le cycle de développement. Les modèles d'exigences peuvent être vu comme des éléments contractuels, affectés à servir comme référence lorsqu'on voudra s'assurer qu'une application est appropriée aux demandes du client. Il est important de noter qu'un modèle d'exigences ne comporte pas d'information sur la réalisation de l'application ni sur les traitements. C'est pourquoi, dans le lexique MDA, les modèles d'exigences sont appelés les

CIM (Computa tion Independant Model), "modèle indépendant de la programmation". Avec UML, un modèle d'exigence peut se résumer à un diagramme de cas d'utilisation. Ce dernier contient en effet les fonctionnalités offertes par l'application (cas d'utilisation) ainsi que les différentes entités qui interagissent avec elles (acteurs) sans donner d'information sur le fonctionnement de l'application.

4.3.1.2 Le modèle PIM

Désque le modèle d'exigences est réalisé et validé par le client, commence alors le travail d'analyse et de conception. Dans cette phase MDA utilise les modèles appelés PIM (Platform Independant Model), désignés aussi par modèle d'analyse et de conception abstraite. En effet, ces modèles spécifiants généralement la logique métier de l'entreprise, sont libres de toute plate-forme technique et ne doivent pas englober d'informations sur les technologies qui seront utilisées pour déployer l'application. En incorporant les détails d'implémentations que très tard dans le cycle de développement, il est possible de maximiser la séparation des préoccupations entre la logique métier des applications et les techniques d'implémentation. Aujourdhui, UML s'es imposé comme la référence pour réaliser les modèles d'analyse et de conception. Il est important de noter que MDA ne fait que préconiser l'utilisation d'UML et qu'il n'exclue pas d'autres langages puissent être utilisés. En effet, l'IDM, au contraire, favorise la définition de langages de modélisation dédiée à un domaine particulier(Jouault 2006) (Domain Specific Languages – DSL) offrant ainsi aux utilisateurs des concepts propres à leur métier et dont ils ont la maîtrise. Ces langages sont généralement de petite taille et doivent être facilement manipulables, transformables, combinables, etc. Quels que soient les langages utilisés, le rôle des modèles d'analyse et de conception est de perpétuer la logique métier de l'entreprise et de faire le lien entre les modèles d'exigences et le code final de l'application. Les modèles PIM doivent être prolifiques, c'est à dire qu'ils doivent être plus précis et contenir plus d'information pour qu'une génération automatique de code soit possible antérieurement.

4.3.1.3 Le modèle PSM

La phase la plus délicate dans MDA, touche la génération de code. Le modèle PSM (Platform Specific Model), appelé aussi le modèle de code ou de conception concrète, est alors mis en jeux. MDA considère que le code final d'application peut être facilement acquis à partir du PSM. En effet, le code d'une application se synthétise à une suite de lignes textuelles, comme un fichier SQL, alors qu'un modèle de code est plutôt une représentation structurée incluant, par exemple, les concepts de table, clé primaire, clé étrangère, etc. L'écriture de code à partir

du modèle de code est donc une opération assez banale. L'essentielle différence entre un modèle de code PSM et un modèle d'analyse et de conception PIM réside dans le fait que le modèle de code est lié à une plate-forme d'exécution. Ainsi, une caractéristique importante des modèles de code est qu'ils assimilent les concepts des plate-formes d'exécution. Pour élaborer des modèles de code, MDA propose, entre autres, l'utilisation des profils UML (voir section 4.3.4). Un profil UML est une adaptation du langage UML à un domaine particulier.

4.3.2 Transformations des modèles dans MDA

L'idee du MDA consiste à passer progressivement des PIM aux PSM pour préparer et assister la génération de code vers la plate-forme technique destinée. Ce passage des PIM aux PSM est une transformation de modeles. Le MDA identifie differents types de transformations de modeles dans le cycle de vie et de développement d'un produit.

PIM vers PIM. Ces transformations s'effectuent pour enrichir, filtrer ou specialiser les informations des modeles en faisant toujours abstraction des informations relatives a la plate-forme technique. Le passage d'un modele d'analyse à un modele de conception est un exemple précis de transformation de PIM vers PIM. Le fait de masquer par exemple des éléments afin de s'abstraire de détails fonctionnels est un autre exemple de transformation de PIM vers PIM.

PIM vers PSM. Lorsque le PIM est suffisamment raffiné pour pouvoir être immergé dans une plate-forme technique donnée, on peut le convertir en PSM en lui rajoutant des informations spécifiques à la plate-forme technique choisie. Cette opération de conversion est une transformation de PIM vers PSM. Les plate-formes visées sont généralement CORBA, .NET, J2EE, et XML. Les régles de conversion (transformations) doivent etre généralisées et capitalisées pour accroıtre leur réutilisation et automatisation dans le futur.

PSM vers PSM. Les transformations de PSM vers PSM s'effectuent pour permette la génération de code. En effet, dans la pratique, une seule transformation de PIM vers PSM n'est pas toujours suffisante pour permettre la génération de code. On est donc parfois amené à transformer les PSM en d'autres PSM en utilisant des formalismes intermédiaires. A ce titre, un scénario possible de génération d'un code C++ à partir d'un formalisme UML consiste par exemple, à passer de UML à SDL (pour Specification and Description Language) puis de SDL à C++. Les transformations de PSM à PSM s'effectuent lors des phases de déploiement, d'optimisation ou de reconfiguration.

PSM vers PIM. Ces transformations sont des opérations de rétroingénieries qui s'effectuent dans le but de reconstruire des modéles indépendants des plate-formes techniques (PIM) à partir de modéles spécifiques aux plate-formes (PSM) techniques. Les PSM considérés peuvent être exécutables (code) ou non exécutables. Ces transformations sont difficiles à réaliser mais elles doivent être prises en compte dans le cadre de l'intégration des produits existants dans la démarche du MDA.

Génération de code. Dans la pratique, certains travaux font la distinction entre les PSM exécutables (ou code) et les PSM non exécutables, mais la génération de code n'est pas toujours considérée comme une transformation de modéles. La figure 4.4 montre quand méme qu'il est possible de passer d'un PSM non exécutable à du code et inversement. Il est important de souligner que le passage du code au PSM est une opération de rétro-ingénierie qui est assez complexe à réaliser. Si le code n'a pas été conçu dans la démarche du MDA, il faut faire appel aux techniques traditionnelles de retro-ingenierie pour effectuer de telles operations.

En somme, il est possible de classer les transformations de modeles possibles dans le MDA dans quatre categories comme l'indique la figure 4.4

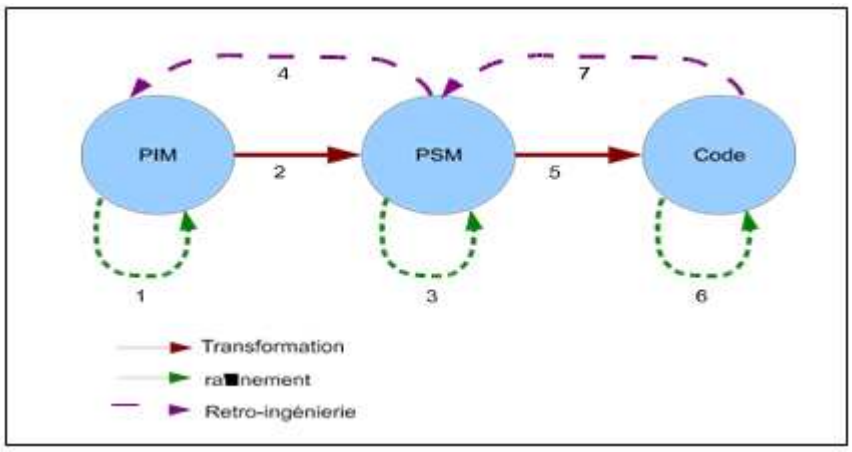

Figure 4.4 Transformations de modèles MDA

– les transformations (2) : décrivent le processus de conversion d'un PIM en un PSM ;

– les raffinements (1), (3)et(6) : introduisent ou suppriment des informations dans un modele ;

– les retro-ingenieries (4)e (7) : convertissent un modèle vers une d'abstraction plus élevé ;

– la generation de code (5) : transforme un PSM non exécutable en un code exécutable.

Nous avons déjà mentionné que la génération de code n'est pas toujours considérée comme une transformation de modèles dans la pratique.

La section suivante présente Le rôle d'UML pour exprimer les modèles et les transformations de modèles dans la démarche du MDA.

4.3.3 Le rôle d'UML dans MDA

L'Unified Modeling Language (UML) (Avgeriou, Guelfi et al. 2005), est le langage standard de modélisation de structures orientées objets. L'apparition des langages orientés objets ont donné naissance à différents langages de modélisation : OMT(Ebert and Engels 1994), Booch (Booch 1993), OOSE(Jacobson, Christerson et al. 1994), etc. Chacune de ces méthodes décrit un formalisme pour construire des modèles objets. Dans les années 90, UML est né de la fusion des modèles OMT, Booch et OOSE. UML est un langage de modélisation objet de plus en plus utilisé. L'OMG a présenté UML comme étant le standard pour l'IDM, tandis que l'IDM permet l'utilisation d'autres langages de modélisation comme nous l'avons vu dans la section précédente.

UML est devenu un des standards les plus utilisés pour spécifier et documenter des systèmes d'information. Cependant, le fait qu'UML a pour but de proposer une notation générique peut limiter la modélisation de domaines particuliers pour lesquels des langages spécialisés peuvent être plus appropriés. UML fournit un ensemble de mécanismes d'extension permettant de remédier à ce problème de généricité et de s'adapter à des domaines particuliers. Dans cette section(4.3.4) nous présenterons le mécanisme d'extension utilisé pour définir des Profils UML. Nous discuterons aussi de l'utilité et de la pertinence des profils UML dans le cadre de l'Ingénierie Dirigée par les modèles.

UML est un formalisme graphique pour spécifier, construire et documenter un système. UML peut donc être utilisé pour développer des systèmes dans des contextes différents et variés tels que la finance, les télécoms, l'Aéronautique, etc., et sur des implémentations différentes (Corba, J2EE, .NET, etc.). Cependant, dans certains cas, un

Langage trop générique comme UML peut ne pas représenter la solution adéquate pour la modélisation d'applications liées à des domaines spécifiques. C'est le cas par exemple lorsque que la syntaxe et la sémantique des entités UML sont dans l'incapacité d'exprimer certains concepts spécifiques de systèmes particuliers ou lorsque l'on cherche à personnaliser les entités UML qui s'avèrent dans certains cas être trop générales.

Pour répondre à ces besoins de spécialisation, l'OMG spécifie deux approches possibles pour définir des modèles ayant pour particularité d'exprimer des sémantiques spécifiques à une plateforme.

La première approche consiste à définir un nouveau langage à utiliser à la place d'UML en utilisant les mécanismes fournis par l'OMG pour définir des langages de modélisation objet (c'est-à-dire en utilisant le même formalisme qui a été utilisé pour définir UML et son métamodèle). Dans une telle approche, la syntaxe et la sémantique des éléments du nouveau langage sont définies dans le but de correspondre aux spécificités de la plateforme cible.

La seconde approche est basée sur le principe de spécialisation d'UML. Dans cette approche les éléments du métamodèle UML sont spécialisés en imposant d'éventuelles nouvelles restrictions entre eux, tout en respectant le métamodèle UML et en conservant la sémantique des éléments non spécialisés. Autrement dit les propriétés des classes, associations, attributs, etc., demeurent inchangées , de nouvelles contraintes seront simplement ajoutées aux éléments originaux. Il est aussi possible de définir au sein de profils UML de nouveaux symboles et icônes pour les éléments créés.

La première approche a été adoptée par les langages tel que le Common Warehouse Metamodel (Poole, Chang et al. 2003) où la sémantique des constructeurs définis ne correspond à aucun élément du métamodèle UML. Ces nouveaux langages sont définis en utilisant le MOF.

Afin de supporter la seconde approche, UML fournit un ensemble d'extensions (stéréotypes, valeurs marquées, et contraintes) pour spécialiser ces éléments à la sémantique d'un domaine particulier. Un profil UML consistera donc à spécialiser les éléments existants du métamodèle UML.

4.3.4 Les profils UML

Nous ne pouvons pas affirmer qu'une approche est meilleure que l'autre dans la mesure où elles ont chacune des avantages et des inconvénients. En effet, en définissant un nouveau langage, il est possible de spécifier une notation qui conviendra parfaitement à une plateforme particulière. Cependant, comme ces nouveaux langages ne respectent pas la sémantique d'UML, il ne sera alors pas possible d'utiliser des outils tiers existants pour créer des modèles, générer du code, faire de la génération inversée, etc. Réciproquement les profils UML, autorisant l'utilisation d'outils de modélisation existants, peuvent ne pas fournir une

notation suffisamment adéquate requise par certains systèmes. Le choix de l'approche à adopter s'avère ainsi non trivial. Cependant, l'historique d'UML, le fait qu'UML soit préconisé par le MOF et la définition de ce formalisme comme étant le standard de modélisation dans le milieu du génie logiciel, sont autant d'arguments qui incitent à opter pour l'utilisation de profils UML dans un cadre d'Ingénierie Dirigée par les Modèles.

Par nature, un modèle UML ne peut pas être productif (dans le sens où cette méthode est un formalisme de modélisation et non un langage de programmation), compte tenu de la multitude de langages existants et d'une sémantique trop générale. Il en découle le besoin de spécialiser, de profiler, UML pour des domaines précis. Un profil permet de spécialiser le formalisme UML pour un domaine particulier. De nombreux profils ont été développés pour des domaines ciblés ou des technologies particulières. Par exemple, les profils les plus répandus sont les profils CORBA et EJB. Dans le cas du profil EJB, il est par exemple possible de préciser qu'une classe au sens UML est une EJB Session ou un autre concept de ce domaine. Il est important de noter que les profils UML permettent la personnalisation de tout métamodèle définis à l'aide du MOF. D'une manière similaire, un profil UML peut aussi en spécialiser un autre. UML met en évidence différentes raisons pour lesquelles un développeur serait amené à spécialiser un métamodèle qu'il aurait défini :

- Pour obtenir une terminologie adaptée à une plateforme particulière ou à un domaine particulier.
- Pour définir une syntaxe pour des constructeurs qui n'auraient pas de notations dans le métamodèle existant.
- Pour définir une notation différente pour un élément déjà existant.
- Pour ajouter des concepts qui n'existent pas dans le métamodèle.
- Pour ajouter des contraintes restreignant l'utilisation du métamodèle et de ses constructeurs dans certains contextes.
- Pour ajouter des informations qui peuvent être utilisées lors de processus de transformation de modèles.

Un profil UML est défini comme un paquetage UML stéréotypé « profil » qui peut soit étendre un métamodèle ou un autre profil. Un profil UML est défini à l'aide de trois mécanismes : stéréotypes, contraintes et valeurs marquées.

Dans la figure 4.5, nous illustrons ces concepts par la définition d'un profil simple et minimal. Dans cet exemple, nous ajoutons deux nouveaux éléments que nous nommons Teinture et Intensité. De plus, nous associons à ces deux éléments des propriétés telles que la couleur pour un élément de type Teinture et une intensité numérique pour un élément de type Intensité. Dans cet exemple, nous restreignons l'utilisation de ces deux nouveaux éléments sur les méta-classes existantes Class et Association.

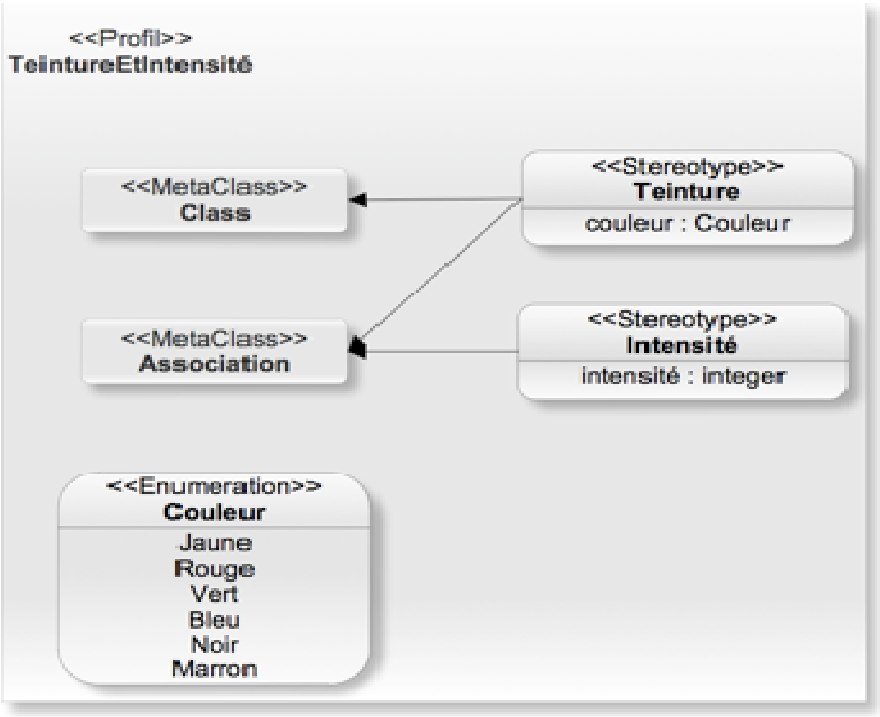

Figure 4.5 Exemple d'un profil UML.

Un stéréotype est défini par un nom et par l'ensemble des méta-classes auxquelles il est attaché. Graphiquement, les stéréotypes sont définis dans des boîtes du formalisme UML, stéréotypées « Stereotypes ». Dans notre exemple le profil UML indique que des éléments de type class et association peuvent être teintées, mais que seules les associations peuvent avoir une intensité. Les éléments du métamodèle sont spécifiés par les stéréotypes

« Metaclass ». La notation d'extension est matérialisée par une flèche allant du stéréotype vers la méta-classe.

Il est possible d'ajouter des contraintes aux stéréotypes définis, imposant ainsi des restrictions aux éléments correspondants du métamodèle. Ces contraintes sont spécifiées à l'aide d'OCL(Richters and Gogolla 2002). Dans notre exemple nous spécifions qu'une association doit avoir la même couleur que les classes liées de la manière suivante :

```
context UML::InfrastructureLibrary::Core:: Constructs::Association

inv : self.isStereotyped("Coloured") implies

self.connection->forAll

(isStereotyped("Coloured")

implies color=self.color)
```

Régle 4.1 Exemple de contrainte OCL.

Des valeurs marquées permettent de définir des méta attributs attachés à la méta classe du métamodèle étendu par un profil UML. Les valeurs marquées possèdent un nom et un type et sont associées à un stéréotype. Dans l'exemple de la figure 4.5, le stéréotype Intensité possède une valeur marquée intensité de type entier. Graphiquement, les valeurs marquées sont spécifiées comme étant des attributs de la classe définissant un stéréotype.

Un profil UML est donc un ensemble de stéréotypes, contraintes et valeurs marquées. Comme nous l'avons mentionné précédemment, le mécanisme de profil permet d'étendre la syntaxe et la sémantique des éléments. Les profils UML restent un moyen efficace lorsque l'on souhaite spécialiser la sémantique d'UML à un domaine particulier.

Différents profils ont été définis et publiquement disponibles. Certains de ces profils ont été adoptés et standardisés par l'OMG, tels que le profil pour CORBA, CCM (CORBA Component Model), EDOC (Enterprise Distributer Object Computing) et le profil EAI (Enterprise Application Integration). Cette liste est non exhaustive. De nombreux profils ont été définis et d'autres sont en cours d'approbation par l'OMG. L'intérêt de ces profils publics est de pouvoir être réutilisables par toute application de modélisation UML.

Dans le cadre d'une approche d'Ingénierie Dirigée par les Modèles, les activités les plus importantes sont la modélisation des différents aspects d'un système et la définition de transformations d'un modèle vers un autre d'une manière automatique. Nous aborderons la problématique de transformation de modèles dans la section 4.3.5

Les profils UML peuvent tenir un rôle important dans la description de la plateforme cible et dans les règles de transformation entre les modèles. Si nous utilisons des profils UML pour spécifier le métamodèle d'une plateforme spécifique, cela garantira que les modèles

dérivés seront en concordance avec UML. Nous pouvons affirmer que le succès d'une approche IDM réside dans l'utilisation maximale de standards.

Les mécanismes fournis par les profils UML conviennent à la description de modèles pour toute plateforme. Nous avons besoin de définir des correspondances (mappings) entre chaque élément d'un PIM, et les stéréotypes, contraintes, et valeurs marquées qui forment un profil UML. L'idée d'un profil UML dans une approche IDM est d'utiliser les stéréotypes définissant les concepts d'un PIM et de produire les éléments correspondants du PSM.

Dans cette section, nous avons présenté les profils UML comme étant un moyen efficace d'étendre le métamodèle UML pour le personnaliser à la sémantique d'une plateforme et d'un domaine spécifique. Les outils actuels de modélisation permettent la définition et l'utilisation de profils mais uniquement au niveau graphique. Cela signifie que la vérification des contraintes associées aux stéréotypes n'est pas ou très peu supportée ce qui nous amènera à proposer un mécanisme de définition de contraintes et plus précisément un formalisme de validation incrémentale que nous présenterons dans la seconde partie de cette thèse concernant nos contributions.

4.4 Ontology definition medel (ODM)

L'ODM(OMG 2014a), est un standard adopté par l'OMG qui supporte le développement d'ontologies et la modélisation conceptuelle dans plusieurs langages de représentation standards. Il fournit une structure cohérente pour la création d'ontologies basée sur MOF et UML.

L'ODM offre un ensemble de méta-modèles et de configurations pour rapprocher le monde de méta-modèles et celui d'ontologies. Il définit cinq méta-modèles (RDFS, l'OWL, Topic Maps, Common Logic and Description Logic), deux Profils UML (le Profil RDFS/OWL, le profil Topic Maps) et un ensemble de configurations (QVT d'UML à OWL, des Topic Maps à OWL et RDFS/OWL à Common Logic).

Actuellement, il existe une mise en œuvre de deux de ces méta-modèles RDFS et l'OWL à travers KM3, du profil RDFS/OWL, ainsi que de la configuration entre UML et l'OWL en utilisant ATL(ATL). La Figure 4.6 présente l'architecture générale de l'ODM pour le passage d'UML à OWL en se basant sur le profil RDFS/OWL :

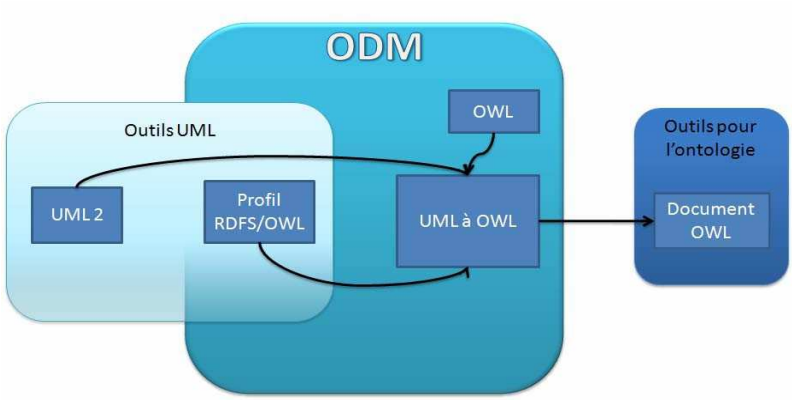

Figure 4.6 Architecture générale de l'approche ODM

La spécification ODM a mis en œuvre la transformation ATL pour accomplir le passage d'UML à OWL en se basant sur la configuration QVT(OMG 2011b). Cette transformation a rendu possible la conversion d'un modèle UML arbitraire dans une ontologie OWL. Le scénario complet de cette transformation est donné dans la Figure 4.7.

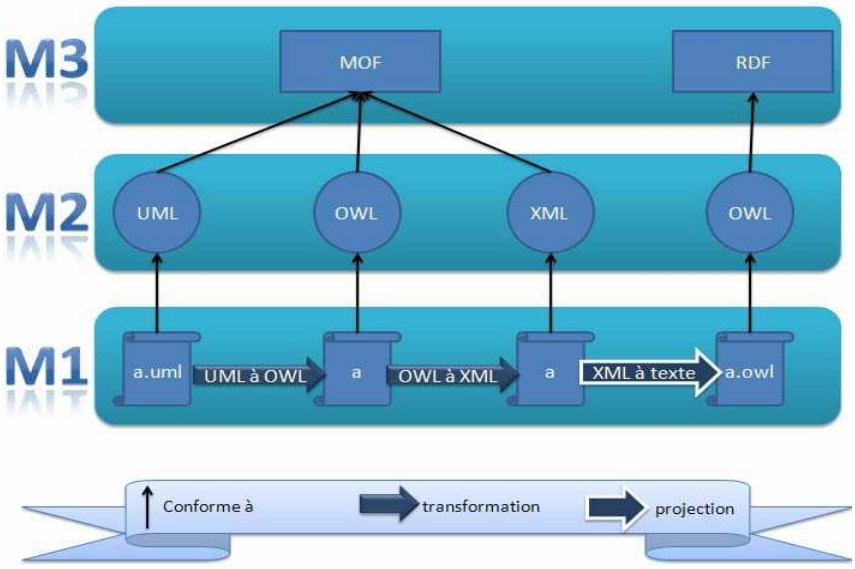

Figure 4.7 Scénario détaillé de l'approche ODM

Ce scénario est composé de deux transformations ATL définies par la spécification W3C. La première transformation fondamentale prend en entrée un modèle UML et produit une ontologie conforme au méta-modèle OWL.

4.5 transformation de modèles

Les deux principes fondamentaux de l'ingénierie des modèles sont la modélisation et la transformation de modèle. D'un point de vue général, on appelle transformation de modèles tout processus dont les entrées et les sorties sont des modèles. De nombreux outils, tant commerciaux dans le monde de l'open source, sont aujourd'hui disponibles pour faire la transformation de modèles. Nous pouvons distinguer quatre catégories d'outils :

• Les outils de transformation génériques qui peuvent être utilisés pour faire de la transformation de modèles. Dans cette catégorie, se trouvent d'une part les outils de la famille XML, comme XSLT(Kay 2000) ou Xquery (Robie 2007), et d'autre part les outils de transformation de graphes(Guerra and De Lara 2004). Les premiers ont l'avantage d'être déjà largement utilisés dans le monde XML ce qui leur a permis d'atteindre un certain niveau de maturité. En revanche, l'expérience montre que ce type de langage est assez mal adapté pour des transformations de modèles complexes (c'est-à-dire allant au-delà des problématiques de transcodage syntaxique) car ils ne permettent pas de travailler au niveau de la sémantique des modèles manipulés mais simplement à celui d'un arbre couvrant le graphe de la syntaxe abstraite du modèle, ce qui impose de nombreuses « contorsions » qui rendent rapidement ce type de transformation de modèles complexe à élaborer, à valider et surtout à maintenir sur de longues périodes.

• Les ateliers de génie logiciel (AGL) intégrant des outils de transformation. Dans cette catégorie, on trouve une famille d'outils de transformation de modèles proposés par des industriels spécialisés dans le développement d'ateliers de génie logiciel. Par exemple, l'outil Arcstyler (OMG 2002a) de Interactive Objects propose la notion de MDA-Cartridge qui encapsule une transformation de modèles écrite en JPython (langage de script construit à l'aide de Python et de Java). Ces MDA-Cartridges peuvent être configurés et combinés avant d'être exécutés par un interpréteur dédié. L'outil propose aussi une interface graphique pour définir de nouvelles MDA- Cartridges. Dans cette catégorie on trouve l'outil Objecteering (Site), qui propose un langage de script pour la transformation de modèles appelé J, et bien d'autres encore, y compris dans le monde de l'open source avec des outils comme Fujaba(Geiger and Zündorf 2006). L'intérêt de cette catégorie d'outils de transformation de modèles est, d'une part, leur relative maturité et, d'autres part, leur excellente intégration dans l'atelier de génie logiciel qui les héberge. Leur principal inconvénient est le revers de la médaille de cette intégration poussée : il s'agit la plupart du temps de langages de transformation de modèles propriétaires sur lesquels il peut être risqué de miser sur le long terme. De plus, historiquement, ces langages de transformation de modèles ont été conçus comme des ajouts au départ marginaux à l'atelier de génie logiciel qui les héberge. Même s'ils ont aujourd'hui pris de l'importance, ils ne sont toujours pas vu comme les outils centraux qu'ils devraient être dans une véritable ingénierie dirigée par les modèles. En outre ces langages

montrent leur limitation lorsque les transformations de modèles deviennent complexes et qu'il faut les gérer, les faire évoluer et les maintenir (Giese and Wagner 2009).

• Les outils conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement normalisés. Ce sont des outils conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standards. Nous trouvons par exemple Mia Transformation, c'est un outil qui exécute des transformations de modèles prenant en charge différents formats d'entrée et de sortie (XMI, tout autre format de fichiers, API, dépôt). Les transformations sont exprimées comme des règles d'inférence semi- déclaratives (dans le sens où il n'y a pas de mécanisme de type de programmation logique), qui peuvent être enrichies en utilisant des scripts Java pour des services additionnels tels que la manipulation de chaînes. Dans le monde académique, on trouve de nombreux projets open source s'inscrivant dans cette approche : les outils ATL (Jouault, Allilaire et al. 2008) et MTL(Silaghi, Fondement et al. 2005), AndroMDA(Bohlen 2007).

• Les outils de métamodélisation dans lesquels la transformation de modèles revient à l'exécution d'une application tels que MetaEdit+(Tolvanen and Rossi 2003). Celui-ci permet de définir explicitement un métamodèle et, au même niveau, de programmer tous les outils nécessaires allant des éditeurs graphiques aux générateurs de code en passant par des transformations de modèles. Dans une catégorie similaire, l'outil MetaGen(Revault, Sahraoui et al. 1995) propose un environnement pour l'édition de métamodèles et un langage à base de règles pour exprimer des transformations de modèles. Les métamodèles sont compilés vers des classes Smalltalk qui peuvent être instanciées par un éditeur de modèles génériques. Plus récemment, l'environnement XMF-Mosaic(Mosaic 2007) de Xactium a été conçu comme un environnement complet pour la définition de langage. Au cœur de XMF-Mosaic se trouve un noyau exécutable de définition de langage (qui est d'ailleurs auto définissant). Tout est modélisé sur cette base, que ce soient les langages (tel que UML), les outils, les GUI, parseur et autres analyseurs XML, mais, à la différence de l'environnement MetaEdit+, obtenu par génération plutôt que par instanciation. Le langage de métamodélisation de base est un langage orienté objet qui a toute la puissance d'un langage de programmation complet ce qui en fait un outil particulièrement puissant pour la transformation.

4.5.1 Technique de la transformation de modèles

Le processus de transformation est composé de trois étapes :

- La définition des règles de transformation,
- L'expression des règles de transformation,
- L'exécution des règles de transformation.

Les deux dernières étapes définissent un système de transformation. Nous détaillons ensuite les trois étapes une par une.

4.5.1.1 Définition des règles de transformation :

Etant donné un modèle source dans un langage L1, (tel que UML) et un modèle cible dans un langage L2 (tel que Java), il s'agit dans cette étape d'élaborer une mise en correspondance des concepts de L1 à ceux de L2 (ex. une classe UML correspond à une ou plusieurs classes Java). Dès lors, on a recours à la technique de métamodélisation pour mettre en place une base de règles exhaustive et générique.

En outre, on peut définir un métamodèle de transformation permettant de définir un langage abstrait de transformations. Les modèles instances de ce métamodèle sont des spécifications de transformations entre deux métamodèles spécifiques. De cette manière, la spécification de transformation devient lisible (comprendre un modèle est plus facile que de comprendre du code), et réutilisable (le métamodèle représente les règles de transformation de manière abstraite indépendante du langage de sa mise en œuvre).

Les règles de transformation sont établies entre le métamodèle source et le métamodèle cible, c'est-à-dire entre l'ensemble des concepts du modèle source et celui du modèle cible. Le processus de transformation prend en entrée un ou plusieurs modèles conformes à des méta-modèles source et produit en sortie un ou plusieurs autres modèles conformes à un ou plusieurs métamodèles cibles, en utilisant les règles préalablement établies.

4.5.1.2 Expression des règles de transformation :

Pour exprimer les règles de transformation, un langage de spécification de règles est nécessaire. Actuellement, il n'existe pas de langage standard.

En 2002, l'OMG a émis un appel à proposition (RFP – Request For Proposal) pour la standardisation du processus de transformation. Ceci va produire le standard MOF/QVT

(Queries/Views/Transformations) qui est une technique de transformation dont la syntaxe doit être définie en un métamodèle MOF. Il doit permettre :

- La correspondance entre les modèles définis en MOF.
- L'interrogation d'un modèle MOF afin de filtrer et sélectionner des éléments du modèle source à transformer.
- La création des vues sur des méta-modèles MOF, une vue sur un système modélisé est un modèle dérivé du modèle du système, ne révèle que certains aspects de ce dernier.

Un langage de transformation peut être déclaratif, impératif ou hybride. Dans la programmation déclarative, on décrit d'une part les données du problème à traiter et d'autre part les contraintes sur ces données. Le programme s'exécute à partir de la situation courante décrite par les données tout en respectant les contraintes. Le langage déclaratif décrit ce qu'on devrait avoir à l'issue d'un certain nombre de données initiales. XSLT (Extensible Stylesheet Language Transformation) (W3C 1999), est un exemple de langage déclaratif de transformation. Par opposition à un programme déclaratif, un programme impératif décrit comment le résultat devrait être obtenu en imposant une suite d'actions que la machine doit effectuer. Un langage hybride regroupe à la fois les paradigmes de programmation déclarative et impérative : l'ordre d'exécution des modules doit être spécifié tandis qu'au sein d'un même module, la détermination de l'ordre d'exécution des règles n'est pas à la charge de l'utilisateur.

4.5.1.3 Exécution des règles de transformation

Une fois spécifiées et exprimées, les règles requièrent un moteur d'exécution pour être exécutées. Ce moteur prend comme entrée le modèle et le métamodèle source, le métamodèle cible, ainsi que le modèle de transformation (les règles de transformation écrites dans le langage de transformation, basées sur les correspondances entre les deux méta- modèles source et cible) et son métamodèle (représentant la grammaire du langage de transformation). Il produit en sortie le modèle cible. La figure 4.8 illustre le processus de transformation de modèles.

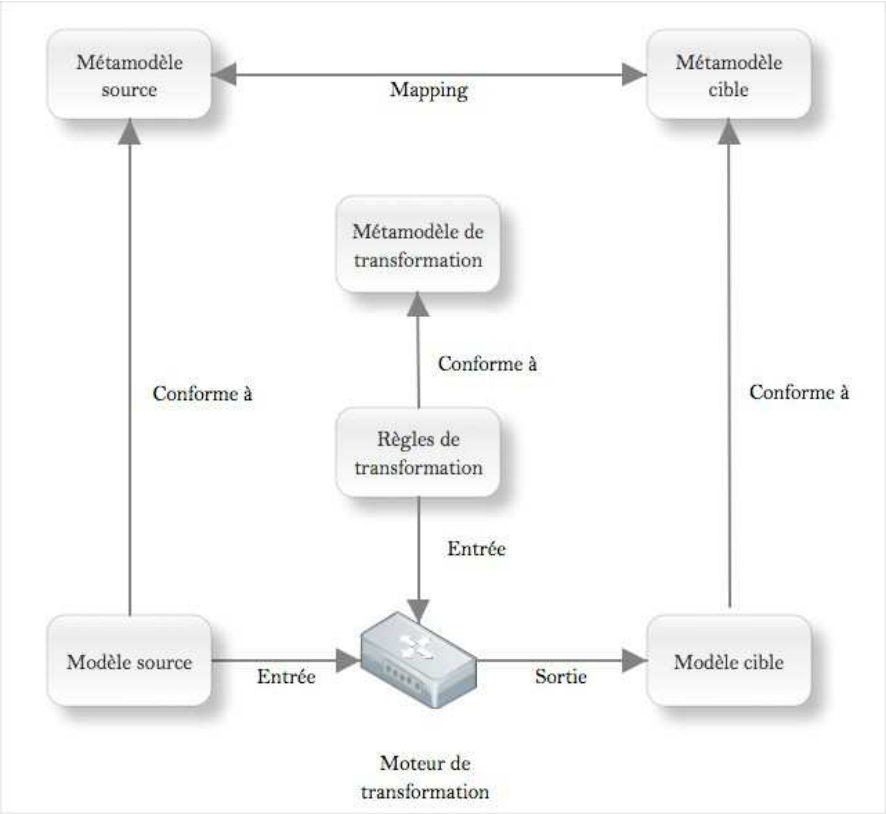

***Figure* 4.7 le processus de transformation de modèles**

La transformation de modèles peut avoir les caractéristiques suivantes :

- Une transformation est dite inversible ou sans perte si la transformation inverse existe.

- Une transformation est dite d'optimisation si elle transforme un modèle en un modèle équivalent optimisé selon une métrique donnée. Un exemple d'optimisation est l'amélioration de la réutilisabilité ou encore la lisibilité.

- Une transformation peut être caractérisée par le niveau d'abstraction des métamodèles source et cible. Plus on est proche de la plateforme d'exécution, moins on est abstrait, et inversement.

- Une transformation peut être caractérisée par sa position dans un processus de développement logiciel. Elle va donc être caractérisée par les acteurs qui l'appliquent et la stratégie de son application.

- Une transformation peut être également caractérisée par son langage d'expression.

De plus, elle peut être totalement ou partiellement automatisée. En outre, une approche de transformation de modèles est caractérisée par :

- les règles de transformation.
- la portée de l'application des règles qui est soit la source soit la cible.
- la relation entre le modèle source et le modèle cible qui précise si la transformation a comme résultat un autre modèle cible, ou un autre modèle mettant à jour le modèle source, ou encore le modèle source modifié.
- la stratégie de l'application des règles détermine la suite des éléments du modèle source sur lesquels les règles vont être exécutées.
- l'ordonnancement des règles détermine l'ordre d'exécution des règles.
- l'organisation des règles spécifie la structure des règles (orientées source, orientées cibles, indépendantes) ainsi que les relations de composition entre elles.
- la traçabilité précise la manière de garder la trace de la transformation. Les traces peuvent servir aux analyses, au débogage, à la synchronisation entre modèles, etc.
- Le sens de la transformation spécifie si les règles sont unies ou bi directionnelles.

4.5.2 Les langages de transformation

L'Ingénierie Dirigée par les Modèles se focalise sur les spécifications du MOF pour définir des langages de transformation de modèles. Dans cette section, nous présentons de façon non exhaustive les langages de transformation qui ont été définis à partir des recommandations du MOF. Nous présenterons les trois langages de transformation standardisés et basés sur le MOF :

- MOF Query/View/Transformation (QVT), le langage de transformation standard de modèles respectant le MOF.
- ATLAS Transformation Language (ATL), un langage de transformation calqué sur

QVT, implémenté en Java et pouvant manipuler des modèles EMF.

- MOFScript, basé sur les spécifications de QVT, transforme des modèles en fichiers texte.

4.5.2.1 MOF Query/View/Transformation

Query/View/Transformation (QVT) (Kurtev 2008) est le langage de transformation standard pour les modèles basés sur une architecture MOF. MOF QVT est considéré comme

étant une partie des spécifications du MOF. QVT est basé sur une architecture de langages déclaratifs et procéduraux permettant une manipulation facilitée des éléments issus de modèles ou de métamodèles et la définition de règles de transformation sur ces éléments. L'architecture proposée par QVT est présentée par la figure 4.9 :

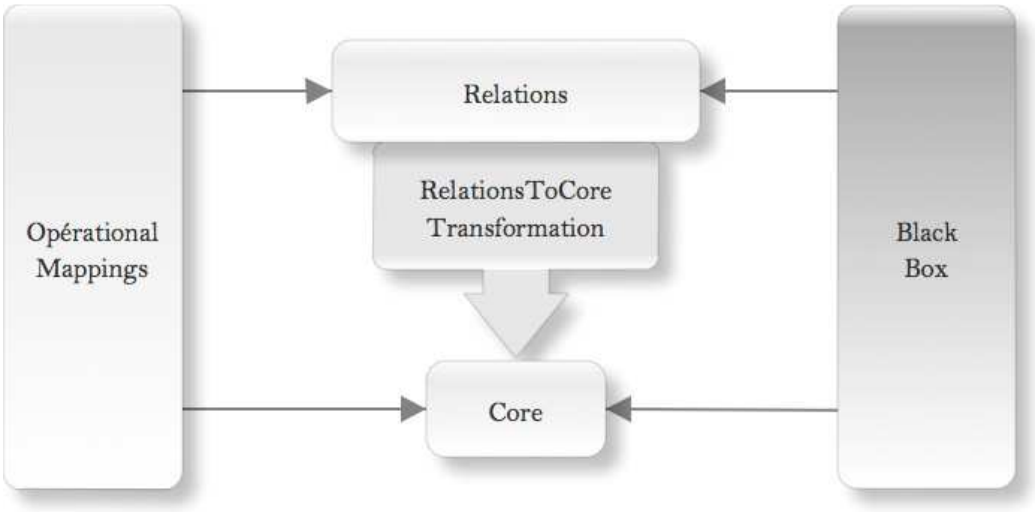

***Figure* 4.8 Architecture QVT des langages de spécification de modèles.**

Parmi les langages déclaratifs, QVT propose un langage de relations (Relations) et un langage noyau (Core) permettant de spécifier des correspondances sur les éléments des modèles manipulés. Pour les langages procéduraux, QVT définit un langage d'opérations de correspondances (Operational Mappings) et un langage d'implémentation interne d'opérations (Black box). Ces langages procéduraux doivent donner accès aux relations et éventuellement aux opérations des modèles manipulés.

Le langage de relations de QVT est utilisé pour définir de manière déclarative des relations existantes entre les éléments d'un modèle source et les éléments d'un modèle cible suivant certaines contraintes. La vérification d'une relation et de ses contraintes conduira à la transformation de l'élément source en élément cible. Il est nécessaire que ce langage soit déclaratif afin que les diverses transformations nécessaires puissent se faire d'une manière parallèle. Par exemple, la transformation d'une classe UML fait appel non seulement à la relation qui concerne cette classe, mais également à la relation qui concerne les éléments englobés par la classe selon le métamodèle (attributs et opérations).

Le langage des opérations doit être réflexif. Nous devons notamment pouvoir définir un ensemble de relations permettant de passer du langage de relations vers le langage noyau. Cet ensemble de relations est représenté par l'élément « RelationsToCore » de la figure 4.9.

Le langage noyau est un langage de bas niveau possédant une syntaxe assez simple, qui permet d'opérer un filtrage sur des ensembles de variables en évaluant des conditions issues des modèles traités. Le langage noyau est implémenté dans une sorte de machine virtuelle.

Le langage d'opérations de correspondances produit les mêmes effets que le langage de relations. Il s'agit d'une enveloppe procédurale du langage déclaratif des relations, permettant en particulier d'assurer une certaine facilité d'utilisation aux développeurs peu habitués aux langages déclaratifs. Des opérations peuvent être utilisées pour décrire des relations complexes ou encore pour décrire une transformation entière procédurale. Ce niveau de langage fait appel au langage OCL.

QVT à pour principal avantage de reposer sur les standards existants à savoir le MOF et OCL, permettant ainsi de développer des outils qui seront d'une part simples à utiliser pour un grand nombre d'utilisateurs, d'autre part compatibles avec un grand nombre d'outils de modélisation existants.

4.5.2.2 ATLAS Transformation Language (ATL)

ATL (Jouault, Allilaire et al. 2008) est un langage de transformations de modèles dans le contexte de l'Ingénierie Dirigée par les Modèles. La première réalisation d'ATL est basée sur MetaData Repository (MDR)(Hnětynka and Píše 2004) et Java, tandis que la deuxième est basée sur MDR, EMF, Java et Eclipse(eclipse).

Une requête ATL est une expression en OCL qui peut retourner des types primitifs, des éléments d'un modèle, des collections, des n-uplets, ou une combinaison de ces types (par exemple, collections de n-uplets). La version actuelle d'ATL ne supporte ni la transformation incrémentale, ni la bidirectionnalité.

Cependant, les concepteurs d'ATL préconisent l'utilisation du support de traçabilité pour réaliser cette caractéristique en ATL. La transformation en ATL est unidirectionnelle. ATL est un langage hybride (déclaratif et impératif). L'approche déclarative d'ATL est basée sur OCL. L'approche impérative en ATL contient des instructions qui explicitent les étapes d'exécution dans une procédure (helpers). La figure 4.9 présente un extrait du métamodèle des règles de transformation ATL. Selon la syntaxe abstraite, une règle est définie par un nom

et peut contenir les éléments ActionBlock et OutPattern. Une MatchedRule spécialise une Rule et contient un InPattern. Une CalledRule spécialise une Rule ou une Operation en OCL.

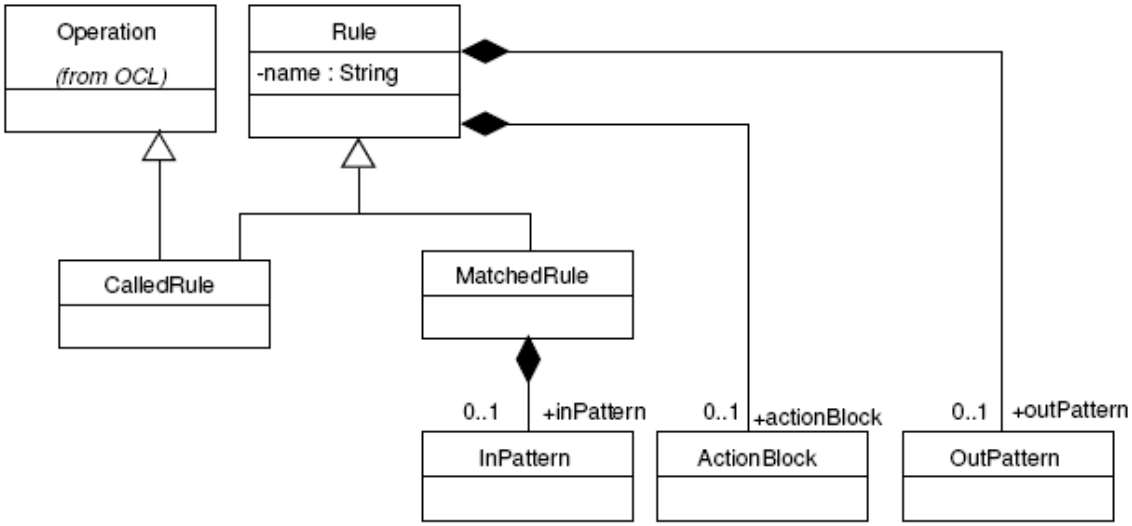

***Figure* 4.9 Extrait du métamodèle des règles ATL.**

Une règle est explicitement appelée en utilisant son nom et ses paramètres (une CalledRule) ou exécutée comme un résultat de la reconnaissance d'un inPattern dans le modèle source (une MatchedRule). Le résultat de l'exécution d'une règle peut être déclaré en utilisant un outPattern, de manière déclarative, impérative ou les deux.

Une règle formée d'un inPattern, d'un outPattern et sans section impérative est définie comme une règle déclarative. Une règle formée avec un nom, des paramètres et une section impérative, mais sans un outPattern est définie comme une règle impérative. La combinaison des deux formes est simplement appelée règle hybride.

La Figure 4.10 présente les détails de la relation entre un InPattern et un OutPattern. L'élément MatchedRule spécifie un patron source (inPattern) comme un ensemble de types d'origine du métamodèle source associé à l'ensemble des noms de variables et optionnellement filtré en utilisant une expression logique en OCL. Un patron cible (outPattern) est un ensemble de types d'origine du métamodèle cible associé à des noms de variables et des liaisons (bindings). Quand une règle contenant un patron cible est exécutée, les éléments cibles sont créés. Une liaison spécifie la valeur utilisée pour initialiser une propriété d'une instance. En fait, un patron cible est un ensemble de noeuds qui peuvent être liés par des liaisons.

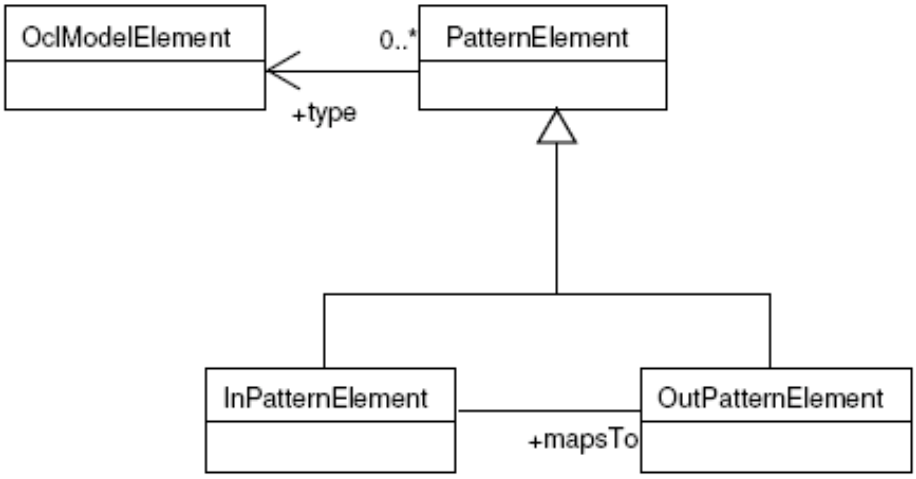

***Figure* 4.10 Patrons d'éléments (syntaxe abstraite).**

Les constructions déclaratives ATL sont constituées d'un InPattern représenté par le mot clé from, d'un outPattern représenté par le mot clé to, et d'un ensemble de liaisons représentées par le symbole <-.

Les instructions impératives ATL peuvent être regroupées en :

- Expressions : basées sur OCL.
- Variables : Une déclaration d'une variable est faite par le mot clé let. Par exemple : let

VarName : varType = initialValue ;

- Attribution : L'opérateur d'affectation <- peut être utilisé à cette fin. De plus, ATL fournit l'opérateur := pouvant être utilisé quand une résolution automatique n'est pas nécessaire. Ceci permet à l'opérande de gauche de prendre la valeur de celle de droite.
- Manipulation d'instances : Les instances peuvent être créées de manière implicite grâce à l'approche déclarative. De plus, il est possible d'expliciter la création ou la destruction d'une instance en utilisant les opérateurs new et delete.
- Déclarations conditionnelles if, else.
- Déclaration de boucles : ATL permet la déclaration de boucles while, do...

while(condition) et foreach.

De plus, ATL fournit le mécanisme des helpers pour éviter la redondance de code et la création de grandes expressions OCL dans une règle. Ceci induit aussi une meilleure lisibilité des programmes ATL.

Un helper en ATL est une fonction qui peut recevoir des paramètres et retourner une valeur ou une instance d'un élément. Les helpers sont toujours utilisés dans le contexte d'une transformation ou pour d'autres helpers.

Selon la taxonomie créée par(Czarnecki and Helsen 2003), ATL a les caractéristiques suivantes :

- Règles de transformation : ATL fait bien la distinction entre LHS (Left Hand Side), et RHS (Right Hand Side). Une règle en ATL est une combinaison de variables syntaxiquement typées et d'une logique exécutable basée sur OCL.
- Stratégie d'application des règles : en ATL, la stratégie est déterministe.
- Organisation de règles : ATL supporte la modularité.
- Traçabilité : ATL supporte la traçabilité par le biais de l'enregistrement des étapes de transformation réalisées par le moteur de transformation.
- Sens des transformations : une transformation en ATL est unidirectionnelle.

Par rapport à la classification générale, ATL est un langage permettant d'effectuer la transformation d'un modèle en un autre modèle avec les caractéristiques suivantes :

- ATL étant en principe déclaratif, il est aussi relationnel (les contraintes de ces relations sont spécifiées en OCL).
- ATL n'est pas directement basé sur la théorie des graphes.
- ATL contient des caractéristiques déclaratives et impératives.

4.6 conclusion

Nous avons introduit dans ce chapitre les notions qui sont à la base des principes généraux de l'IDM, c'est-àdire la métamodélisation d'une part et la transformation de modèles d'autre part. Ces deux axes constituent les deux problématiques clé de l'IDM sur lesquelles la plupart des travaux de recherche se concentrent actuellement. Aujourdhui, l'IDM reste fortement marquée par l'approche MDA et les standard de l'OMG, c'est pourquoi nous avons focalisé

notre présentation et nos travaux de recherche sur cette approche. Nous nous sommes particulièrement intéressés à trois axes principaux de l'approche MDA et sur lesquels portent nos travaux de recherche :

– Les techniques et langages de modélisation et métamodélisation dans MDA

– Les techniques et langages de transformation de modèles dans MDA

– Le processus de développement dans MDA

Nous nous sommes focalisé sur la transformation de modèles qui est au centre de l'approche MDA. Le chapitre suivant sera dédié aux travaux connexes dans lequel nous allons passer en revue les différant travaux qui ce rapproche au notre

5 Chapitre V Travaux connexes

5.1 INTRODUCTION

La communauté du génie logiciel a reconnu très tôt, les ontologies comme une voie prometteuse pour la résolution des problèmes de génie logiciel(Happel and Seedorf 2006, Gašević, Kaviani et al. 2009). Par exemple, les ontologies sont proposées pour être utilisées dans l'ingénierie des exigences, la conception de logiciels, la maintenance de logiciels, la réutilisation de logiciels et la gestion des connaissances, etc. Ces synergies entre ontologies et génie logiciel ont également attiré l'attention des organismes de normalisation dont plusieurs travaux sont en cours.

Afin de faciliter le travail des chercheurs et des praticiens Happel et Seedorf(Happel and Seedorf 2006) définit un système de classification simple qui permet une meilleure différenciation entre les différentes ontologies de génie logiciel. Ils utilisent deux dimensions pour classer les ontologies. D'une part, en fonction de leur position dans le cycle de vie du génie logiciel (analyse, conception, développement, tests, etc.) et, d'autre part, en fonction de leur utilisation. En Combinant ces deux dimensions, quatre zones différentes sont définies:

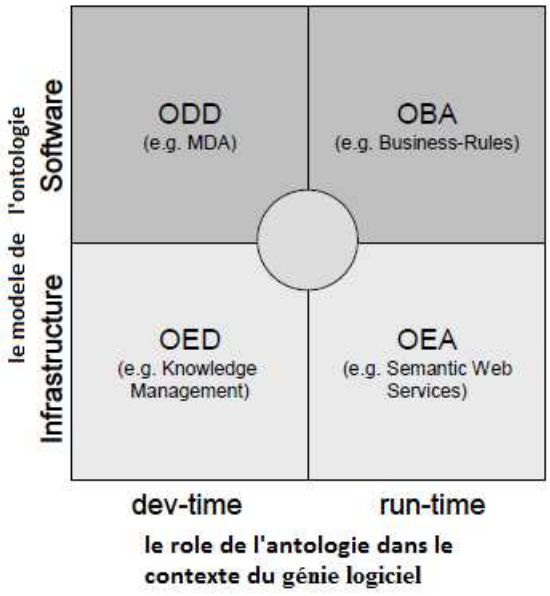

Figure 5.1 Classification des ontologies de génie logiciel

- Ontology-driven development (ODD) : englobe l'utilisation d'ontologies au moment de développement qui décrivent le domaine du problème lui-même.
- Ontology-enabled development (OED) utilise également les ontologies au moment du développement, mais pour soutenir les développeurs dans leurs tâches.
- Ontology-based architectures (OBA) utilisent une ontologie comme un artefact principal à l'exécution. L'ontologie constitue un élément central de la logique d'application.
- Ontology-enabled architectures (OEA) Enfin, tirent partie des ontologies pour fournir un soutien de l'infrastructure au moment de l'exécution d'un système logiciel.

Ruiz et Jilera (Ruiz and Hilera 2006) ont présenté l'état de l'art en matière d'utilisation des ontologies en génie logiciel et de la technologie logicielle. Ils ont offert une taxonomie basée sur deux grands axe à savoir 1) les ontologies de domaine: pour représenter la connaissance d'un domaine particulier au sein de génie logiciel et de ses technologie ou 2) les ontologies comme artefacts logiciels: Les ontologies sont utilisées comme une sorte d'artefact au cours d'un processus logiciel.

Dans notre travail, et selon les taxonomies présentées, nous utilisons une classification afin d'examiner les diverses ontologies réservées au procédé logiciel (génerique ou specifique) et l'objectif d'utilisation de l'ontologie(exigences, maintenance et gestion des connaissances).

5.2 Les ontologies génériques

Wille, Abran, Desharnais, et Dumke (Wille, Abran et al. 2003),ont été les premiers à examiner la possibilité d'exploiter les connaissances consignées dans le Guide SWEBOK, objectivant la construction d'une ontologie pour l'un de ses sous-domaines du génie logiciel - la qualité logicielle (SWEBOK KA11). Leurs résultats préliminaires, identifient le concept de qualité (et leurs multiples sous-concepts);analysent la terminologie utilisée pour se référer à celui-ci; vérifient comment il est utilisé à travers les dix sous-domaines de connaissances de SWEBOK et finalement, proposent une première structuration hiérarchique pour le concept de qualité logiciel, le reliant à d'autres concepts pertinents, ainsi qu'en identifiant les types de liens.

Dans Wille, Dumke, Abran, et Desharnais(Wille, Dumke et al. 2004), les auteurs posent les premières réflexions concernant l'intérêt de la construction d'une ontologie pour le génie

logiciel et l'opportunité de le faire, à partir du corpus des connaissances consensuelles colligées par SWEBOK. Ils soulignent un défi majeur auquel les participants au début des travaux du Projet SWEBOK ont été confrontés Concernant l'absence d'une terminologie du domaine génie logiciel ayant une acceptation de façon plus large. SWEBOK au cours des panels Delphi a contribué à la construction d'un tel consensus à propos du vocabulaire utilisé dans le domaine. Ensuite les auteurs approfondissent la réflexion concernant la construction d'une ontologie de domaine, utilisant SWEBOK comme source primaire de connaissances, exploitant le consensus construit au long des huit années de travaux de SWEBOK à propos du corpus de connaissance du domaine génie logiciel et proposent une approche pour la construction d'une telle ontologie.

Dans Mendes et Abran (Mendes and Abran 2005) les auteurs présentent les premiers résultats globaux d'une ontologie du génie logiciel (encore au niveau conceptuel à ce stade-ci), contenant plus de 6.000 concepts reliés par des liens normalisés, pour l'ensemble des dix domaines de connaissance faisant partie du corpus des connaissances du génie logiciel. Un processus de validation réalisé à plusieurs niveaux (interne et externe) représentatifs du domaine est proposé, impliquant la participation de praticiens et d'experts du domaine. De plus, l'utilisation de cette ontologie est évoquée pour l'harmonisation du vocabulaire et du niveau des descriptions employées dans le Guide SWEBOK, à travers un processus de vérifications croisé de concepts utilisés à travers les divers chapitres de SWEBOK. Il est évoqué aussi, l'utilisation de ces résultats pour améliorer la structuration du Guide SWEBOK lors des prochaines révisions programmées.

Finalement, (Sicilia, Cuadrado et al. 2005) reconnaissent que 1) SWEBOK représente une étape importante dans la construction d'un consensus autour du contenu du génie logiciel en tant que domaine de connaissances; 2) que les ontologies sont les outils appropriés pour traduire le consensus acquis, sous forme de règles logiques pouvant être utilisées/exploitées par des applications informatiques 3) que le contenu des descriptions de SWEBOK ne peut pas être directement traduit du format texte vers des éléments dans une ontologie, car les définitions en langage naturel ne sont pas formelles et requièrent un certain degré d'interprétation par un humain et parfois, le recours directement aux sources (contenues dans le corpus des connaissances externe), mentionnées dans les références.

Pour conclure, Sicila et al., (2005)(Sicilia, Cuadrado et al. 2005) à la lumière de certains résultats de recherche de Wille et al (Wille 2004) et al et Mendes et Abran (Mendes and

Abran 2005), proposent d'utiliser l'ontologie SWEBOK à l'intérieur d'une procédure de révision structurée des descriptions de SWEBOK, afin de dépister des ambiguïtés ou des définitions trop superficielles. De plus, la mise à jour périodique du contenu de l'ontologie SWEBOK devrait contribuer à assurer à son tour, un arrimage pour les révisions de contenu et d'évolution du Guide SWEBOK. Les futurs travaux comprendront la continuation du développement de l'Ontologie SWEBOK ainsi que la mise au point d'une interface de recherche basée sur cette ontologie pour assister le processus de révision du guide.

Guizzardi et (Guizzardi, de Almeida Falbo et al. 2008) a présenté les derniers développements dans l'ontologie fondamentale UFO. UFO est une ontologie fondamentale développée pour soutenir les activités de la modélisation conceptuelle et organisationnelle. Elle est composée de trois niveaux, qui reflètent la structure du monde: (i) UFO-A (Endurants), une ontologie des objets. (ii) UFO-B (Perdurants), une ontologie des événements. (iii) UFO-C (entités sociales et intentionnelles), une ontologie des entités sociales, y compris les aspects linguistiques. Afin de rendre possible l'activité de modélisation conceptuelle via UFO, il a été proposé un langage de modélisation conceptuelle qui utilise des contraintes ontologiques de UFO-A comme primitives de modélisation. Un tel langage, qui est nommé OntoUML, a été spécifié au-dessus du méta-modèle UML 2.0, à savoir, des Meta-Object Facility (MOF). L'objectif était d'assurer la correspondance entre MOF et la structure UFO.

Falbo et Bertollo(Falbo and Bertollo 2009) en se basent sur l'idée de processus logiciels standard, l'auteur a créé l'ontologie (SPO) Software Process Ontology. Cette ontologie a évolué au cours des dernières années, avec l'aide d'une ontologie UFO l'ontologie fondamentale unifiée. En outre, Falbo et al. (de Almeida Falbo, Barcellos et al. 2013) ont évalué tous les artefacts de l'ontologie SPO pour créer ce qu'il appelle Patterns langage d'ontologie pour Software Process (OLP-SP). L'objectif est de définir des morceaux d'ontologies de processus logiciel qui peuvent être réutilisés par d'autres ontologies. Cette ontologie est indiquée comme étant suffisamment expressive pour être utilisée comme un terrain commun pour le mappage des fragments de processus logiciel de normes telles que ISO / IEC 12207-ISO 9001: 2000 ISO / IEC 15504, CMMI, RUP et SPEM.

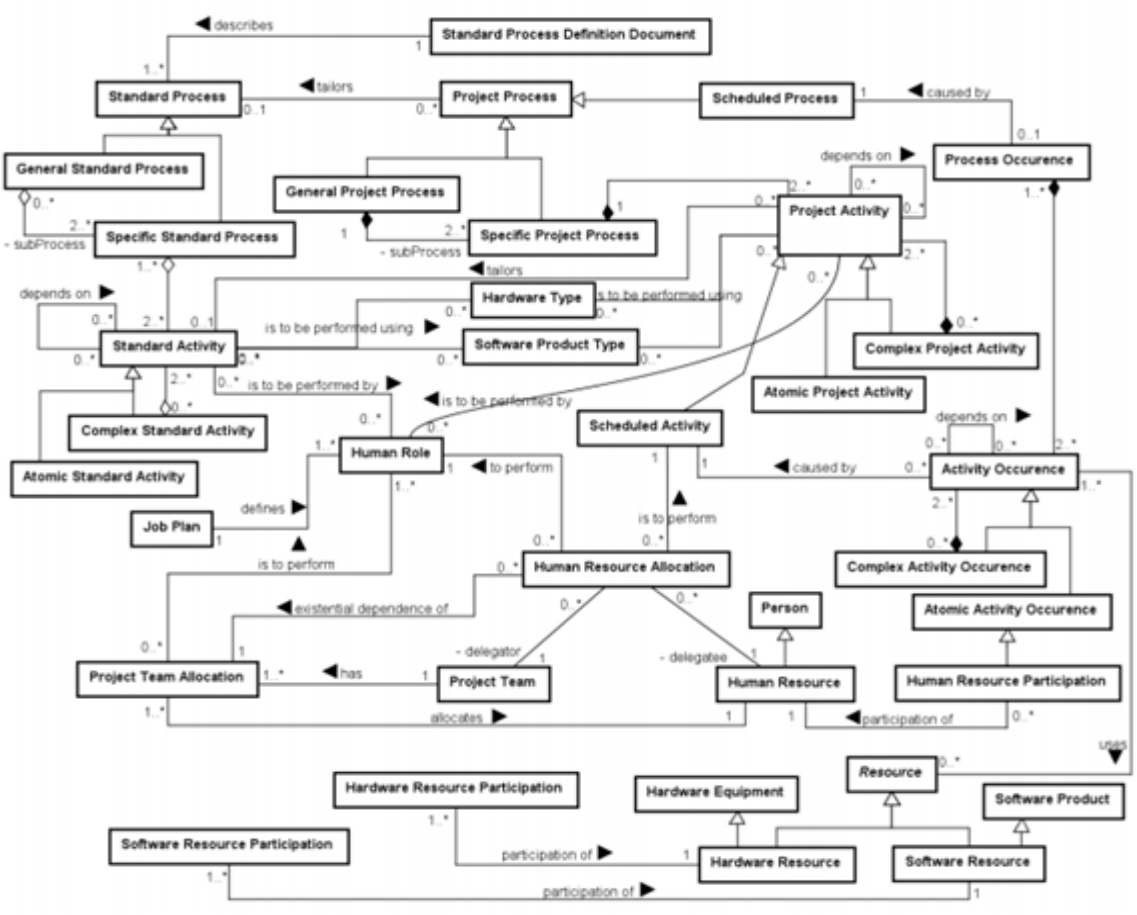

Figure 5.2 la version actuelle de SPO (Software Process Ontology)

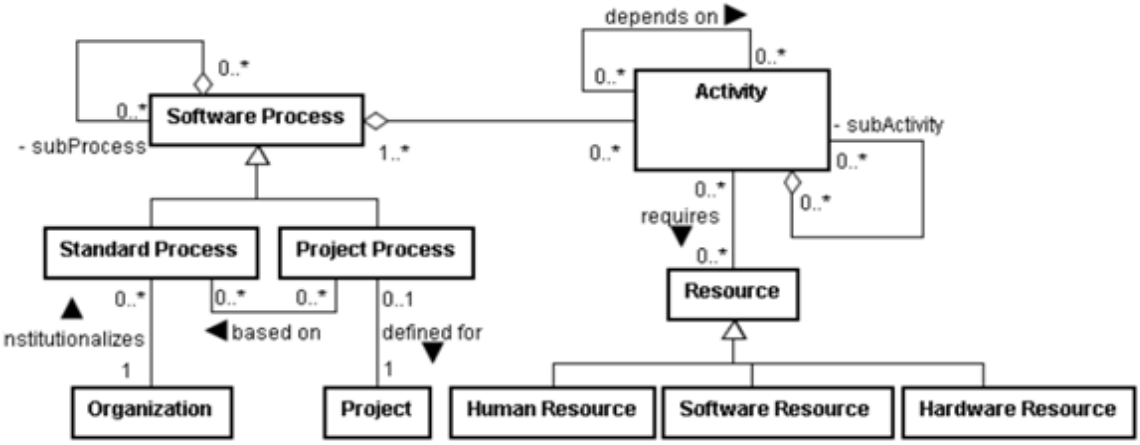

Figure 5.3 Vue partiel de SPO (version original)

VIZCAÍNO et al (VIZCAÍNO and al 2016) ont proposé l'ontologie O-GSD(Ontology Global Software Development) dans le but de promouvoir une compréhension commune du vocabulaire GSD basée sur un projet particulier appelé ORIGIN. Cette ontologie a atteint son but en facilitant la communication entre les membres de l'équipe de projet ORIGIN et en

aidant les membres de l'équipe à éviter les malentendus lorsqu'ils travaillent en groupe et à partager une vision similaire du projet. L'ontologie de GSD a finalement été structurée en trois sous-ontologies principales: les objectifs, les barrières et le projet. La description fournie pour chaque ontologie est divisée en deux parties principales: la description de la sous-ontologie elle-même, dans laquelle ses concepts et ses relations sont expliqués; Et la raison d'être de sa création dans laquelle nous répondons à la question "comment cette ontologie a-t-elle évolué?"

A. La sous-ontologie des objectifs

Cette ontologie identifie les objectifs / avantages que l'entreprise tente d'obtenir lors de l'utilisation de GSD. Elle tente d'indiquer les différents avantages, objectifs et buts que l'entreprise souhaite atteindre lors de l'utilisation de GSD.

B. La Sous-ontologie des obstacles GSD

La sous-ontologie des obstacles GSD est construite autour du concept principal de GSD obstacle comme le montre la figure 5.4, qui peut être divisé dans: le facteur humain, le facteur d'équipe; le facteur de gestion de projet l'auteur distingue également les différents types paramètres qui augmentent les barrières; Ceux-ci peuvent être en termes de emplacements géographique. En termes de temps, il y a une autre distance, appelée distance temporelle. En ce qui concerne les facteurs socio-culturels.

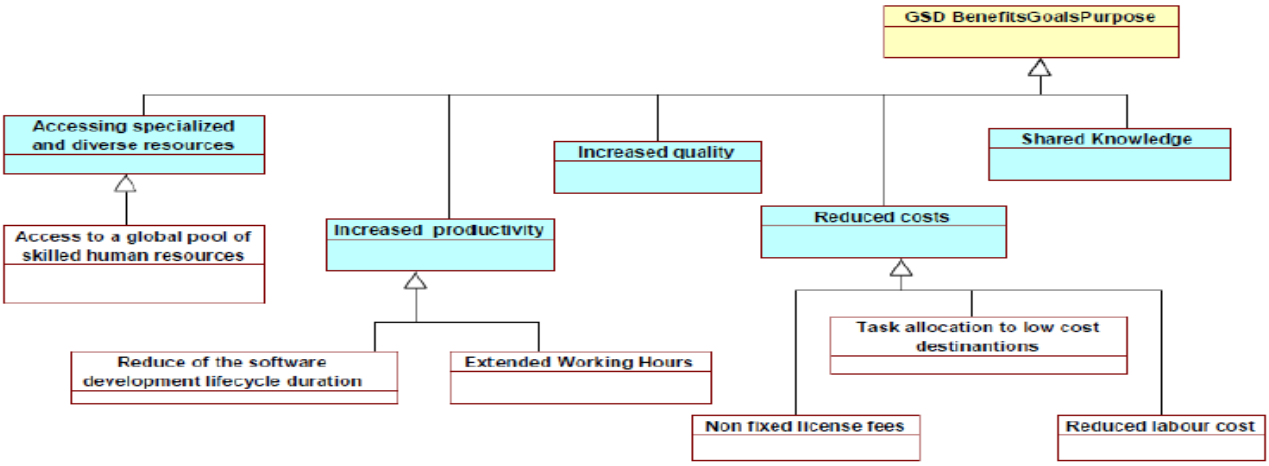

Figure 5.4 Vue partiel de la sous-ontologie obstacle

C. Sous-ontologie du projet GSD

La sous-ontologie principale est la sous-ontologie du projet GSD comme le montre la figure 5.5. Les concepts et les relations autour desquels cette sous-ontologie du projet ont été construites sont les suivantes:

• Un projet GSD peut couvrir plusieurs entreprises. Chaque entreprise coordonne au moins un sous-projet GSD. En outre, une entreprise peut participer au projet selon une

Stratégie d'approvisionnement particulière, en fonction de l'emplacement dans lequel elle se trouve

• Les projets GSD suivent une affectation de tâches qui pourrait être basée sur la division du produit (produit), du processus (basé sur le processus), du temps ou des fonctionnalités du logiciel. Pour être plus précis, une approche basée sur le produit signifie que le travail est divisé en modules de produits liés à au moins un artefact. Cependant, l'allocation des tâches pourrait également être fondée sur les processus, ce qui signifie que le travail serait divisé en processus GSD qui appartiennent à un élément de travail.

• Une équipe de projet GSD comprend un ensemble de membres de l'équipe qui travaillent sur un site particulier avec un lieu et un fuseau horaire; L'équipe comprend souvent un coordonnateur de projet. En outre, un membre de l'équipe joue un rôle de GSD qui peut être courtier (c'est le moyen de médiateur, de liaison culturelle ou d'ambassadeur culturel), le coordonnateur de projet et le chef d'équipe fonctionnel.

• Un rôle GSD exécute WorkItem, tandis qu'un WorkItem appartient simultanément à un processus GSD et il y a un artefact qui utilise ou produit un WorkItem.

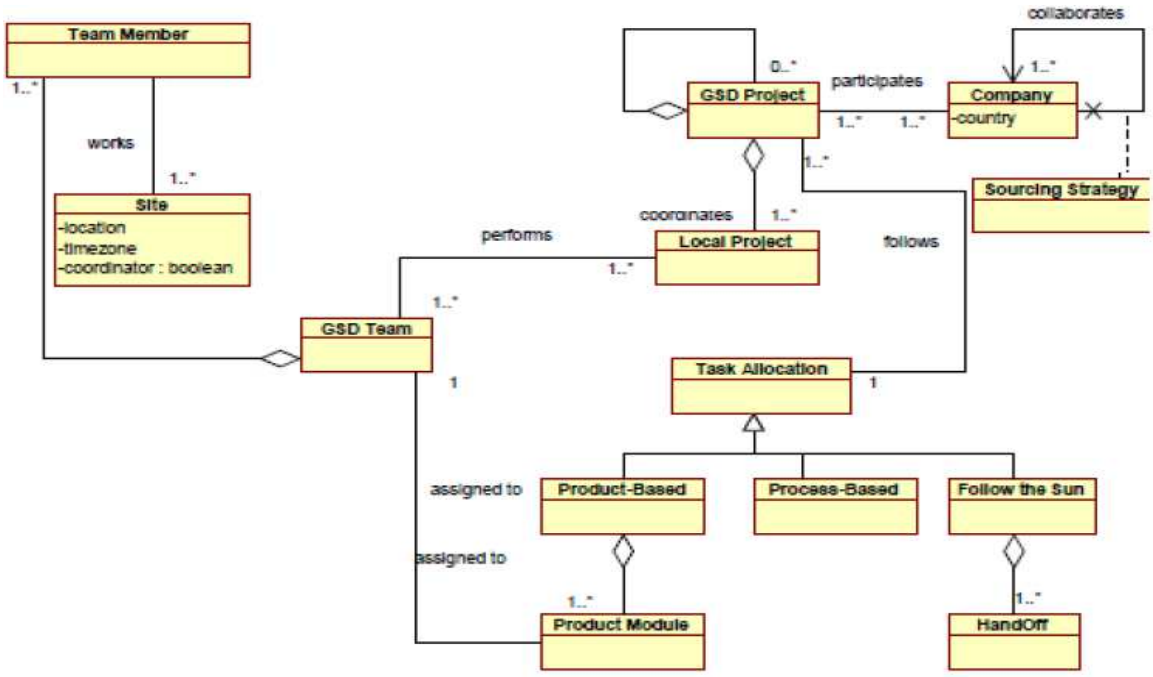

Figure 5.5 une Vue partiel de la sous-ontologie Projet

5.3 Les ontologies spécifiques

5.3.1 Les ontologies pour les méthodes AGILES

Ceravolo et al. (Ceravolo, Damiani et al. 2003) dispose d'un processus de développement de L'ontologie logiciel qui spécifie les concepts clés utilisés dans l'Extreme Programming (XP) et leurs propriétés. Le document traite l'utilisation de cette ontologie dans un processus d'acquisition sémantique et l'exploration de données sur les activités de l'équipe et des référentiels de contenu, visant à extraire de nouveaux concepts et d'identifier les facteurs cruciaux pour le développement agile. La définition de l'ontologie pour XP a été adoptée sur la base d'une approche top-down, définissant en premiers les concepts de domaine qui seront mis en évidence par un ensemble de classes, puis ils ont spécialisés ces classes pour obtenir un modèle de domaine complet. Trois classes principales ont été mise en évidence, à partir des quelles les concepts de la méthodologie XP sont regroupés à savoir: Rôle organisationnel (Organisation de Rôle) , Phase (Phase) , Produit

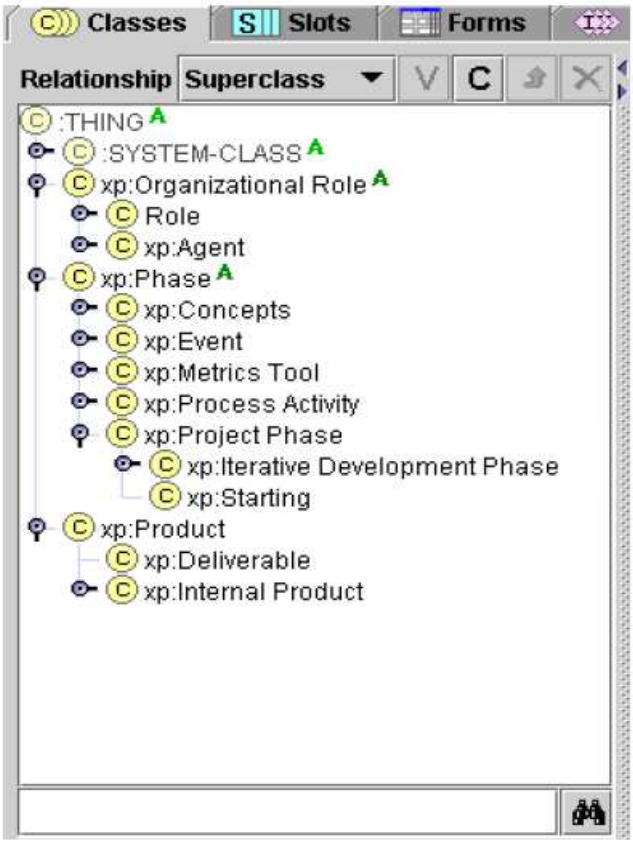

Figure 5.6 L'ontologie XP

(Fujita and Zualkernan 2008) ont présenté une ontologie pour les évaluation de Génération pour le processus Scrum. Le motif est de créer des évaluations pour une compréhension correcte d'un processus qui peut être utilisé dans un développement de logiciels comsociété.

Une proposition alternative pour les logiciels d'ontologies de processus a été faite par Parson (Parson 2011) , qui a conçu une ontology fondée sur une analyse d'un certain nombre de méthodes aussi agiles couramment utilisées comme Scrum(Ken 2004), XP(Beck and Fowler 2001), FDD(Palmer and Felsing 2001), etc.

Parson lui a fallu sept méthodes agiles et a tenté de résumer leur terminologie, et construire une ontologie initiale de méthodes agiles que les tentatives pour englober les différentes caractéristiques des méthodes couramment utilisées. Sur ce sens, son travail peut être considéré comme une ontologie générique processus agile. Progresser sur ses recherches, Parsons (Parsons 2011) traite le développement de logiciels orientés vers l'aspect et intégration des méthodes agiles. L'auteur a proposé une ontologie de développement basée sur une analyse des ontologies existantes de développement logiciel Orientée Aspect, une pro-ontologie basée sur des méthodes agiles, et une ontologie dérivée axée aspect sur le développement agile.

Lin et al. (Lin, Hilaire et al. 2012) ont développés l'ontologie K-CRIO qui est une ontologie dédiée à l'étude des organisations et à l'analyse organisationnelle des processus métiers qu'elles mettent en œuvre. Plus précisément, elle est utilisée pour comprendre, analyser et raisonner sur ces organisations. Les organisations visées sont celles composées d'acteurs humains impliqués tout au long de la conception de produits et, pour ce faire, organiser selon un processus métier. L'éventail de ce type d'organisations est assez large. L'ontologie K-CRIO se limite aux organisations qui produisent des logiciels comme objectif final du processus. Dans ce contexte, l'ontologie K-CRIO peut être utilisée pour modéliser la structure organisationnelle du processus et les activités qui en résultent. Cette ontologie peut ensuite être exploitée afin de concevoir des outils d'assistance à la mise en œuvre des processus ciblés au sein des organisations décrites. Plus précisément, l'ontologie fournit des moyens de raisonnement, d'annotation des ressources, et de suivi des processus de conception, permettant des recherches et de proposer pro-activement des conseils et des contenus appropriés. Afin d'illustrer l'utilisation de K-CRIO, l'auteur applique K-CRIO sur deux processus différents: le modèle en cascade et la méthodologie Scrum. Ces exemples sont des processus de développement de logiciels classiques. En outre, pour le processus Scrum, qui est un processus agile de développement de logiciel, largement utilisé dans les entreprises de logiciels, l'auteur a aussi conçu et développé un outil d'assistance intelligent. Cet outil contribue principalement à aider les Scrum Masters en leur fournissant des indicateurs pour

les assister dans leurs prises de décisions ainsi que par la constitution d'une base de connaissances sur les activités

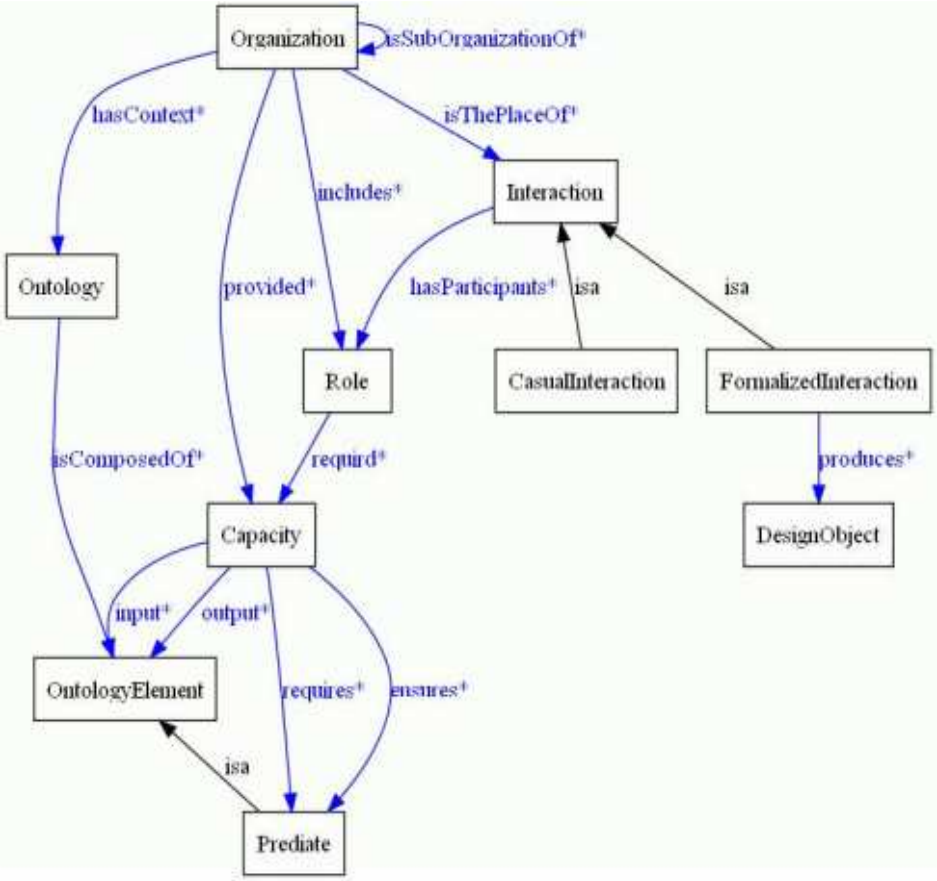

Figure 5.7 L'ontologie K-CRIO

Santana (Santana 2013)ont tenté de modéliser Scrum méthodologie de logiciels dans OntoSCRUM ontologie. Le PUR principale pose selon l'auteur de cette ontologie qui était de réutiliser des objets du projet et de partager des connaissances tacites au sein des organisations et des équipes de projet. Zualkernan (Zualkernan 2008)

Siddiqui et Alam (Siddiqui and Alam 2013)ont développé une ontologie pour Feature Driven Development(FDD) du cycle de vie qui peut être utilisée pour le développement de modèle d'application à la conception et mise en œuvre en vedette. Ces caractéristiques sont précisément définies dans le modèle de domaine basé sur OWL. Transition d'OWL basée sur un modèle de domaine qui présente la liste qui est directement définie dans des règles de transformation.

De même, Valaski et al. dans (Valaski, Malucelli et al. 2011)ont développé une ontologie dite OntoRUP ayant pour but de classer le matériel d'apprentissage en génie logiciel. A cet effet,

ils ont combiné le guide du génie logiciel (SWEBOK)(Bourque and Fairley 2014) et le Rational Unified Process (RUP). SWEBOK a été utilisé pour définir la structure de la zone de connaissances en génie logiciel, tandis que RUP a été utilisé pour définir les axiomes qui repèrent les relations entre les concepts et envoyer pour permettre le raisonnement à SWEBOK dans les domaines de connaissances.

5.3.2 Les ontologies pour CMM / CMMI

Liao et al. (Liao, Qu et al. 2005) Software Process Ontology (SPO) les auteurs analysent deux modèles de processus de logiciels différents, tels que Capability Maturity Model (CMM) et ISO / IEC 15504 pour définir les concepts clés dans le processus de logiciels (qui sont ensuite mis en correspondance d'un à un autre). L'analyse permet la définition d'un cadre modèle processus de logiciel basé sur l'ontologie qui englobe un processus logiciel ontologie (SPO), qui définit le modèle de processus à un niveau conceptuel. Le SPO est utilisé pour construire deux ontologies différentes à remplir pour les deux modèles analysés. L'utilisation de la SPO et ses extensions consiste à permettre à un utilisateur de vérifier la description des processus et des pratiques recommandées par le modèle de référence.

Capability Maturity Model Integration (CMMI) est un modèle de référence de processus qui a été développé par le Soft Institut de génie logiciel (SEI) de Carnegie Mellon University. CMMI porte sur le développement et la maintenance d'activités non appliqués aux produits et aux services. Ce modèle peut être utilisé pour l'amélioration des processus, et mesurer la capacité d'un processus ou la maturité d'une organisation. Composants CMMI (y compris un modèle, son matériel de formation et les documents d'évaluation connexes) sont conçus pour répondre aux besoins de certains domaines spécifiques d'intérêts, qui est appelé constellation. Il ya trois constellations, soutenue par le cadre: CMMI Développement(Team 2002) , CMMI pour les services, et CMMI pour l'acquisition.

Il ya seulement quelques études sur CMMI ontologie dans la littérature.

Soydan et Kokar (Soydan and Kokar 2006)présentent une formalisation qui capture les définitions d'un certain nombre de concepts de CMMI-DEV et les relations entre ces concepts. La formalisation est exprimée dans un langage formel, OWL. Les deux principaux objectifs de cette formalisation devaient être compatibles avec le modèle CMMI-DEV et opérationnel, à savoir, pour permettre une détermination automatique d'un niveau de maturité des processus de développement basé sur des données sur les pratiques au sein d'une organisation donnée. La formalisation présente les concepts dans un certain niveau de détail

des plus généraux aux plus spécifiques. Une justification de la sélection des concepts et des relations est donnée. Pour évaluer la validité de la formalisation, un certain nombre de cas de test pour le scénario de la détermination automatique du niveau de maturité ont été développés. Le Raisonneurs OWL génériques est ensuite utilisé pour calculer les niveaux de maturité. Alors que les résultats des tests étaient tous positifs. La valeur réelle de cette formalisation vient du fait qu'il capture fidèlement les principaux aspects de CMMI-DEV Dans "Une ontologie pour CMMI-ACQ modèle,

Sharifloo(Sharifloo, Shamsfard et al. 2008)ont introduit une ontologie développée pour représenter les connaissances du domaine CMMI-ACQ. CMMI-ACQ et l'un des modèles de la collection CMMI qui fournit aux organisations des ressources et des pratiques efficaces pour soutenir l'amélioration des processus d'acquisition. Cette ontologie a été développée sur la base de SUMO ontologie de haut niveau. Le but principal de développer l'ontologie CMMI-ACQ est de l'utiliser dans un outil qui évalue automatiquement le niveau de maturité de l'organisation, et aide à l'amélioration de ses processus. Les interfaces de l'outil récupèrent les données sur l'état d'avancement des pratiques génériques et spécifiques de l'utilisateur autour de l'organisation. Une partie de cet outil code les données recueillies dans un fichier SUOKIF. Après avoir attaché ce fichier à la base de connaissances de l'outil, l'outil indique le niveau de maturité actuel de l'organisation et le niveau de chacun des domaines du processus. À ce stade, l'utilisateur indique un état souhaitable et l'outil se charge de trouver l'écart entre l'état actuel et l'état souhaité, et présente ensuite une liste de pratiques nécessaire améliorer les performances. Ce processus serait très fastidieux et source d'erreurs s'il a été effectué manuellement. En outre, l'utilisateur peut poser des questions différentes à l'outil, qui codent les questions en requêtes pour le moteur de l'ontologie qui récupèrent les réponses connexes.

Lee et al (Lee and Wang 2009) ont proposé une ontologie d'aide à la décision basée sur des agents intelligents ed (OIDSA) appliquée à la surveillance et le contrôle du modèle (CMMI). Le travail de l'agent est fondé sur l'analyse et l'extraction d'informations à partir d'ensembles de données linguistiques. L'OIDSA est composé d'un agent de traitement de langage naturel, un agent d'inférence floue, et un agent d'aide à la décision. L'agent d'inférence floue calcule la similitude entre le rapport d'avancement prévu et les progrès réels, basé sur l'ontologie CMMI, l'ontologie personnelle du projet, et les résultats de traitement du langage naturel. Les résultats fournis par le OIDSA soutiennent une évaluation de la performance des membres du projet par le gestionnaire de projet. "L'évaluation montre que l'agent aide à la décision

intelligente basée sur des ontologies (OIDSA) pour la surveillance et le contrôle du projet CMMI peut effectivement évaluer le pourcentage atteint du progrès pour réduire l'effort humain et les coûts du projet(Gómez-Pérez, Ramírez et al. 2007).

Gazel et al. (Gazel, Sezer et al. 2012) présente un outil d'évaluation des processus logiciels basé sur l'ontologie qui a été développée pour soutenir la phase de collecte des données pour l'évaluation des processus et du suivi de la conformité des processus de logiciels CMMI comme modèle de référence. L'outil OCMQT (Ontology-based CMMI Mapping and Querying Tool)a été développé comme un plug-in en un outil de gestion des processus open-source, à savoir « EPF Composer" qui est une réalisation de l'ingénierie des processus méta-modèle SPEM. L'étude explique aussi les résultats d'un exemple d'utilisation de la OCMQT dans une organisation de développement système et logiciel. Dans l'outil OCMQT, on toujours besoin de connaissances d'experts. En fait, les activités d'amélioration des processus et d'évaluation nécessitent toujours des experts. Cependant, OCMQT peut aider des utilisateur non-experts a faire des erreurs involontaires dans une organisation.

5.3.3 Les ontologies pour SPEM

Le premier travail se propose de représenter SPEM dans la logique de description (DL) (Wang, Jin et al. 2006). Le travail a pour but de créer deux cartographies. La première de MOF vers DL et la deuxième cartographie de la contrainte OCL (OMG, 2006b) de SPEM à DL. L'objectif est de représenter le métamodèle SPEM basé sur MOF avec DL et à partir de cette dernière représenter des contraintes OCL supplémentaires qui complètent le métamodèle de SPEM avec une sémantique supplémentaires. Le résultat de ces correspondances est un métamodèle formelle de SPEM dans DL. Le travail présente en outre un exemple simple de raisonneur avec la représentation SPEM DL d'un projet. L'exemple couvre un processus de livraison de système d'information.

Le travail de Rodriguez (Rodríguez and Sicilia 2009) propose une définition de contrainte de processus de SPEM avec des règles sémantiques SWRL. L'auteur indique que la plupart des termes spem peuvent être directement convertis en OWL. En outre, il affirme que la traduction ne vise pas à se substituer au modèle original, mais il sert de complément pour ajouter le raisonnement et le soutien de l'inférence des modèles basés sur SPEM. Une conception du prototype d'un moteur pour la vérification de contrainte de processus est présenté. L'idée principale est de vérifier la cohérence entre les contraintes du processus de SWRL qui peuvent être obtenus à partir du Compositeur EPF et SWRL contraintes de

processus à partir d'un plan de projet. En d'autres termes le travail a l'intention de vérifier un plan de projet avec un processus de SPEM utilisant SWRL. Notez que le compositeur EPF permet de sérialiser modèles SPEM avec XMI.

Líška et Navrat (Líška and Navrat 2010)ont présenté une approche qui utilise des processus logiciels Meta-Modèle (SPEM) qui sont ancrées dans les approches de l'ingénierie des connaissances. Ils montrent comment les meta-modèle SPEM peuvent être utilisés dans la Web sémantique espace technique tout en développant une ontologie de SPEM. Par conséquent, ils utilisée comme projet de génération de plans et de vérification

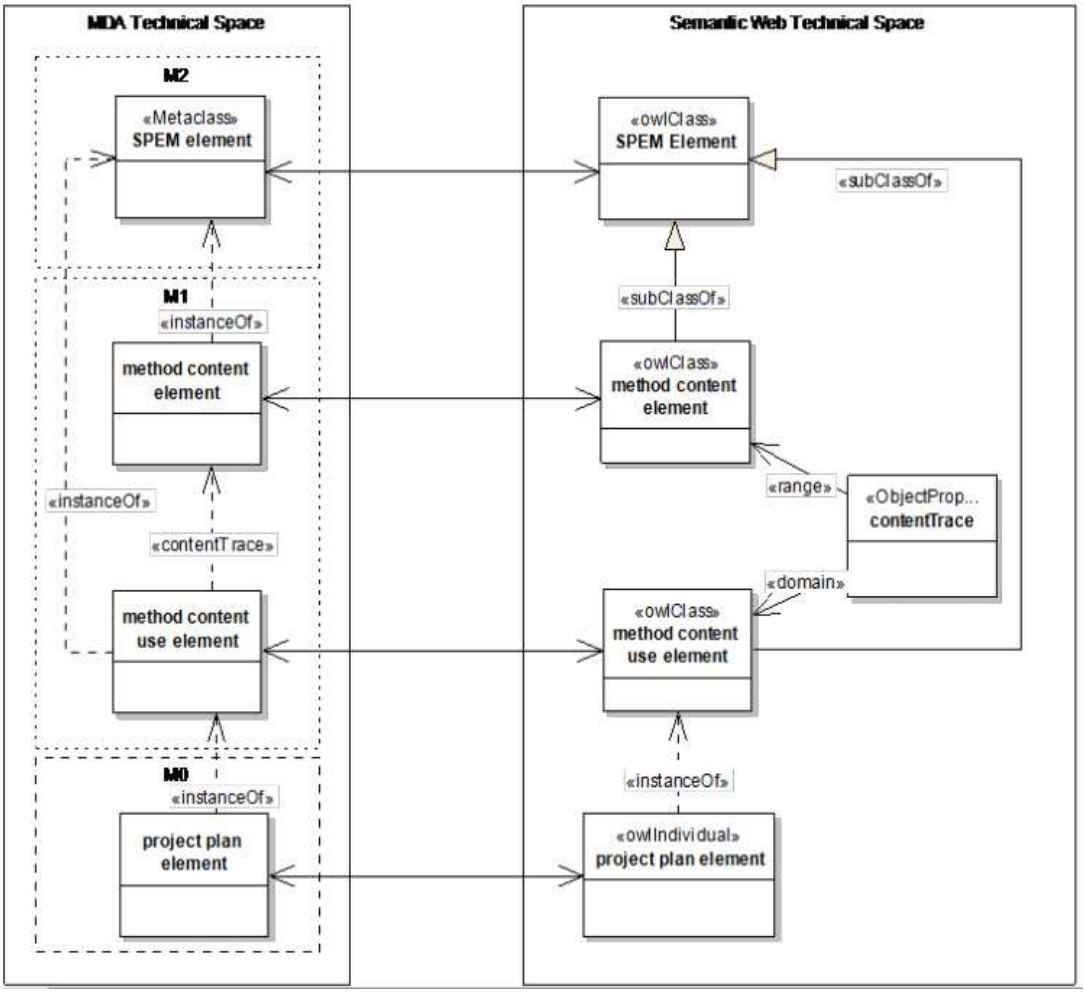

Figure 5.8 L'ontologie SPEM

Fadila AOUSSAT (Aoussat, Oussalah et al. 2011) propose une nouvelle approche de réutilisation de procédés logiciels à base d'architectures logicielles : AoSP (Architectures oriented Software Processes). Cette approche se base sur les insuffisances des approches existantes, elle tente de tirer des avantages des solutions offertes par les architectures logicielles tout en respectant les spécificités des procédés logiciels (centré humain, différents

types d'exécutions...etc.).Pour prendre en compte l'expérience et le savoir-faire des développeurs, cette approche exploite une ontologie de domaine à partir du metamodèle spem permettant de capitaliser les connaissances procédées logiciels et incluant des mécanismes d'inférence pour déduire de nouvelles connaissances procédés logiciels. Pour l'auteur le métamodèle SPEM regroupe des concepts concernant un large éventail de PLs, il est aussi un standard adopté par l'OMG et un métamodèle référence dans le domaine de la modélisation des PLs.

5.3.4 Les ontologies pour les normes ISO

La plupart des modèles et des normes sont créés de façon indépendante, sans nécessairement partager la même sémantique. Cela donne souvent lieu à des incohérences entre eux. Ce problème s'est amplifié lorsque des normes différentes sont utilisés ensemble, causant des problèmes d'interopérabilité sémantique(Ruy, Falbo et al. 2014). Même les normes proposées par la même organisation ont ce problème. Cela est dû au fait que chaque norme définit ces propres champs d'application, ces structures, ces termes et définitions (Pardo, Pino et al. 2012).

L'Organisation internationale de normalisation (ISO) reconnaît ce problème. Les normes élaborées par le SC7 de l'ISO/IEC (le sous-comité de l'ISO responsable des normes logiciels et des systèmes d'ingénierie) emploient fréquemment des termes dont les définitions varient considérablement entre une norme et une autre. Afin de traiter ce problème, l'ISO a créé, en 2012, un groupe d'étude pour développer une infrastructure ontologique visant à être une référence cohérente pour toutes les normes de SC7 (Henderson-Sellers, McBride et al. 2013, Henderson-Sellers, Gonzalez-Perez et al. 2014). L'objectif est d'établir un ensemble de définition de base d'ontologies, qui peuvent être utilisés pour dériver DES ontologies plus spécifiques. Ces ontologies spécifiques sont destinés à répondre aux différents sous-domaines SE (par exemple, validation de logiciel), qui sont à leur tour l'objet de normes de SC7 spécifiques (par exemple, ISO / IEC 29119). L'initiative de l'ISO est un travail en cours, qui a porté sur les ontologies de définition, en prenant essentiellement la norme ISO / CEI 24744 (ISO/IEC (2007)) en compte. L'objectif est de un framework à couches comprenant un réseau de cinq ontologies dont trois hupper ontologie(deo ,cdo et afos) et deux ontologie specifiques (sdo et) . Dans le haut niveau du framework proposé, on trouve l'ontologie DEO(the Definitional Elements Ontology), qui a pour but de fournir les définitions des concepts, et les contraintes qui dictent la façon dont ils doivent être liés. A partir de l'ontologie DEO, on peut definir l'ontologie CDO (Configured Definitional Ontology) , la CDO donne une structure

aux concepts utilisés et nécessaires dans un domaine spécifique. La seule CDO étant travaillé à ce jour est un CDO pour la norme ISO / IEC 24744 métamodèle (Software Engineering Métamodèle pour les méthodologies de développement - SEMDM). De la CDO, les ontologies spécifiques ont des normes particulières, appelées standard de domaine ontologies (SDO), qui peuvent être dérivés. Le framework considère également à l'avenir, d'étendre DEO en considérant les distinctions ontologiques avancées par ontologies fondamentales. Cette extension est appelée avancée Foundational Ontologie de normalisation (AFOS). Et enfin il a été proposé de créer un diagramme de relation de normalisation (SRO) comme une ontologie de domaine de ces normes produites. SEMDM est la principale base de l'ensemble du framework, fournissant la sémantique pour toutes les normes ISO / SC7. Toutefois, pour la réussite de cette initiative, la cohérence de cette base ontologique est cruciale.

Une ontologie de base pour l'initiative d'harmonisation ISO devrait être: (i) suffisamment souple pour permettre à des ingénieurs de l'ontologie à explorer d'autres modèles dans la conception d'ontologies spécifiques pour les différents sous-domaines de processus du logiciel; (ii) modulaire, afin de permettre à l'ingénieur de l'ontologie pour sélectionner les fragments pertinents de l'ontologie pour pouvoir les réutiliser; et (iii) suffisamment large pour couvrir les concepts généraux dans l'univers du discours des processus logiciels. Pour la réalisation de ces caractéristiques

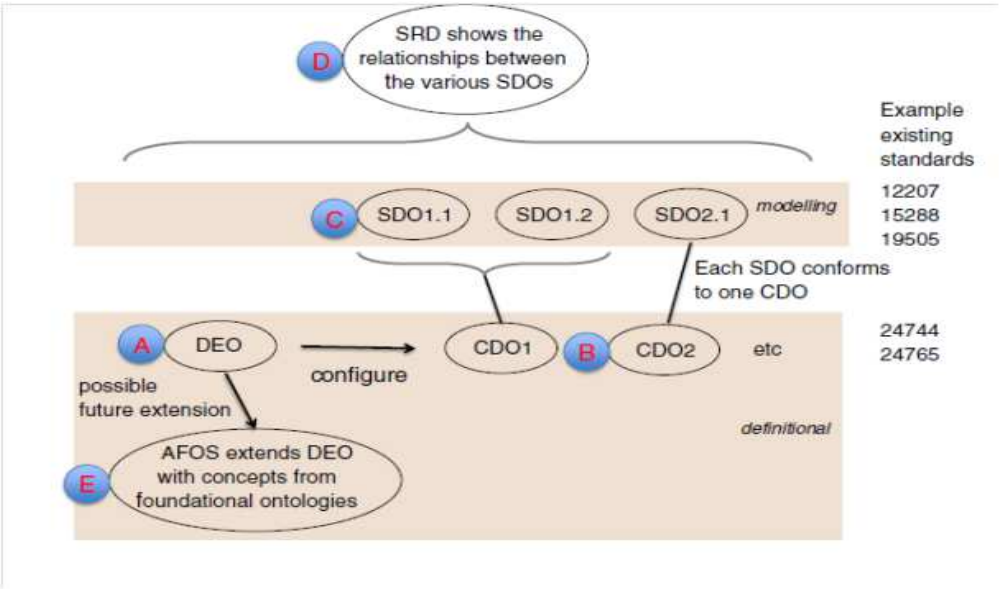

Figure 5.9 La relation entre les cinq ontologies ISO

En 2014 Ruy (Ruy, Falbo et al. 2014) présente une analyse ontologique du métamodèle (SEMDM), fourni par la norme ISO / IEC 24744. Dans la même optique de l'initiative de

l'ISO qui vise à utiliser SEMDM en tant que source d'une ontologie à des fins d'harmonisation ces normes, cette analyse ontologique peut aider à assurer la sémantique nécessaires de cet objectif. Cette analyse est effectuée à la lumière de l'Foundational Ontologie Unifiée (UFO). Comme résultat, l'auteur présente quelques-uns des problèmes identifié dans SEMDM, ainsi que des fragments de modèles alternatifs.

Sur la même lancé Ruy et al (Ruy, Falbo et al. 2015) soutiennent l'idée que : en suivant une approche orientée modèle, une ontologie de base peut devenir plus modulaire et extensible. Une ontologie de base pour l'initiative d'harmonisation ISO devrait être: (I) suffisamment souple pour permettre à des ingénieurs d'ontologie d'explorer d'autres modèles dans la conception d'ontologies spécifiques pour les différents sous-domaines de processus du logiciel;(Ii) modulaire, afin de leur permettre de sélectionner les fragments pertinents. Pour résoudre les problèmes de l'ontologie pour leur réutilisation ; et (iii) suffisamment large pour couvrir les concepts généraux dans l'univers des processus logiciels. Pour la réalisation de ces caractéristiques, l'auteur propose que le noyau ontologie devrait être organisé comme une ontologie Pattern Language (OPL). Un OPL est un réseau de motifs d'ontologies interconnectées relatifs aux domaines qui fournit un soutien pour résoudre une classe de problèmes de développement de l'ontologie pour un domaine spécifique.

Un OLP offre un ensemble de modèles de domaines interdépendants, un ensemble de directives explicites sur des problèmes qui peuvent survenir dans ce domaine, et la façon de résoudre ces problèmes, en suggérant un ou plusieurs motifs pour résoudre chaque problème spécifique.

Le tableau 5.1 montre une synthèse des travaux cités dans ce chapitre, Pour ce faire, nous les avons classés selon septs critères de base à savoir :1) nom de l'ontologie et son auteur, 2) son type (générique ou spécifique) ,3) type de processus ,4) son domaine d'application ,5) son objectif ,6) le langage dans là quel elle a était défini, 7) l'existence d'un outil qui la support

Comme continuité aux travaux (Henderson-Sellers, McBride et al. 2013, Henderson-Sellers, Gonzalez-Perez et al. 2014)qui a fourni une approche globale pour l'harmonisation des normes ISO, Gonzalez-Perez et al (Gonzalez-Perez et al,2016)propose un processus de raffinement de l'ontologie DEO(Definitional Elements Ontology)qui englobe le noyau des concepts qui sont Pertinents en génie logiciel par Exemple, Processus, produit logiciel ou équipe. Avec une définition minutieuse et l'identification des relations inter-concept, cet ensemble de base constitue la base d'une ontologie abstraite du domaine qui ne fournit pas de

détails précis sur aucun Standard international. Ce processus de raffinement doit aboutir à une ontologie CDO (Configured Definitional Ontology) qui est spécifiquement orientée vers un but ou une norme particulière. Ce processus passe par trois étapes

a) L'élimination des concepts non recherchés : Étant donné que la DEO doit être exprimé à un niveau d'abstraction très élevé, elle contiendra probablement des concepts qui couvrent un large éventail de sous-domaines de l'ingénierie logicielle. Il est donc très peu probable qu'un CDO particulier ait besoin de prendre en considération tous ces concepts.

b) Ajout d'éléments au CDO : Comme étant très abstrait, la DEO est supposée contenir que des concepts de haut niveau et donc généraux. Il est donc probable que des définitions sont nécessaires pour des CDO spécifiques qui ne sont pas présentes dans le DEO. Dans les cas Comme celui là, il peut être nécessaire de créer une nouvelle classe pour la CDO créé. Cependant, il est important de noter qu'une telle introduction ne doit pas contredire toute sémantique existante dans la DEO. Afin de s'assurer que tel est le cas, les ajouts doivent être effectués par spécialisation (est-un-genre-de) d'un concept DEO existant.

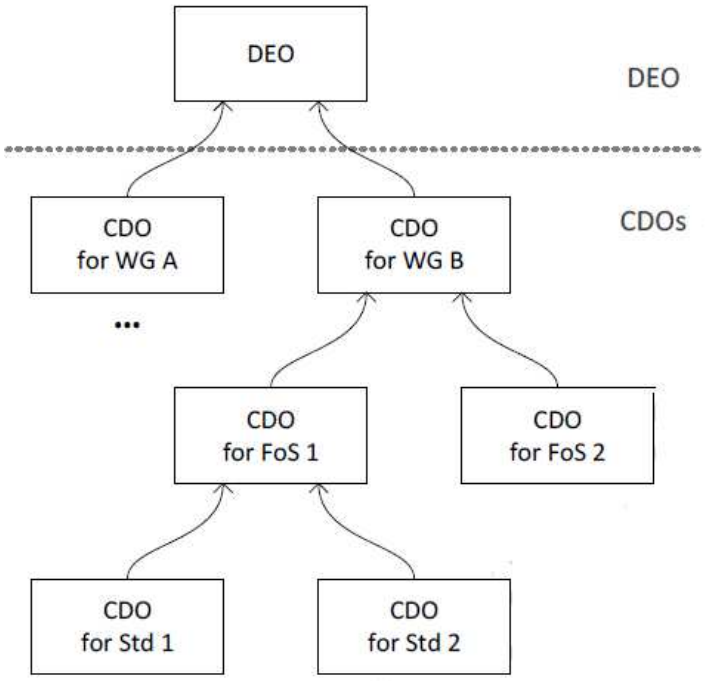

Figure 5.10 Les étapes du processus de raffinement

c) Chaînage des CDOs : Dans le contexte du SC7, les normes internationales appartiennent souvent aux «familles» (c-à-d. L'adoption de normes thématiquement liées), de sorte que les

CDO sont similaires pour chaque membre de la famille. Ainsi, le raffinement hiérarchique (par spécialisation - comme ci-dessus) peut créer un ensemble cohérent de CDO. La figure 5.10 montre une adaptation du DEO. D'abord dans deux CDO correspondant à deux groupes de travail (WG) de SC7. Ensuite, au sein de chaque groupe de travail, il peut y avoir des collections cohérentes de normes qui forment des familles de normes de groupe intra-travail (FoS). Un raffinement supplémentaire peut alors conduire à des CDO pour les normes individuelles au sein de chaque famille. On appelle ici le "chaînage des CDO", par lequel la spécificité et le détail sont ajoutés progressivement et parallèlement à la structure organisationnelle où les ontologies seront utilisées. En tout point de la hiérarchie, les CDO maintiennent une relation claire avec les concepts incorporés dans le DEO, avec des détails plus ou moins compliqués. La figure 5.10 montre les étapes de ce processus de raffinement.

Tableau 5.1 récapitulatif des ontologies des procédés logiciels

Auteur/nom de l'ontologie	Type d'ontologie	Type de PLs	Domaine	Objectif	Langage de description	Outils
Wille et abran(Wille, Abran et al. 2003)	Génrique	Non spécifié	SWEBOK KA	Structuration hiérarchique pour le concept de qualité logiciel,	XML	Non
Mendes and Abran (Mendes and Abran 2005)	Génrique	Non spécifié	SWEBOK	Modélisation de la discipline de génie logiciel	Graphe	Non
Sicilia et al.(Sicilia, Cuadrado et al. 2005) /Onto-SWEBOK	Génrique	Non spécifié	SWEBOK	Évaluation de l'ontologie basée sur swebok	OWL	Non
Falbo and Bertollo(Falbo and Bertollo 2009, Falbo 2010) / SPO	Génrique	Non spécifié	Standard software processes	modèle pour aider les organisations de développement de logiciels à utiliser les normes dans leurs efforts d'amélioration de processus logiciel	UML	Non
Falbo et al. /OLP-SP(de Almeida Falbo, Barcellos et al. 2013)	Génrique	Non spécifié	Standard software processes	Une ontologie qui peut être réutilisé par d'autres ontologies	UML	Non
(VIZCAÍNO et al 2016)	Génrique	Non spécifié	Standard software processes	vocabulaire GSD basée sur un projet particulier appelé ORIGIN	UML	Non
Ceravolo et al. (Ceravolo, Damiani et al. 2003)/Extreme Programming Ontology (XPO)	Spécifique	Semi-Formels	XP	Analyse et l'exploitation des processus agiles afin d'identifier les facteurs critiques dans le développement logiciel agile.	OWL	Non cite
Parson (Parson 2011, Parsons 2011)	Spécifique	Semi-Formels	SCRUM, XP, FDD	Résume la terminologie, et construit une ontologie initiale de méthodes agiles qui tente d'englober les différentes caractéristiques des méthodes couramment utilisées.	OWL	Non cite
Lin et al. (Lin, Hilaire et al. 2012) /K-CRIO	Spécifique	Semi-Formels	SCRUM	Compréhension, analyse et raisonnement sur les organisations.	OWL	Oui
Siddiqui and Alam (Siddiqui and Alam 2013)	Spécifique	Semi-Formels	FDD	Développement de modèle d'application à la conception et mise en œuvre	OWL	Non cite
Valaski(Valaski, Malucelli et al. 2011)	Spécifique	Semi-Formels	RUP,SWEBOK	classification du matériel en génie logiciel. Utilisé dans la methode RUP	OWL	Non cite
Liao et al. (Liao, Qu et al. 2005)	Spécifique	organitionnel	CMMI, ISO/IEC 15504	Représentation du modèle de processus.	OWL	Non cite

Soydan (Soydan and Kokar 2006)	Spécifique	Organitionnel	Cmmi dev	Modélisation du modele CMMI-SW.	OWL	Oui
Sharifloo et al. (Sharifloo, Shamsfard et al. 2008)	Spécifique	Organitionnel	CMMI for Acquisition	Évaluation du niveau d'une organisation de la maturité et propose des recommandations.	OWL/SWRL	Oui
Lee and Chen (Lee, Wang et al. 2006, Lee and Wang 2009)/OIDSA	Spécifique	Organitionnel	CMMI	Système d'aide à la décision pour le suivi et le contrôle de la capacité d'intégration du modèle de maturité projet.	OWL	Oui
Gazel et al (Gazel, AKCAPINAR SEZER et al. 2012).	Spécifique	Organitionnel	CMMI	Évaluation des processus et de suivi de la conformité des processus CMMI comme le modèle de référence de processus.	OWL	OCMQT
Wang(Wang, Jin et al. 2006)	Spécifique	descriptive	SPEM	Validation d'un processus SPEM	Logique description	Non cite
Rodríguez (Rodríguez and Sicilia 2009)	Spécifique	descriptive	SPEM	Vérification de la cohérence d'un processus spem	OCL/OWL/ SRWL	Non cite
Líška and Navrat (Líška and Navrat 2010)	Spécifique	descriptive	SPEM	Démonstration sur la façon dont le spem peut être utilisé pour la planification du projet.	OWL DL/SWRL	Non cite
Aousset (Aoussat, Oussalah et al. 2011)	Spécifique	descriptive	SPEM	Reutilisation de processus	OWL	Non cite
Ruy (Ruy, Falbo et al. 2014)	spécifique	descriptive	ISO/IEC 24744	Une analyse de la norme ISO/IEC 24744 à la lumière de l'UFO	UML	Non
Henderson(Henderson-Sellers, McBride et al. 2013, Henderson-Sellers, Gonzalez-Perez et al. 2014)	Spécifique	descriptive	Toutes les normes ISO	Définition du projet SC7 et l'établissement du fondement théorique pour la creation des ontologie DOE CDO		
ruy(Ruy, Falbo et al. 2015)	Spécifique	descriptive	Toutes les Normes ISO	Définition des fragments d'ontologie dans l'objectif d'harmonisation des normes ISO	UML	Non
Hamri et Benslimane(Hamri et Benslimane, 2015)	Spécifique	descriptive	ISO/IEC 24744	Représentation du modèle de processus.	OWL	OUI
Gonzalez-Perez et al, 2016)	Spécifique	descriptive	ISO/IEC 24744, 24765, 12207, 15288 et 33061	Référencement sémantique de toutes les normes ISO pour génie logiciel	UML	Non

5.4 Synthése & Conclusion

Cette recherche systématique a montré que le problème le plus important dans l'utilisation des ontologies dans le domaine de la gestion de projet logiciel est la normalisation. Le grand nombre de méthodologies de gestion de projet et de modèles de processus de logiciel crée un environnement extrêmement difficile à comprendre et à utiliser. Les raisons qui contribuent à cette complexité sont l'existence de différentes normes qui se chevauchent à la fois dans les domaines de la gestion de projet et dans les domaines de l'ingénierie logicielle. Assez souvent basée sur des ontologies SWEBOK ou PMI comprenant des centaines de concepts et de termes. En outre, le grand nombre de modèles de processus logiciel rend difficile le partage des connaissances. Deuxièmement, dans de nombreux cas étudiés les ontologies sont expérimentales ce qui implique qu'elles ne sont pas utilisées dans des cas réels où elles ne couvrent pas tous les domaines de la gestion de projet mais seulement quelques phases du processus logiciel, a l'instar du groupe SC7 d'ISO qui donne une définition abstraite d'une ontologie appelé DEO basée sur la norme ISO/IEC 24744 mais sans donner une description de cette ontologie ni la façon de la concevoir.

Cette étude confirme qu'un grand nombre d'ontologies ont été développées dans les domaines de la gestion de projet logiciel et du génie logiciel. Nous avons effectué une revue systématique de la littérature et nous nous sommes concentrés sur les ontologies qui abordent le processus de gestion de projet et sur les ontologies comme support de développement (là ou les ontologies sont utilisés comme un outil de soutien pour les développeurs). Vue le nombre important d'ontologies existantes, nous avons choisi d'examiner seulement les ontologies de génie logiciel générique basées sur des processus d'ingénierie de logiciels bien établis et bien connues (par exemple RUP et CMMI) ou pratiques (par exemple XP et Scrum). Ce choix a été fait à partir un très grand nombre d'ontologies qui ont été développées pour répondre à des sous-domaines spécifiques de l'ingénierie logicielle (par exemple la gestion des exigences, analyse et conception). Dans de nombreux cas, il existe des chevauchements qui rendent extrêmement difficile la recherche de catégorisation des ontologies dans les catégories disjointes.

Cette étude a aussi permis de constater le manque d'une ontologie de référence dans le domaine de la gestion de projet logiciel. Cette ontologie devrait être et bien mise en place afin d'augmenter la probabilité de l'adoption accrue et de l'utilisation dans les cas de la vie réelle.

Dans ce chapitre nous cité les travaux récents qui couvrent le domaine des ontologies des procédés logiciels avec un positionnement de notre approche par rapport aux travaux connexes. Dans le chapitre suivant nous allons aborder l'approche que nous avons proposée pour la création d'une ontologie pour la norme ISO/IEC 24744.

6 Chapitre VI Architecture ontologique à base de MDA pour la norme ISO/IEC 24744

6.1 Introduction

Dans la majorité des cas, la description des méta-modèles est faite de façon manuelle ce qui peut générer des ambiguïtés ou des déficiences dans les modèles. Dans le but d'améliorer la qualité de notre méta-modèle, nous avons exploré les principes ontologiques. Ces principes jouent un rôle important notamment pour supporter le raisonnement sur les modèles. En effet, le besoin d'adresser plusieurs perspectives en utilisant le même modèle nécessite de raisonner sur l'ensemble du référentiel du modèle (i.e., éléments et relations) pour donner du sens au modèle selon une préoccupation donnée (perspective). Plus spécifiquement, on donne du sens (enrichissement sémantique) au modèle en raisonnant sur les relations entre les éléments structurant le modèle. Pour ce fait, nous avons utilisé, sur une vue purement organisationnelle, les principes pour la construction des ontologies vu qu'on gère au niveau de notre méta-modèle des concepts et les relations entre ces concepts.

En outre, le Model Driven Architecture (MDA) devient un paradigme important pour le développement des applications. Il peut être appliqué dans plusieurs domaines, étant donné sa capacité à réduire la complexité du processus de développement. MDA et son architecture à quatre couches fournit une base solide pour définir le métamodèle de n'importe quel langage de modélisation. Il offre donc une fondation pour rapprocher l'ingénierie de logiciels et des méthodologies comme UML aux technologies du Web Sémantique basées sur RDF et OWL. Ainsi, une fois qu'un langage de modélisation du Web Sémantique comme OWL modélisé en MOF, ses utilisateurs peuvent adopter les capacités de MOF pour la création, la gestion de modèles, la génération de code et l'interopérabilité avec des modèles MOF. ODM définit cinq métamodèles (RDFS, OWL, Topic Maps, Common Logic et Description Logic), deux profils UML (le profil RDFS/OWL, le profil Topic Maps) et un ensemble de correspondances QVT, de UML à OWL, Topic Maps à OWL et RDFS/OWL à Common Logic.

Dans cette lignée, nous avons défini une approche de type MDA, permettant de générer l'ontologie de domaine à partir du metamodele de la norme OSO/IEC24744:(Hamri et Benslimane, 2015).

Dans la première section, nous commencerons par décrire l'architecture globale de notre approche. Ensuite, nous détaillerons chaque modèle et la manière avec laquelle il est intégré dans l'approche. Finalement, nous présenterons les différentes étapes à suivre pour passer d'un niveau MDA à un autre, jusqu'à la génération de l'ontologie finale.

6.2 Architecture de construction de l'ontologie ISO/IEC 24744

Bien que les avantages présentés par le méta-modèle SEMDM soient indéniables(Henderson-Sellers and Gonzalez-Perez 2006), l'utilisation de celui-ci peut, parfois, s'avérer intimidante. La divergence des approches conventionnelles et l'adoption de nouveaux concepts constituent un changement de paradigme qui implique une nouvelle façon de penser et de percevoir le processus de méta-modélisation. Ainsi, la courbe d'apprentissage peut être importante. Aussi, le fait que SEMDM soit spécifié en termes de concepts purement abstraits sans offrir une notation spécialisée permettant de modéliser les méthodes de façon simplifiée, rend le processus de conception et de validation de méthodes plus difficile. L'utilisation des ontologies comme référence sémantique peut être une bonne solution pour aider les utilisateurs à dépasser ces problèmes.

Malgré tous les avantages de la norme ISO/IEC 24744 par rapport aux approches OMG tel que SPEM dans le domaine du génie logiciel, il n'en demeure pas moins que MDA possède un savoir-faire et une expérience dans le domaine de l'ingénierie ontologique à base de modèle. De plus, le problème de dualité de l'approche MDA ne se pose pas ici du moment que notre objectif est de créer une ontologie de domaine qui permet de définir les concepts de la norme et leurs relations. Ajoutez à cela le fait que la norme utilise des diagrammes de classes UML(OMG 2011a) à but de définir ses métamodèles et facilite l'utilisation de l'approche MDA et de ses outils largement disponibles.

6.2.1 Architecture de modélisation des ontologies dans un processus MDA

Notre approche s'inspire des travaux de(Djurić, Gašević et al. 2005. Hamri, 2012) pour la création d'ontologie dans un processus MDA dans ses étapes de transformation. Avant de commencer la transformation proposée par Djuric, nous extrayons trois méta-modèles séparés correspondants respectivement aux concepts Endeavour, Template, et Ressource extraits à partir de la spécification du méta-modèle de la norme. Ensuite, nous créons un profil UML pour la norme qui regroupe trois packages correspondants à chacun des trois méta-modèles extraits. Ce profil UML sera utilisé comme point de départ pour notre processus de transformation basé sur MDA. Quant à la validation de l'ontologie OWL de domaine,

plusieurs outils de validation existent dans la littérature. Dans notre travail nous utilisons l'éditeur ontologique Protégé, OPPS.

L'architecture que nous proposons dans ce travail et qui se base sur les langages du Web sémantique (OWL, RDFs, etc.) comporte figure 6.1 (Hamri et Benslimane, 2015) :

Un Profil UML pour la norme ISO/IEC 24744 (UPISO) ;

Un Profil UML d'ontologie pour ISO(OUPISO) ;

Un méta-modèle de définition d'ontologie (ODM (OMG 2014a)) ;

ODM représente le trait d'union entre MDA d'une part et le Web sémantique et le génie logiciel d'autre part. Ce qui fait de lui le noyau de cette architecture (Hamri et Benslimane, 2015).

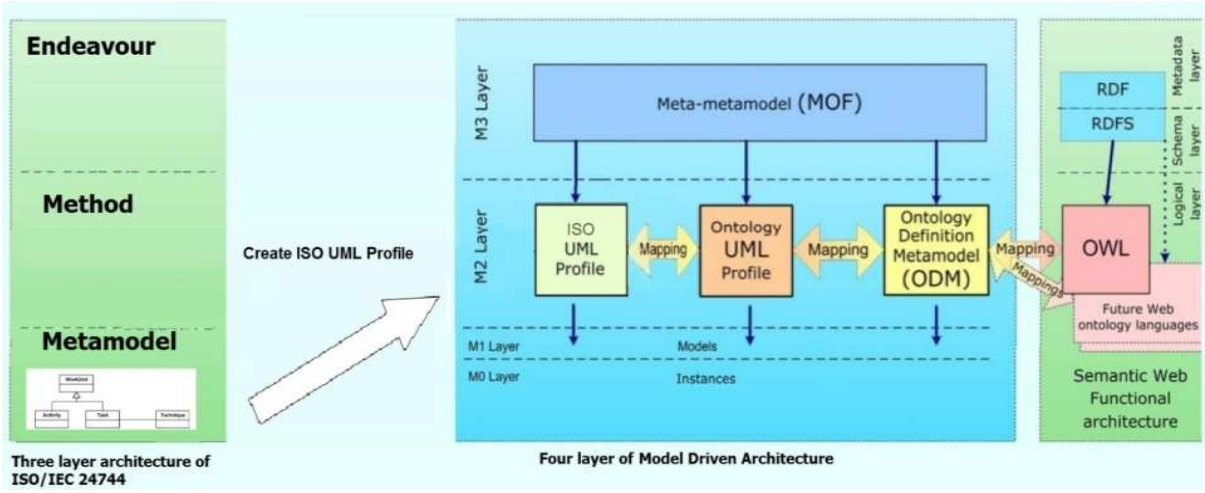

Figure 6.1 Architecture ontologique à base de MDA.

Notre architecture est donc, une architecture ontologique basée sur MDA qui inclue un Profil UML pour la norme ISO/IEC 24744 (UPISO). Pour construire ce Profil, nous choisissons d'abord, le package qui contient l'ensemble de stéréotypes qui nous permettent de projeter ce Profil vers le méta-modèle cible (PSM) qui peut être OWL. Ensuite, nous exploitons les ontologies pour modéliser les aspects sémantiques de la norme pour définir un Profil UML d'ontologie qui sera traduit vers un méta-modèle de définition d'ontologie (ODM). Ce méta-modèle regroupe tous les concepts ontologiques. Il est défini selon le MOF (Meta-Object Facility) (OMG 2013) et sa construction est basée sur le langage OWL.

Une fois l'ODM construit, il est traduit vers le méta-modèle OWL, pour générer une description OWL. Par conséquent, la passerelle entre MDA et le Web sémantique est créée via l'utilisation d'ODM qui offre plusieurs avantages tels que la flexibilité et l'ouverture. En nous focalisons uniquement sur le méta-modèle OWL ; nous allons proposer dans la section suivante, notre approche pour construire une ontologie de domaine.

6.2.2 Approche à base de MDA pour la construction d'une ontologie de domaine ISO /IEC24744

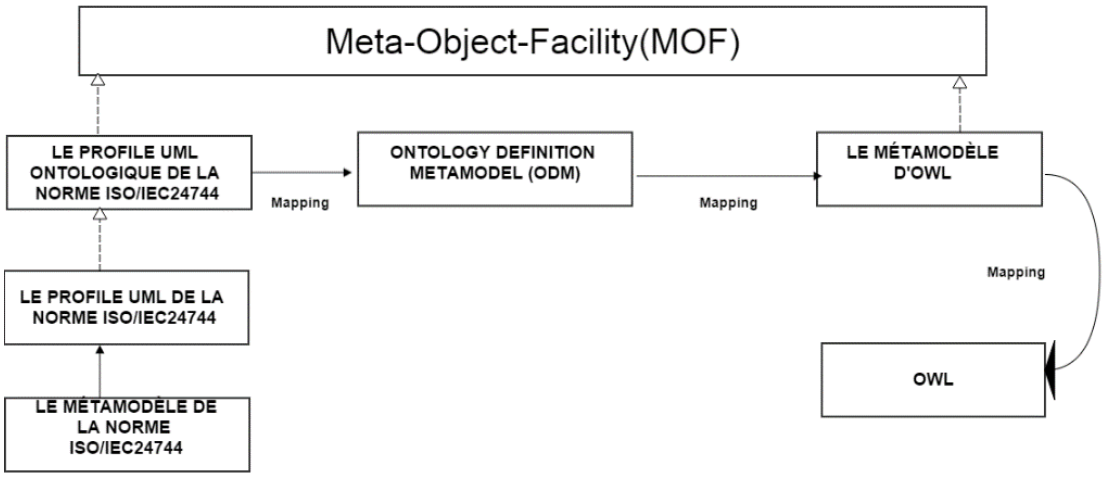

Figure 6.2 Étapes de construction d'une ontologie de Domaine ISO /IEC24744

La figure 6.2 donne une vue globale sur les étapes à suivre durant la construction de notre ontologie. La base de notre architecture est la construction des profils UML de la norme ISO/IEC 24744 à savoir un profil UML pour Endeavour, Template, et Resource. Pour pouvoir générer automatiquement notre ontologie OWL, nous procédons d'abord par l'extraction des informations relatives aux concepts Endeavour, Template et Resource à partir de la description du méta-modèle de la norme.

Dans notre extraction, nous utilisons les outils UML Plug-in sous la plate-forme Eclipse (Eclipse, 2015) pour créer un Endeavour profil qui aura pour source le méta-modèle Endeavour de la norme, Template profil qui aura pour source le métamodèle Template de la norme et Resource profil qui aura pour source le métamodèle Resource de la norme. Une fois les profils créés, ils seront stéréotypés avec des concepts de la norme que nous appelons UPISO (UML Profil de la norme ISO) qui sont en fait trois profils UML.

Pour générer une ontologie de domaine décrite en OWL, nous avons utilisé des mappings entre les concepts d'UPISO, d'OUPISO, d'ODM et d'OWL. Pour la validation de cette

ontologie générée, nous avons utilisé l'outil Protégé. La figure 6.3 montre le processus pour la génération de l'ontologie ISO/IEC 24744 (Hamri et Benslimane, 2015).

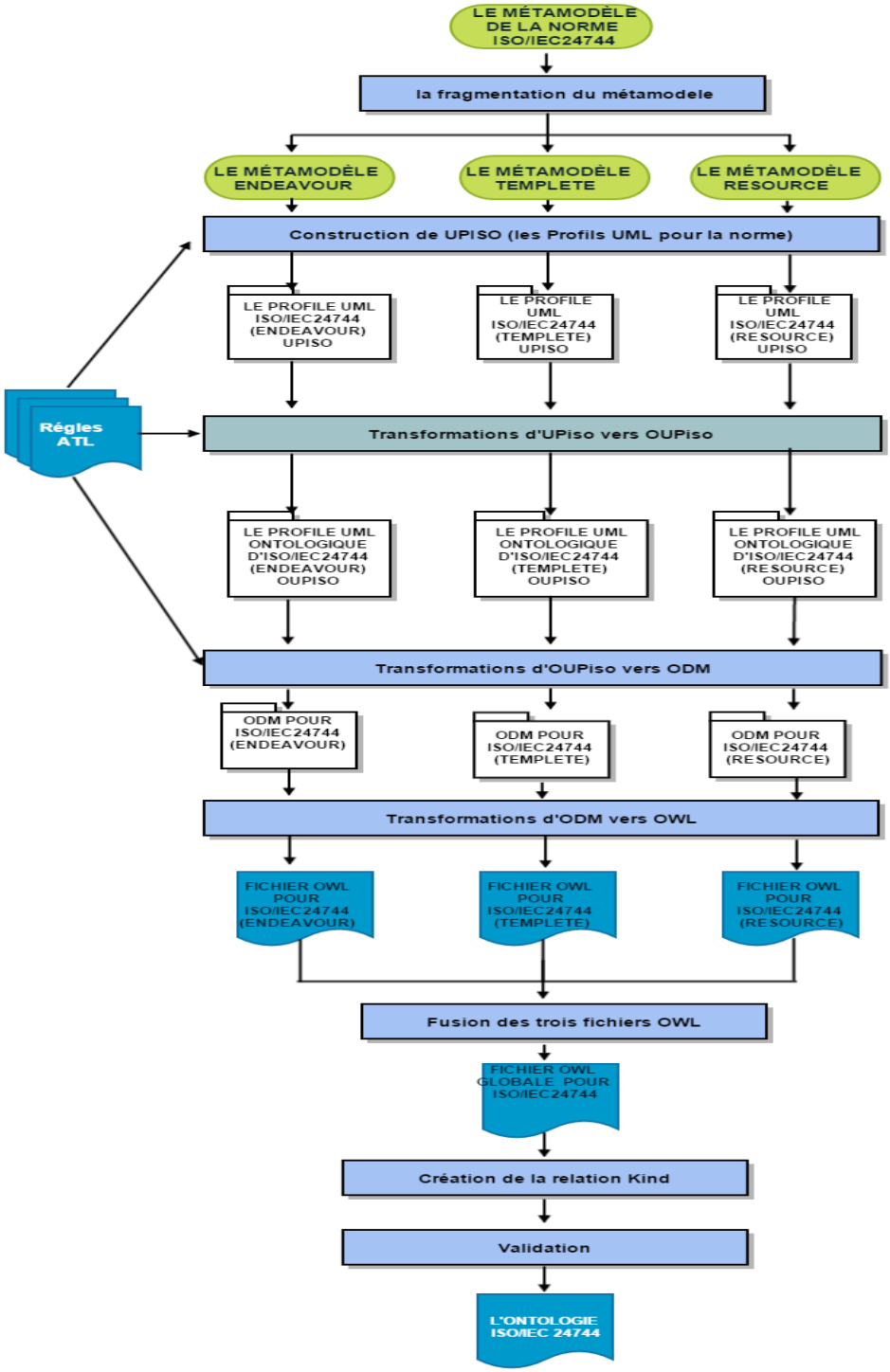

Figure 6.3 Processus de génération d'une ontologie de domaine ISO /IEC24744

Ce processus comporte huit étapes pour la génération d'une ontologie de domaine ISO/IEC24744 :

Étape 1 : La fragmentation du méta-modèle.

Cette étape permet l'extraction des informations relatives aux concepts Endeavour, Template et Resource de la description de la norme. Durant cette étape, nous sélectionnons les différentes parties du méta-modèle relatives aux trois aspects du domaine décrie dans la norme à savoir Endeavour, Templates et Resources.

Étape 2 : Construction de l'UPISO (les Profils UML pour la norme).

Après la fragmentation, nous utilisons l'outil UML (Eclipse Modeling Tools et le Plugin Papurus) pour créer trois Profils UML avec des stéréotypes qui incluent les constructeurs de la norme. Dans cette étape, nous obtenons trois profils : un Profil Endeavour (Endeavour Profil), un Profil les Templates (Templates profil) et enfin un Profil pour les Resources (Resources Profil).

Étape 3 : Transformations de l'UPISO vers OUPISO.

Dans cette étape, nous transformons ces trois profils UPISO vers leurs profils OUPISO correspondants : un Profil UML d'ontologie pour le Profil Endeavour (Endeavour Ontology UML Profil) ; un Profil UML d'ontologie pour le Profil les Templates (Templates Ontology UML profil). La conformité du méta-modèle ISO/IEC 24744 avec MOF(Meta-Object Facility) de l'architecture de MDA, du fait que sa description est basée sur UML permet de générer d'abord un document XMI (XML Metadata Interchange). Ensuite, nous employons le langage ATL (ATLAS Transformation Language) pour transformer la représentations XMI source vers sa représentation cible correspondante. Les règles de transformations utilisées pour l'étape 3 sont définies dans la table 6.2. Une fois, ces règles sont exécutées, nous aurons enfin, trois représentations de XMI pour les trois fragments du méta-modèle source

Étape 4 : Transformations de l'OUPISO vers ODM.

À ce niveau, les trois Profils OUPISO sont transformés vers les méta-modèles de définition d'ontologies (ODM) correspondants. Puisque tous les méta-modèles sont conformes au MOF (Meta-Object Facility) de l'architecture de MDA, nous générons d'abord des documents XMI (XML Metadata Interchange) pour ces méta-modèles (des formats XMI pour UML,

OUPSWS et pour ODM). Par la suite, nous utiliserons le langage ATL (ATL) pour faire la transformation des représentations sources vers leurs représentations cibles correspondantes. Dans la section implémentation nous citerons les règles de transformations utilisées dans cette partie. Une fois ces règles exécutées, nous aurons les trois représentations de XML pour les méta-modèles ODM.

Étape 5 : Mapping du méta-modèle ODM vers une description OWL.

Le but de cette étape est de générer une description OWL de la norme. Cette étape n'est pas vraiment difficile puisque les concepts ODM sont quasi identiques à ceux d'OWL. Les règles de transformations utilisées sont définies voir la Table 6.3. Les règles de transformation sont définies dans la partie 6.3.2

Étape 6 : Fusion des descriptions OWL

La sixième étape consiste à regrouper les trois documents OWL correspondants respectivement aux profils Endeavour, Template et Resource pour créer un seul fichier OWL complet correspondant à l'ontologie de domaine ISO/IEC 24744. Cette étape va se faire d'une manière manuelle.

Étape 7 : Création des relations Kind

Cette étape consiste à établir le lien entre les concepts « element » et « element kind » par la création d'une relation non taxonomique Kind. Elle a été implémentée à l'aide de Jena, une API java qui permet de créer et manipuler des ontologies en OWL. Dans cette étape, nous parcourons l'ontologie et pour chaque concept de la sous ontologie Endeavour nous créons la relation Kind avec le concept correspondant de la sous ontologie Templete. (par exemple, le concept Workunit sera relier avec WorkunitKind).

Etape 8 : Validation de l'ontologie.

L'étape de validation de l'ontologie permet de détecter des erreurs de syntaxe ou d'éventuelles incohérences dans la description finale de l'ontologie OWL. Les documents XMI sont exportés vers l'outil Protégé pour leur validation.

6.3 Construction de profils et transformation vers OWL

Le profil est la pierre angulaire de notre travail car il permet le passage vers l'espace MDA dont la suite du travail repose sur lui. Ce Profil UML est utilisé pour modéliser les aspects sémantiques de la norme. Un groupe d'extensions d'UML constitue un Profil UML. UML offre trois mécanismes d'extension pour étendre les éléments du méta-modèle de base d'UML: les Stéréotypes qui sont des classes du profil qui définissent comment une méta-classe existante peut être étendue dans le cadre d'un profil. Les valeurs étiquetées (ou tagged values) introduisent les propriétés et les contraintes qui jouent le rôle de restriction sur les nouveaux éléments en respectant les éléments de base.

Aussi, nous utilisons un Profil UML pour marquer le modèle indépendant de la plate-forme, appelé PIM (Platform Independent Model). Pour faciliter sa transformation vers le méta-modèle cible PSM (Platform Specific Model), qui est dans notre cas le méta-modèle OWL, nous aurons besoin par la suite de ce PSM pour la génération de notre ontologie de domaine dans le langage ontologique OWL. Cette phase aura comme résultat trois profils Endeavour, Template et Resource. Le plus important de ces profils c'est Endeavour profil, qui sera composé des trois principaux éléments à savoir Produit, Producteur et Processus. Cependant, pour générer une ontologie OWL de la norme ISO/IEC 24744, nous avons adopté une méthodologie comportant plusieurs étapes que nous décrivons dans la section suivante.

6.3.1 Construction des Profils UML pour (UPISO Endeavour)

Les activités de conception d'un profil sont intimement liées à celles de la création d'un langage. Que la syntaxe concrète ou de surface du langage, soit : un modèle profilé, un modèle conforme à un méta-modèle conforme au MOF. La réussite de cette activité est déterminante pour le succès de tout futur développement de système et requiert un processus de conception clair. Il est commun d'identifier trois activités de conception que Sjouke Mauw et al. (Mauw, Wiersma et al. 2002) caractérisent ainsi : 1) identification du domaine précise les entités qui décrivent un domaine, 2) identification de l'espace de problème sélectionne un sous-ensemble des éléments du domaine qui vont être disponibles pendant la résolution d'un problème et 3) définition du langage caractérise la manifestation du langage (syntaxe, sémantique, pragmatisme).

La première chose à faire durant cette étape est la construction du package qui regroupe l'ensemble des stéréotypes de la norme ISO/IEC 24744. Ces stéréotypes correspondent aux concepts de la norme. La figure 6.4 montre une capture d'écran du package des stéréotypes Endeavour éléments.

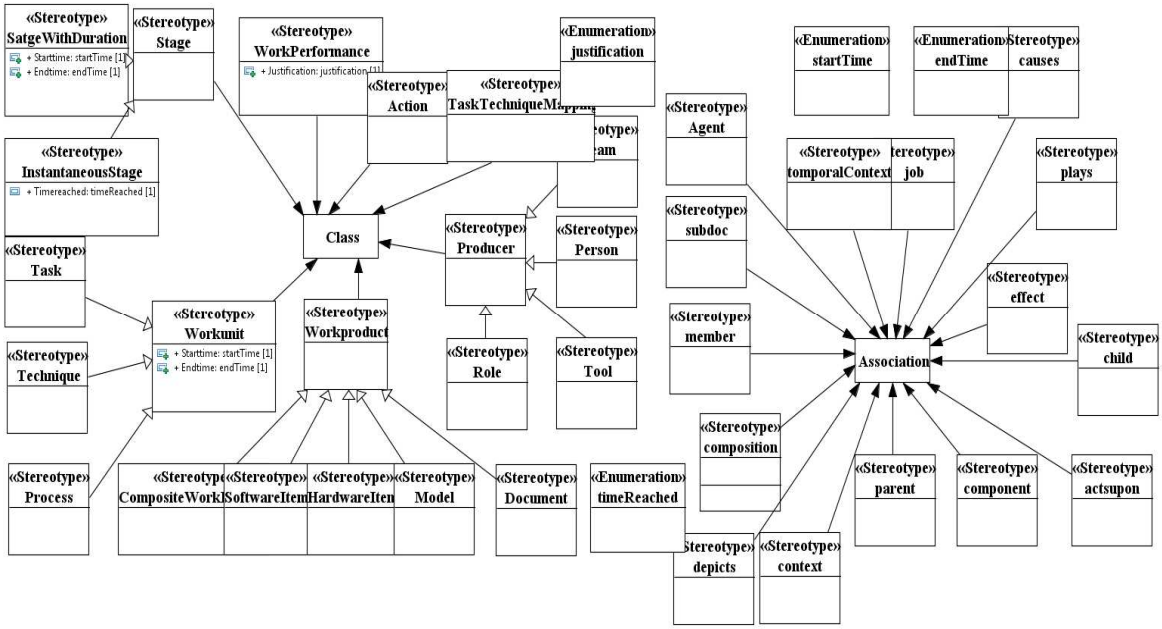

Figure 6.4 **Package des stéréotypes Endeavour elements**

Le tableau 6.1 présente les différents stéréotypes du profil ISO endeavour. Dans la première colonne, nous présentons chaque stéréotype du profil. Dans la deuxième nous présentons la description informelle de chaque stéréotype, La troisième colonne présente la métaclasse de base de ce stéréotype. La quatrième et cinquième colonnes présentent respectivement d'éventuelles valeurs marquées et contraintes sur chaque stéréotype.

Tableau 6.1 liste des stéréotypes du profil UMl UPISO(Endeavour)

Stéreotype	Description sémantique	metaclasse	Valeur marqué	contrainte
Stage	représente un bloc de temps (*managed time frame*) dans un projet en spécifiant la structure temporelle globale de la méthode.	Class	//	
Stagewith-duration	Est un stage dont la durée de temps est déterminée au préalable	class	Startime Endtime	
Producer	Le stéréotype *Producer* représente l'aspect producteur qui définit les entités (i.e. personnes, groupes, rôles, etc.) responsables de l'exécution des unités de travail (*WorkUnit*)	Class	//	
Software-Item	Instance de product, produit de type logiciel	Class	//	
Hardware-Item	Instance de product, produit de type materiel	Class	//	
Workproduct	Le stéreotype *WorkProduct* représente un type spécifique de produits (documents, matériels ou collections d'informations) caractérisée par la nature de son contenu et l'intention de son utilisation.	Class	//	Un produit fait toujours l'objet d'une ou de plusieurs actions.
Instantaneoustage	Stage instantané est un point de gestion en temps au sein d'une entreprise.instance de stage	Class	Time-reached	
Workunit	Une unité de travail est caractérisée par son but et son niveau de capacité minimale. C'est cette dernière caractéristique qui détermine le niveau minimal auquel il devient judicieux d'exécuter l'unité de travail.	Class	Starttime Endtime	
Process	Le stéreotype Process représente une activité grande en taille, qui s'exécute à l'intérieur d'un domaine d'expertise particulier	Class	//	

Task	Le stéreotype Task représente des petites unités de travail qui servent à implémenter le Process. La relation d'agrégation entre les unités de travail (WorkUnit) et les tâches (Task) permettent de définir récursivement des unités de travail jusqu'à atteinte du niveau de détail désiré	Class	//	
Technique	Le stéreotype Technique représente une unité de travail, petite en taille, qui se concentre sur la manière d'implémenter une tâche (Task). Une technique est caractérisée, dans SEMDM, par son but dans le projet.	Class	//	
TaskTechnique-Mapping	Ce stéreotype représente le fait qu'une certaine technique est utilisée pour accomplir une certaine tâche	Class	//	
Rôle	Ce stéreotype représente un ensemble de responsabilités qu'un producteur peut assumer dans un projet.	Class	//	Un rôle est joué par un seul producteur et un producteur peut jouer un ou plusieurs rôles
Team	Le stéreotype Team représente un groupe de producteurs qui se concentrent collectivement sur des unités de travail communes.	Class	//	
Tool	Le stéreotype Tool représente les outils et les mécanismes aidant les autres types de producteurs à accomplir leurs tâches d'une manière automatisée.	Class	//	
Workperformance	Le stéreotype Workperformance permet d'établir le lien entre les unités de travail (WorkUnit) et les producteurs (Producer) responsables de leur production	Class	Justification	
Model	Le stéreotype Model est utilisé pour donner une représentation abstraite de la syntaxe abstraite, de la syntaxe concrète et de l'architecture.	Class	//	
Compositework-Product	Le stéreotype CompositeWorkProduct représente le fait qu'un artefact (WorkProduct) est composé d'autres artefacts.	Class	//	
Document	Le stéreotype Document est utilisé pour représenter les documents gérés durant un projet	Class	//	
Action	Le stéreotype action est un événement d'utilisation effectué par une tâche sur un produit du travail.	Class	//	
Agent	Association qui relie le sterotype Workperformance avec le stérotype Producer	association	//	

Causes	Association qui relie le sterotype Task avec le stérotype Action	Association	//	Inverse de l'association Effect
Tomporelle context	Association qui relie le sterotype Process avec le stérotype Stagewithduration	Association	//	
Job	Association qui relie le sterotype Workunit avec le stérotype Workperformance	Association	//	
Plays	Association qui relie le sterotype Producer avec stérotype Role	Association	//	
Effect	Association qui relie le sterotype Action avec le stérotype Task	Association	//	Inverse de l'association causes
Child	Association qui relie le sterotype Process avec lui-même	Association	//	Inverse de l'association Parent
Parent	Association qui relie le sterotype Process avec lui-même	Association	//	Inverse de l'association child
Context	Association qui relie le sterotype workunit avec le stérotype task	Association	//	
Depicts	Association qui relie le sterotype document avec le stérotype workproduct	Association	//	
Actsup	Association qui relie le sterotype action avec le stérotype workproduct	Association	//	
Composition	Association qui relie le sterotype Tasktechniquemapping avec task et technique	Association	//	

Une fois le package construit, nous construirons trois Profils UML (UPISO) correspondants à la spécification modulaire de la norme concernant les trois parties. Pour construire ces profils UML, nous devons d'abord, sélectionner les stéréotypes correspondants à chaque partie Endeavour, Templates et Ressources à partir du package. Une fois élaboré, chaque Profil UML est alors sauvegardé dans son package correspondant. Puisque les transformations sont nombreuses, nous nous contentons uniquement de présenter quelques étapes de construction et de transformation de ces Profils UML. La Figure 6.5 montre le profil UPISO (Endeavour).

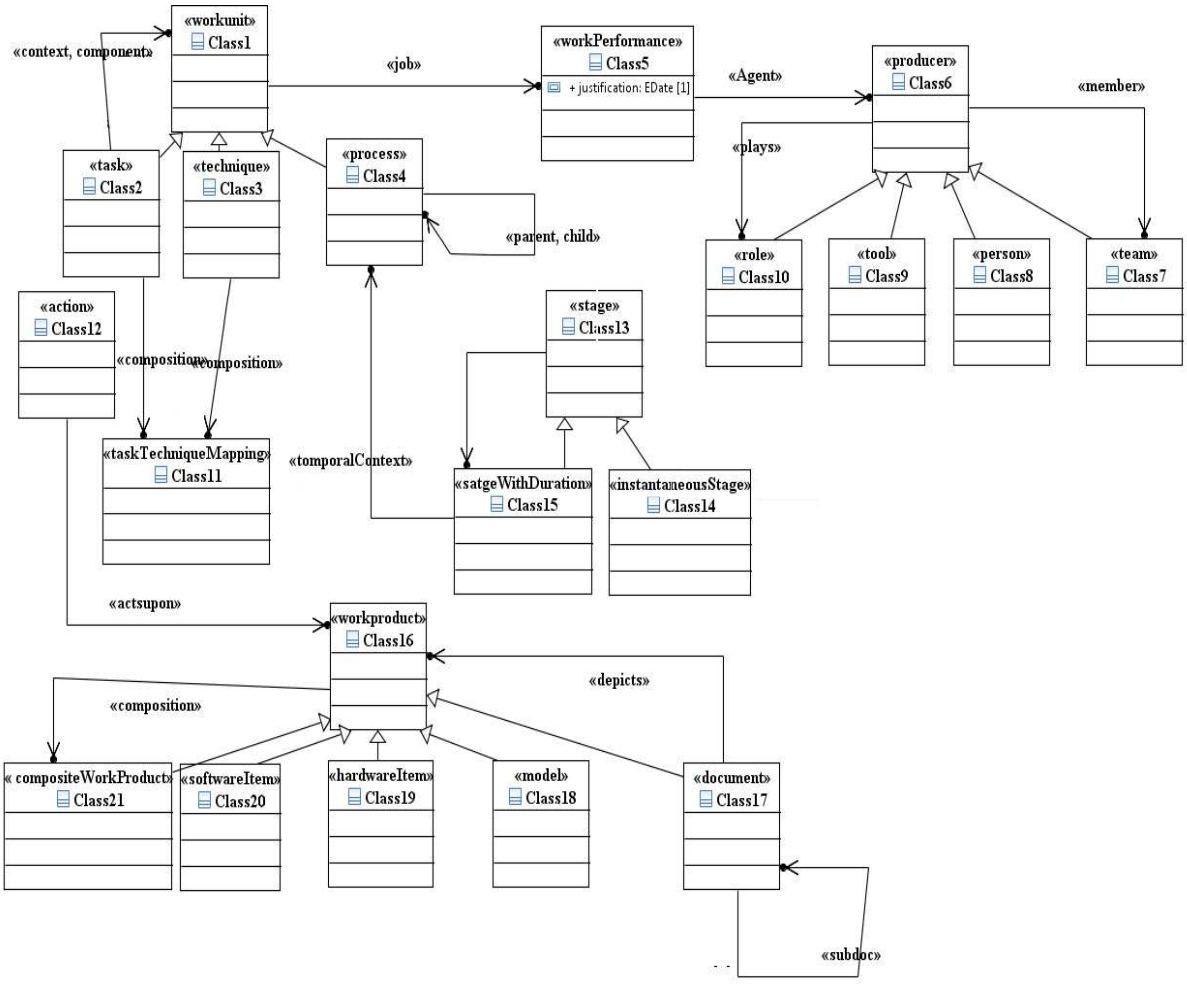

Figure 6.5 PROFIL UPISO ENDEAVOUR

6.3.2 Transformations des Modéles

Pour réaliser les transformations de modèles, et pour rester dans le contexte de standardisation défini par MDA, nous avons utilisé ATL, un langage qui répond à la spécification MOF 2.0. QVT (Query/Views/Transformations) définie par l'OMG en tant que méta-modèle de transformation de modèles (OMG 2011b).

Les méta-modèles UPISO, OUPISO, ODM et OWL sont d'abord traduits vers des représentations XMI, puis nous réalisons les transformations via ATL. Ces transformations

concernent les mappings de l'UPISO vers l'OUPISO, puis de l'OUPISO vers ODM et enfin d'ODM vers OWL. Enfin, l'outil ontologique Protégé est utilisé pour la validation des descriptions OWL générées.

6.3.2.1 Transformation UPISO vers OUPISO

La traduction de l'UPISO vers l'OUPISO est effectuée à l'aide des règles de transformation définies en concordance avec la Table 6.2. Les éléments de l'UPISO désignent des concepts ontologiques dans OUPISO. Par conséquent, OUPISO étend UML avec de nouveaux constructeurs pour supporter nos concepts ontologiques spécifiques tels que les classes stéréotypes : <<Ontology>> et <<OntClass>>. Nous convertissons les classes stéréotypées vers des classes ontologiques avec le stéréotype <<OntClass>>. Les valeurs marquées (ou tagged values) qui correspondent aux propriétés ou aux attributs sont transformées en données ontologiques avec le stéréotype << DatatypeProperty>>. Enfin, le stéréotype <<Package>> devient <<Ontology >> et les associations stéréotypées sont translatées vers le concept <<ObjectProperty>>.

Tableau 6.2 Mappings entre les concepts UPISO et OUPISO.

Concepts UPISO (Stéréotypes ou tags)	Concepts OUPISO (Concepts Ontologiques)
« Package »	« Ontology »
« Class »	« OntClass »
«Association »	« ObjectProperty »
« Attribute »	« DatatypeProperty »

6.3.2.1.1 Règles de transformation UPISO vers OUPISO

Les règles suivantes permettront de faire le passage du profil UML UPISO vers le profil UML ontologique OUPISO.

a) Règle 1 : Classe vers classe

```
rule Class2Class {from
                    a : UPISO!Class
                    to
                    p:OUPISO!Class(name<-a.getDefNameSet(),generalization<-a.generalization),
                    on:OUPISO!OntClass(base_Class<-p)
                }
```

b) Règle 2 : Généralisation vers généralisation

```
rule Generalization2Generalization {from
                                    a : UPISO!Generalizatio
                                    to
                                    p:OUPISO!Generalization(general<-a.general)
                                    }
```

c) Règle 3 : Propriété de données vers Propriété de données

```
rule property2datatypeproperty {from
                        a : UPISO!property
                        to
                        a1 : OUPISO!Property(name<-a.base_Property.name,
                        type<-a.base_Property.type),
                        p2:OUPISO!Association( name<-a.base_Property.name--.asSequence().first().name,
                                               ownedEnd<-prop ,ownedEnd<-a1),
                                               prop:OUPISO!Property(name<-'src',
                                               type<-a.base_Property.class),
                                               d:OUPISO!DataTypeProperty(base_Association<-p2)
                            }
```

d) Règle 4 : Propriété de données vers classe

```
rule DataType2Class {from
                     a : UPISO!DataType
                     to
                     p:OUPISO!Class(name<-a.name), on:OUPISO!OntClass(base_Class<-p)
                     }
```

6.3.2.2 Transformation OUPISO vers ODM et OWL

La deuxième transformation consiste à produire un document OWL, conforme à la syntaxe OWL/XML, à partir du méta-modèle OWL généré par la première transformation en utilisant un extracteur XML. Le fichier OWL obtenu pourra être utilisé et manipulé à travers des outils d'ontologies tels que Protégé2000. Cette transformation se fait à l'aide des règles selon les correspondances décrites dans le Tableau 6.3.

Tableau 6.3 Correspondances entre quelques éléments UML et OWL

Concepts OUPISO **Concepts ontologiques comme stéréotypes ou tags**	**Concepts ODM** **Concepts ontologiques**	**Concepts OWL**
«Ontology »	Class Ontology	OWL : Ontology
«OntClass»	Class Class	OWL : Class
«ObjectProperty»	Class ObjectProperty	OWL : objectProperty
«DatatypeProperty»	Class DatatypeProperty	OWL : datatypeProperty

6.3.2.2.1 Règles de transformation OUPIOS vers ODM

Les règles suivantes permettront de faire le passage de l'ontologie UML profil UPISO vers le modèle ODM.

a) Règle 1 : Classe vers classe

```
rule Class2Class { from
                    a : OUPISO!Class
                    to
                    p:ODM!Class(name<-'Class',ownedAttribute<-pr,
                                generalization<-a.generalization),
                    pr:ODM!Property(name<-a.name)
                   }
```

b) Règle 2 : Généralisation vers généralisation

```
rule Generalization2Generalization {from
                                    a : OUPISO!Generalization
                                    to
                                    p:ODM!Generalization(general<-a.general)
                                    }
```

c) Règle 3 : Association vers classe

```
rule Association2Class {from
                        a : OUPISO!Association
                        to
                        p:ODM!Class(name<-a.getname(),ownedAttribute<-pr,
                        ownedAttribute<-e,
                        ownedAttribute<-k,
                        generalization<-a.generalization),
                        pr:ODM!Property(name<-a.name),
                        e:ODM!Property
                        (name<-'Domain',type<- a.ownedEnd->asSequence()->first().type),
                        k:ODM!Property(name<-'Range',type<- a.ownedEnd->asSequence()-
>last().type),
                        e1:ODM!Property(name<-'',type<- a.ownedEnd->asSequence()-
>first().type),
                        k1:ODM!Property(name<-'',type<- a.ownedEnd->asSequence()-
>last().type),
                        S:ODM!Property(name<-'',type<-p),
                        S1:ODM!Property(name<-'',type<-p),
```

6.3.2.2.2 Règles de transformation ODM vers OWL

a) Règle 1 : Classe ODM vers classe OWL

Les règles suivantes permettront de faire le passage d'ODM vers le modèle OWL :

```
rule ODMClass2OWLClass {from
                        a :ODM!Class
                        to
                        p: OWL!OWLClass
                        (subClassOf <- a.general,uriRef <-u,label <- label),
                        label : OWL!PlainLiteral ( lexicalForm <- a.name ),
                        u : OWL!URIReference ( fragmentIdentifier <- l,uri   <- uri ),
                        l : OWL!LocalName ( name <- a.name ),
                        uri : OWL!UniformResourceIdentifier ( name <- a.name )
                       }
```

b) Règle 2 : Propriété d'Objet ODM vers propriété d'objet OWL

```
rule ODMObjectProperty2OWLObjectProperty {from

                        a :ODM! ObjectProperty
                        to
                        p: OWL! ObjectProperty
                                        (uriRef <-u,domain <- a.class,range <- a.type),
                        u : OWL!URIReference
                                        ( fragmentIdentifier <- l, uri <- uri ),
                        l : OWL!LocalName
                                        ( name <- a.class.name + '.' + a.name ),
                        uri : OWL!UniformResourceIdentifier
                                        ( name <- a.class.name + '.' + a.name )
                                }
```

c) Règle 3 : Propriété de donnée ODM vers propriété de donnée OWL

```
rule ODMDataProperty2OWLDataTypeProperty {from

            a :ODM!DataProperty
            to
            p: OWL!OWLObjectProperty (
            uriRef <- u,
            domain <- p.class,
            range <- p.type),
            u : OWL!URIReference ( fragmentIdentifier <- l, uri <- uri ),
            l : OWL!LocalName ( name <- a.class.name + '.' + a.name ),
            uri : OWL!UniformResourceIdentifier ( name <- a.class.name + '.'
                  + a.name )
                                        }
```

6.4 Fusion et la création de la relation KIND

La fusion est une agrégation de deux opérations ontologiques : l'intégration et l'assemblage. L'objectif de la fusion d'ontologie est donc l'obtention d'une nouvelle ontologie à partir de deux ou plusieurs ontologies sources liées au même sujet. Les ontologies sources sont donc remplacées et unifiées dans l'ontologie résultat de la fusion. Pour parvenir à ce résultat, les approches les plus utilisées sont l'union ou l'intersection. Dans notre cas, nous allons fusionner les trois descriptions OWL obtenues à savoir Endeavour.owl, Template.owl et Resource.owl.

À la fin de l'opération de fusion nous obtenons un seul fichier OWL qui regroupe tous les concepts et relations de la norme, appelés ISO.owl.

Une fois les trois ontologies regroupées dans un seul fichier nous allons parcourir le fichier pour créer pour chaque concept « élément KIND » une relation « kind of » avec le concept « élément ».

La figue 6.6 montre l'outil développé pour la fusion et la création de la relation KIND.

Figure 6.6 Outil de fusion d'ontologie

6.5 Validation de l'ontologie de domaine ISO /IEC24744

Après avoir terminé l'étape précédente nous avons utilisé l'outil OntOlogy Pitfall Scanner ! (OOPS !) (Poveda-Villalón et al. 2012) qui est un outil indépendant de tout éditeur d'ontologies. Le but de « OOPS » est l'identification des anomalies ou mauvaises pratiques dans une ontologie. Pour cela les auteurs ont défini un certain nombre de « pitfalls » (embûches, pièges) répertoriés en langage naturel dans un catalogue. On en compte actuellement 40, dont 32 sont implémentés en tant que classes java et ajoutés au module d'analyse des pitfalls. En entrée, l'application prend l'URI d'une ontologie ou bien le code source en RDF 12. L'ontologie est chargée via l'API Jena avant d'être analysée pour en extraire les erreurs potentielles. Le résultat est une page Web sur laquelle sont répertoriées les pitfalls (les erreurs identifiées) accompagnées d'une proposition de résolution. Les pitfalls peuvent concerner des éléments individuels, plusieurs éléments ou toute l'ontologie.

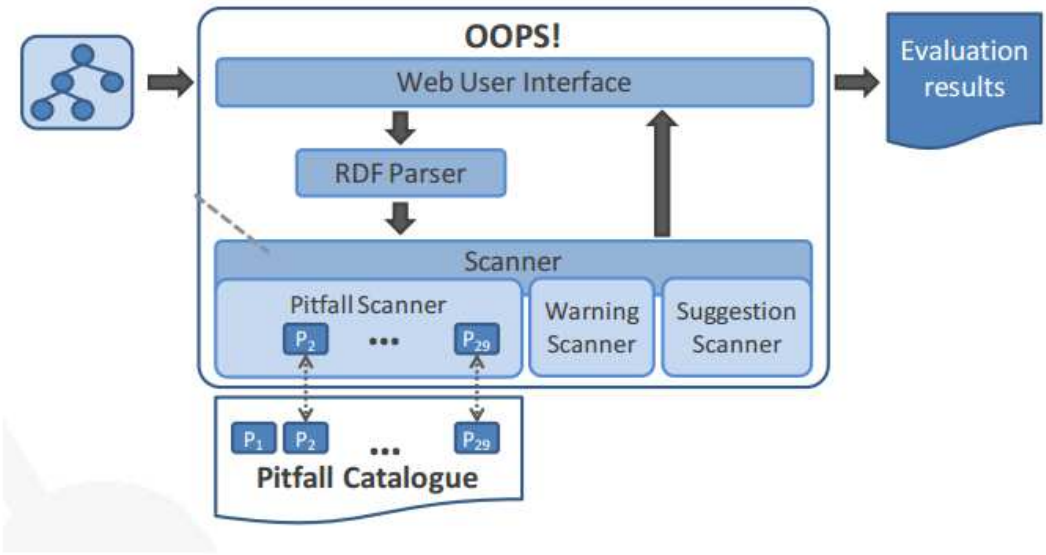

Figure 6.7 Architecture de l'outil OntOlogy Pitfall Scanner !

6.6 Evaluation de l'ontologie de domaine ISO /IEC24744

L'évaluation des ontologies peut s'opérer sur plusieurs niveaux. La plupart des chercheurs consacrent une attention particulière à l'évaluation de l'ontologie dans son ensemble, et proposent des indicateurs pour examiner la qualité de son contenu.

Divers métriques ont été proposées dans la littérature afin d'évaluer les ontologies. Elles sont considérées comme des moyens qualitatifs utiles pour évaluer les différents aspects d'une ontologie. Dans ce qui suit, nous présentons les principales métriques utilisées pour mesurer la qualité d'une ontologie. Il est très important de signaler que certaines métriques proposées

ci-dessous sont mesurables, mais d'autres sont considérées comme des objectifs à atteindre lors de l'évaluation (c'est-à-dire, elles ne sont pas mesurables).

Dans notre évaluation , nous avons opté pour l'utilisation des métriques proposées par (Burton-Jones, Storey et al. 2005) afin évaluer la qualité de notre ontologies à savoir:

- L'éligibilité : la fréquence des erreurs syntaxiques,
- La richesse : le nombre de relations non-hiérarchiques par rapport au nombre de relations hiérarchiques,
- L'interprétabilité : la présence des termes utilisés dans WordNet,
- La consistance : le nombre de concepts impliqués dans des contradictions,
- La clarté : les termes utilisés dans l'ontologie ont-ils plusieurs sens dans WordNet?
- La compréhensibilité : le pourcentage des concepts et relations annotés,
- L'exactitude : le pourcentage de fausses relations,
- L'autorité : combien d'autres ontologies utilisent les concepts de l'ontologie à Evaluer ?
- l'historique : le nombre d'accès à l'ontologie

Dans notre travail nous avons calculé les différentes métriques précédemment citées pour notre ontologie ainsi l'ontologie Generic Process Ontology GPO (Becker, Papa et al. 2015) que nous avons dressé dans le tableau 6.4 pour une meilleure lisibilité

Tableau 6.4 Résultat de l'évaluation

	ISO/IEC 24744 ONTOLOGY	Generic Process Ontology (GPO)
Eligibilité	00	00
Richesse	2 ,6	03
Interopérabilité	63%	59%
Consistance	00	00
Clarté	43%	39%
Compréhensibilité	13%	Non estimée
Exactitude	00%	Non estimée
Autorité	64,3%	42%
Historique	Non estimée	Non estimée

6.6.1 Interprétation

Après avoir évalué notre ontologie selon les critères établies par Burton(Burton-Jones, Storey et al. 2005) et comparé les résultats obtenus par rapport a l'ontologie GPO nous pouvons dire que les résultats sont très satisfaisantes. Pour mieux expliquer nos résultats nous allons les discuter paramètre par paramètre :

- L'éligibilité : aucune erreur syntaxique n'est à relever dans les deux ontologies, et cela est dû aux outils de validation.
- La richesse : ce point là ne figure pas dans l'histogramme car c'est un taux et pas un pourcentage. Si ce taux est inférieur à un, il indique que l'ontologie est plus proche d'une taxonomie qu'une ontologie proprement dite. Dans notre cas nous avons obtenu 2,6 qui est un taux très respectable par rapport à l'ontologie GPO qui a un taux de 3
- L'interopérabilité : 63% des termes utilisés sont dans WordNet, il faut soulingner que certains termes qui ne font pas parti de WordNet sont une composition de deux ou trois termes, qui eux par contre existent dans WordNet (exemple : SoftwareItem, StageWithDuration) malgré cela le résultat obtenu est légèrement meilleur que celui de l'ontologie GPO qui affiche 59%.
- La consistance : pour notre ontologie aucun concept n'est impliqué dans des contradictions. Pour l'ontologie GPO nous n'avons pas pu estimer ce paramètre.
- La clarté : 43 % des termes utilisés dans notre ontologie ont plusieurs sens dans WordNet, par rapport à 39% dans l'ontologie GPO.
- La compréhensibilité : nous avons 13% des concepts et relations annotés, et aucune indication pour l'ontologie GPO.
- L'exactitude : 00% de fausses relations, pour les deux ontologies.
- L'autorité : un taux de 64% des concepts et relations existe dans les deux ontologies est ce taux peut être réellement mieux car plusieurs concepts dans notre ontologie sont défini par d'autre terme dans d'autre ontologie.
- l'historique : pour le nombre d'accès à l'ontologie aucun pourcentage ne peut être donné pour le moment.

6.7 Implémentation

6.7.1 Introduction

Dans le chapitre précédant, nous avons exposé notre approche concernant la génération d'ontologie OWL de la norme ISO/IEC 24744 à partir de son métamodèle en utilisant des technique MDA. Nous y avions également présenté ODM qui est un standard développé par l'OMG qui permet de transformer un modèle UML en modèle OWL et inversement. Notre objectif durant cette expérimentation est de vérifier la faisabilité du processus sur lequel repose notre approche. Ici nous allons voir comment nous avons procédé pour mettre en œuvre notre approche. Dans un premier temps nous allons présenter comment créer un profil UML à partir de son métamodéle , par la suite l'implémentation de ODM que nous utilisons .

En suite nous présenterons par la suite la méthodologie que nous avons adopté, ainsi que l'API que nous avons mis en place pour le chargement des modèles et l'exécution des différentes transformations.

6.7.2 Outillage

Dans notre implémentation et nos transformations, nous nous sommes basé sur le Framework Eclipse (eclipse) et le plugin Papyrus (Papyrus). Notre choix a été motivé par le fait que l'EMF est actuellement l'environnement de référence pour l'Ingénierie Dirigée par les Modèles, car il propose l'implémentation des couches Méta pour la plupart des outils issus de la recherche dans le domaine de l'IDM. Pour les transformations de modèles, notre choix s'est porté sur l'un des projets les plus adoptés par la communauté Eclipse, le moteur de transformation (ATL). Ce moteur fournit une implémentation proche du standard Query View Transform (QVT) d'OMG, qui supporte à la fois l'énonciation déclarative et impérative des règles de transformation. Dans notre cas, nous avons développé l'intégralité de nos profils UML avec l'outil UML, reposant lui-même sur la plateforme Eclipse. On remarquera que nos outils pour la manipulation des modèles ont été eux aussi réalisés grâce à des technologies IDM, et une partie de leur code a été généré grâce à des modèles.

Afin de pouvoir être utilisés en transformation de modèles et en entrée de la phase de sérialisation, nos profils ont étés implémentés comme un méta- modèle de EMF au format Ecore. Son édition a été intégralement réalisée en UML, avec l'outil Papyrus. Cependant, comme Papyrus est un éditeur UML, les méta-classes qu'il instancie et sérialise dans ces modèle sont celles du méta-modèle d'UML et non pas du méta-modèle du MOF. Nous avons dû convertir des modèles UML en modèles Ecore. Ce format étant l'implémentation du MOF que fournit l'EMF.

Pour illustrer notre processus MDA de génération décrit précédemment, nous allons présenter uniquement quelques captures d'écran liées à l'élaboration des Profils UML pour OWL, à leurs transformations, puis à la validation de l'ontologie de domaine.

6.7.3 Creation du profil UML UPISO

L'outil Papyrus(Papyrus) qui est un outil open source et basé sur la technologie Eclipse est l'un des outils les plus utilisés pour la modélisation des profils UML. Il est dédié à la modélisation UML2 et soutient également les langages SysML et MARTE. Le point qui m'a intéressé le plus avec cet outil, c'est qu'il contient un module spécifique à la gestion des profils UML (voir figure 7.1) et il est accompagné d'une bonne documentation spécifique à

l'usage des profils UML. Il fournit en plus un assistant qui facilite la génération du code OCL pour la génération des contraintes et de les appliquer aux profils.

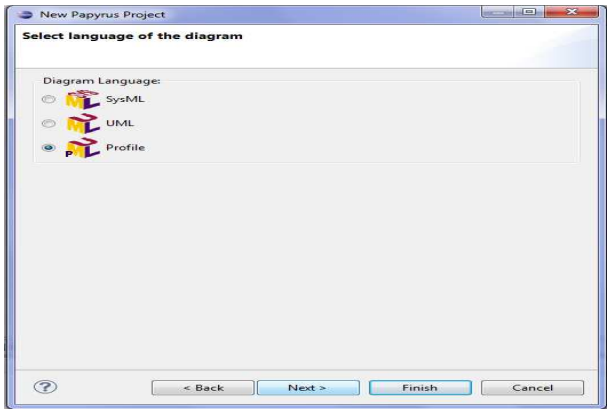

Figure 6.8 Modules de gestion des profils avec Papyrus

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

La première étape consiste à mettre en place les stéréotypes définis précédemment dans le chapitre. Nous pouvons voir dans la figure 7.2 les différents stéréotypes composant le profil avec l'étiquette « stereotype », les propriétés des classes du méta-modèle sont définies de même dans le profil mais en tant que tagged values.

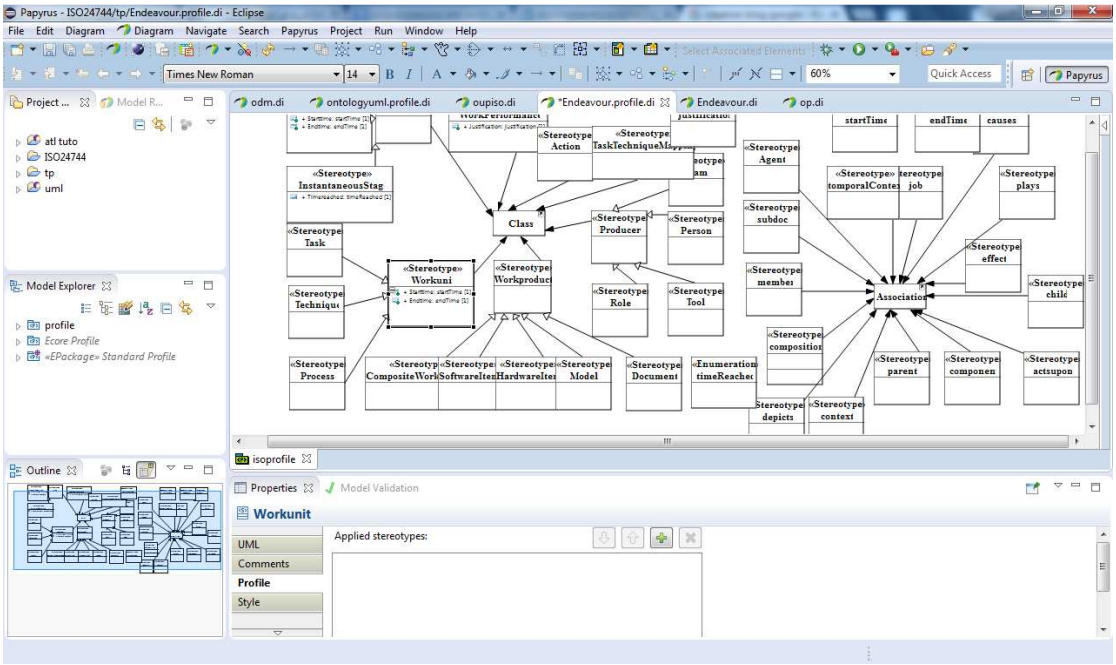

Figure 6.9 Capture d'écran du profil UML UPISO

6.7.4 Transformations UPISO vers OUPISO

Ces transformations concernent les mappings de concepts d'UPISO vers ceux d'OUPISO. OUPSWS est un Profil UML d'ontologie pour la norme ISO/IEC 24744 et ses concepts sont ontologiques. Avant d'effectuer ces transformations, nous devons d'abord construire un package contenant un ensemble de stéréotypes pouvant supporter des concepts ontologiques tels que "Ontology", "Ontoclass", "ObjectProperty" et "DatatypeProperty". Les stéréotypes que nous créons devraient par la suite supporter le méta-modèle de définition d'ontologie (ODM) dans le but de faciliter le mapping entre OUPISO et ODM. Une fois crée, l'utilisateur peut effectuer ses transformations. La Figure 6.11 montre une capture d'écran du package des stéréotypes OUPISO.

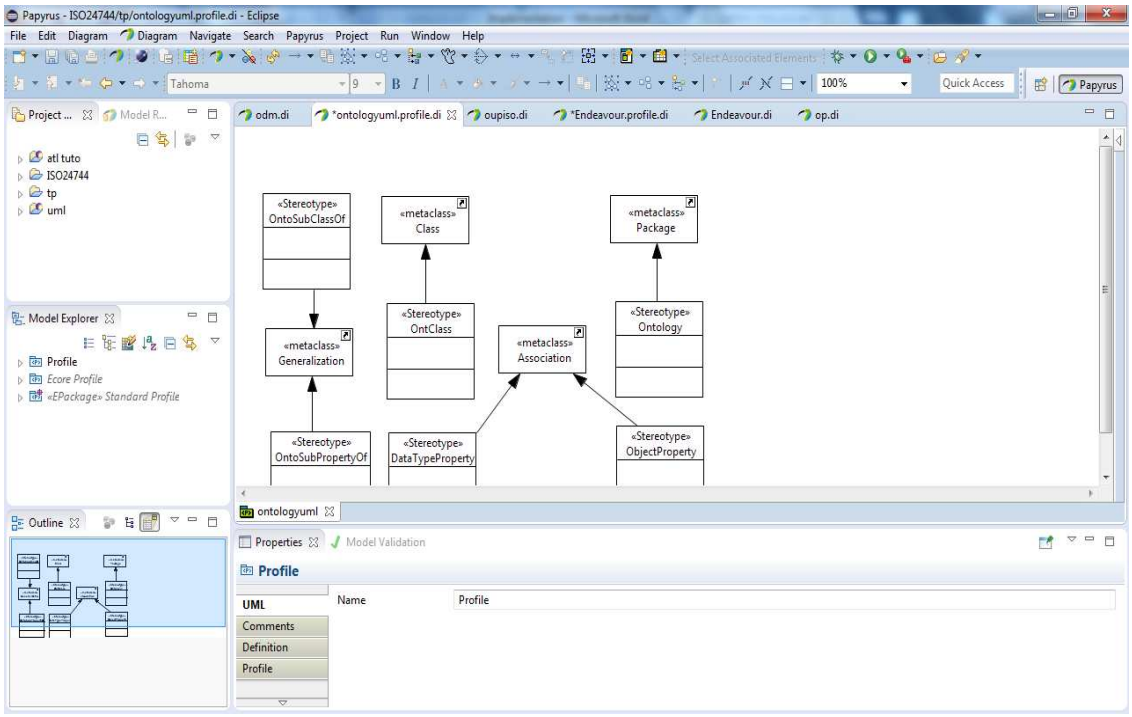

Figure 6.10 ***Capture d'écran du Package des stéréotypes OUPISO.***

Une fois les stéréotypes OUPSWS créés, nous traduisons le Profil UML (UPISO), construit précédemment, vers leurs correspondants OUPISO. Pour ce faire, nous exécutons les règles de transformations définies dans le tableau 6.2. La figure6.12 montre une capture d'écran après l'exécution complète des règles de transformation qui a permis la génération du profil ontologique d'UML OUPISO.

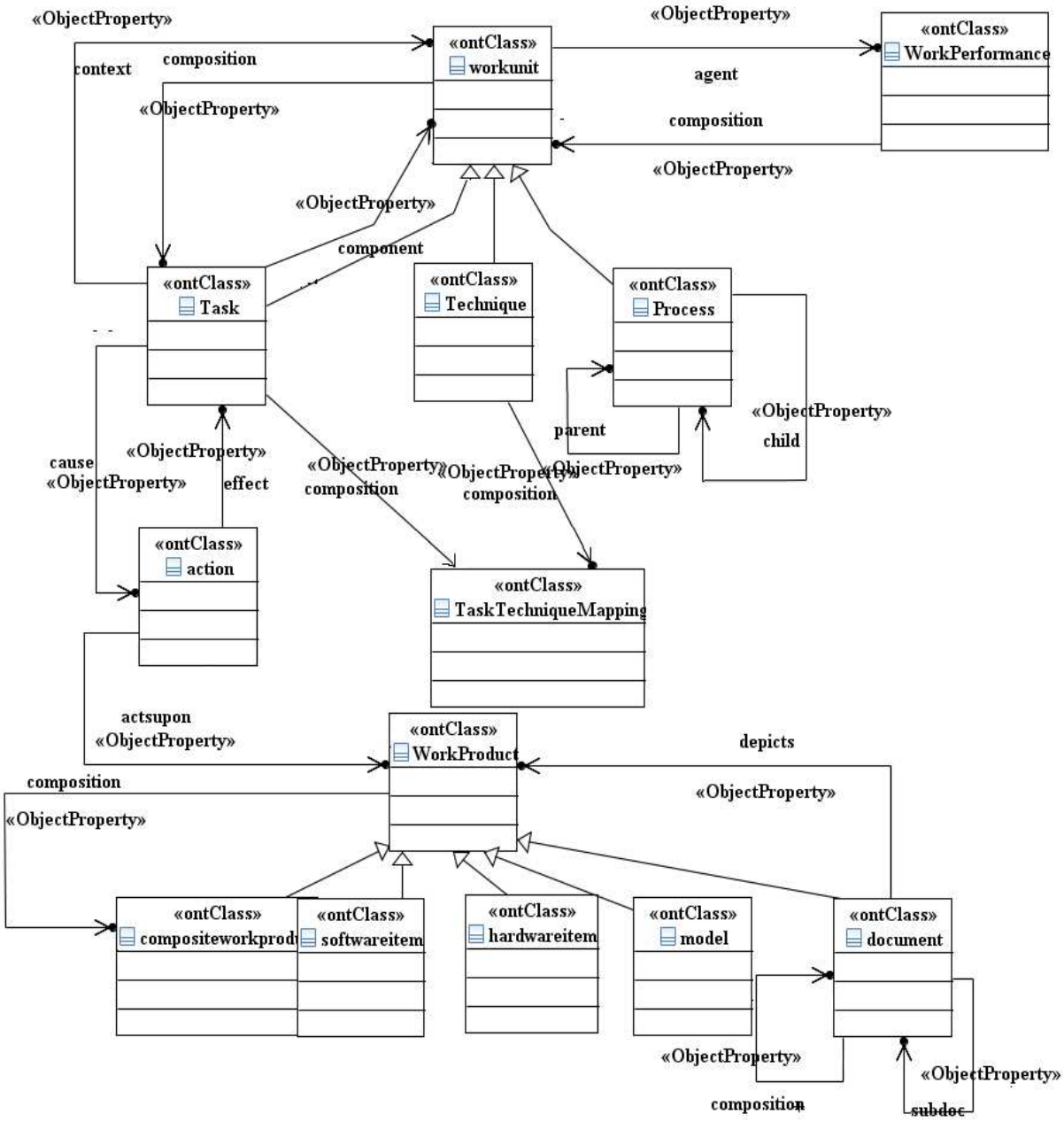

Figure 6.11 La partie OUPISO générée pour Endeavour

6.7.5 Transformations OUPISO vers ODM

Les transformations du Profils UML d'ontologies (OUPSWS) vers ODM sont simples du moment que les concepts ODM sont utilisés en tant que stéréotypes dans OUPSWS. Nous obtenons après exécution des règles correspondantes à cette partie, la partie ODM générée Endeavour (Figure 6.13).

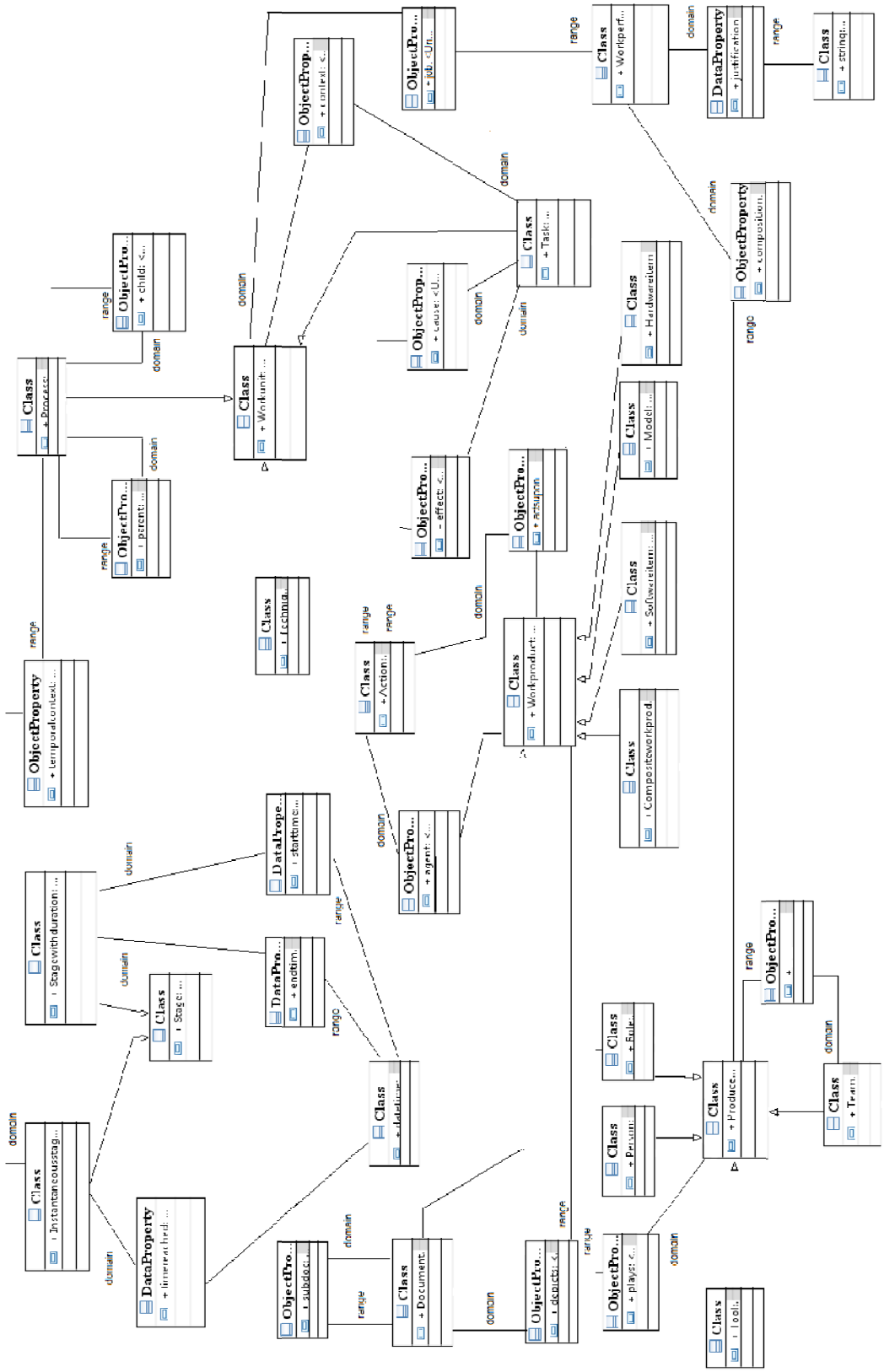

Figure 6.13 La partie ODM générée pour Endeavour

6.7.6 Transformations ODM vers OWL

Une fois les trois parties ODM générées, nous exécutons les règles de transformations correspondantes à cette phase pour générer les parties d'OWL correspondante. Ces règles de

transformations sont déjà définies dans le tableau 6.2. Une fois, les règles ATL sont exécutées, nous obtenons une description OWL que nous présentons dans la figure 6.14.

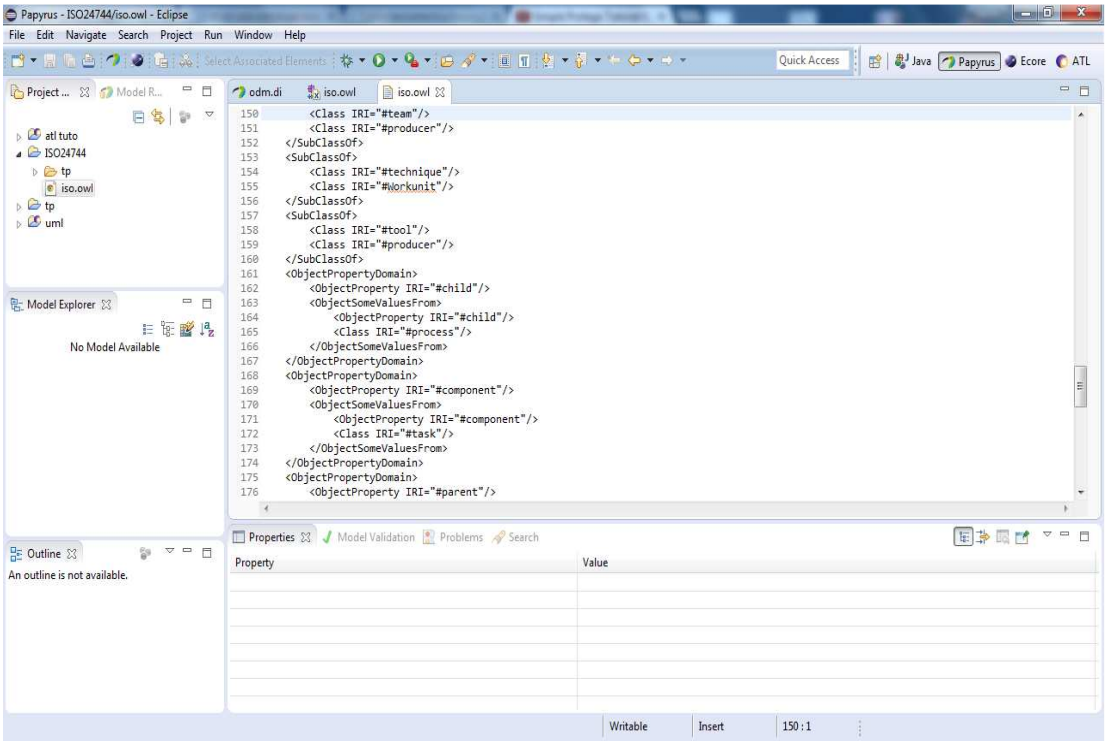

Figure 6.12 la partie OWL générée pour Endeavour

6.7.7 Fusion des descriptions OWL générés

Dans cette dernière étape, nous fusionnons les trois descriptions OWL générées dans les étapes précédentes en un seul fichier OWL. Nous utilisons l'API JENA pour générer les liens entre les éléments Endavour et Templete avec la relation Kind. Le fichier OWL obtenu sera exporté vers l'outil OOPS pour validation. À partir de cette base, nous avons développé un petit outil java pour pouvoir utiliser APIJENA. Cet outil doit avoir en entrée les trois fichiers OWL à fusionner et aura comme sortie un seul fichier fusionné.

Figure 6.13 Outil de fusion des ontologies

6.7.8 VALIDATION

Pour la validation de la structure d'une ontologie, nous avons donc opté pour l'outil OOPS ! pour plusieurs raisons : disponibilité de l'outil et mise à jour régulière, critères utilisés qui ont été définis suite à une étude du domaine, possibilité de conserver son code source privé, gratuité, indépendance du module, utilisation sous n'importe quel navigateur Web.

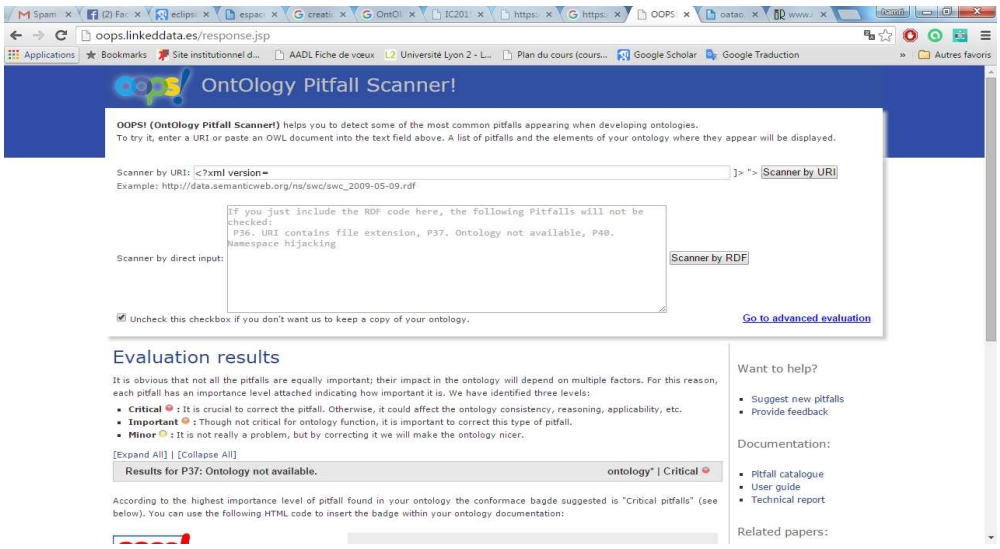

Figure 6.14 **Capture d'écran de l'outil OntOlogy Pitpall Scanner ! (OOPS !)**

6.8 Conclusion

A l'issu de ce chapitre, nous avons construit l'architecture de notre approche. Cette architecture nous permet ainsi, de générer une ontologie pour la normeISO/IEC24744 dans n'importe quel langage de représentation d'ontologies basé Web tels que OWL ou RDF-S. Parmi les avantages que présente cette méthodologie basée MDA. La standardisation d'ODM nous permet, depuis un métamodèle UML, d'avoir son équivalent en modèle d'ontologie OWL. L'architecture ainsi bâtie, nous fournit une passerelle entre le génie logiciel et l'ingénierie ontologique par l'utilisation du méta-modèle de définition d'ontologie ODM pour décrire la sémantique de manière conceptuelle, indépendamment de tout langage ontologique. Nous avons présenté également comment définir un Profil UML à partir d'un metamodele, aussi, nous avons exposé dans ce chapitre l'implémentation des étapes suivies pour concevoir le profil UML pour supporter la norme ISO/IEC 24744. Nous avons tout d'abord défini le modèle de domaine et les règles liées au méta-modèle. Nous avons par la suite présenté le mapping des concepts du profil vers ODM pour arriver à la fin à un fichier OWL.

7 Chapitre VII Conclusion

Ces dernières années, les ontologies ont suscité beaucoup d'intérêt et ont fait l'objet d'une attention toute particulière de la part de la communauté du développement logiciel. Leur pouvoir à exprimer les solutions à un niveau d'abstraction élevé et leur capacité à améliorer la productivité et la qualité des produits logiciels sont les principales motivations derrière cet intérêt. Néanmoins, le développement de ces ontologies est encore considéré comme une activité difficile et coûteuse, ce qui constitue souvent un frein à leur adoption dans un cadre industriel.

Le travail de recherche décrit dans cette thèse est une contribution à l'axe des procédés logiciels et l'ingénierie ontologique. Le but principal était d'élaborer une méthode pour la définition d'une ontologie qui décrit, entre autres, les phases et les activités à entreprendre, les artefacts à manipuler et les techniques à utiliser lors du développement d'un projet logiciel basé sur le metamodèle de la norme ISO/IEC 24744 (SEMDM) en utilisant une démarche MDA.

Ainsi, pour Remplir et atteindre nos objectifs nous avons étudié un certain nombre de domaines à savoir :

– L'ingénierie ontologique : les concepts, les mécanismes d'inférences, les langages tels que OWL, SPARQL, SQWRL, SWRL...etc, ainsi que les outils existants tels que "Protégé" ont été étudié pour la mise en place de notre ontologie de domaine. Nous avons dédié un chapitre à la définition des concepts de base des ontologies de domaine.

– Les procédés logiciels : en plus de envelopper et appréhender les concepts des PLs de façon générale, nous nous sommes focalisés sur le metamodèle de la norme ISO/IEC 24744 de façon particulière. L'étude du domaine des PLs est présentée dans le chapitre -2-.

- l'ingénierie dirigée par les modèles (IDM) : nous présentons les concepts clés de l'IDM et les principales approches rattachées, puis nous focalisons notre discussion sur l'approche MDA et les travaux de normalisation de l'OMG. Nous détaillons les deux premiers axes principaux de l'IDM selon l'approche MDA et sur lesquels portent nos travaux de recherche. Nous présentons la transformation de modèles qui est le troisième axe clé de MDA.

Notre état de l'art a couvert aussi les travaux connexes relatifs aux différentes ontologies pour les procédés logiciels et qui nous a permis de constater qu'il n'existait pas une ontologie proprement dite pour la norme ISO/IEC 24744, bien que cette norme est considérée comme la plus importante des travaux ISO dans le domaine du procédé logiciel(Henderson-Sellers, Gonzalez-Perez et al. 2014).

En résumé, cette phase d'exploration nous a permit de comprendre les enjeux et les concepts au tour des domaines en relation avec notre thèse et d'approfondir la problématique de développement des ontologies pour les procédés logiciels. Dans cette même phase, nous avons réalisé une synthèse de la littérature traitant le développement d'ontologie pour les procédés logiciels afin d'identifier les pratiques et les techniques généralement impliquées dans un processus de développement. Ce sont ces pratiques et ces techniques qui nous ont servi d'intrants pour la phase d'élaboration de notre ontologie.

7.1 Contributions

L'étude de ces différents domaines en plus de l'analyse des solutions existantes nous a permit de présenter deux contributions majeures :

- La construction d'une architecture ontologique basée MDA pour la prise en compte de l'aspect sémantique au niveau metamodéle. Cette construction est basée sur l'intégration des trois espaces technologiques: MDA, les ontologies et les procédés logiciels qui conjointement semblent être une voie prometteuse pour favoriser l'interopérabilité sémantique. Cette contribution a abouti sur la création d'une ontologie pour la norme ISO/IEC 24744. Cette ontologie peut être utilisée pour lever l'ambigüité terminologique sur les différents concepts de la norme et offre une représentation formelle de son vocabulaire.
- Parallèlement à la construction de notre ontologie nous avons défini un profil UML pour la norme ISO/IEC 24744. Afin de pouvoir utiliser le metamodèle de la norme ISO/IEC 24744 dans notre architecture ontologique basée MDA nous nous sommes trouvés devant l'obligation de la conception d'un profil UML pour la norme ISO/IEC 24744. L'intérêt primordial de ce profil UML 2.0 est d'exprimer les concepts architecturaux en UML 2.0 et donc de définir de manière formelle. En d'autre terme, l'utilisation de stéréotypes, de règles et de valeurs marqués permet précisément de mieux capturer les concepts d'architecture de la norme pour pouvoir mieux les exploiter dans une stratégie de transformation directe.

7.2 Positionnement de la thèse par rapport à l'état de l'art

Les divers travaux de recherche menés jusqu'à l'heur pour la création d'ontologie pour la norme ISO/IEC 24744 montre que le sujet est toujours d'actualité et mobilise un grand nombre de chercheurs et d'organismes internationaux comme ISO mais aucun travail n'est arrivé réellement à la création d'une ontologie décrite d'une façon formelle et automatique. Les travaux de Henderson et al (Henderson-Sellers, McBride et al. 2013, Henderson-Sellers, Gonzalez-Perez et al. 2014) donnent une description a l'aide de diagrammes UML. Les travaux de Ruy et Falbo(Ruy, Falbo et al. 2014, Ruy, Falbo et al. 2015) proposent une analyse ontologique pour la norme ISO/IEC 24744 mais ils ne décrivent en aucun cas une approche ou une proposition d'approche pour la création de l'ontologie. À partir de ces travaux nous pouvons dire que ce sujet n'a encore atteint le degré de maturité souhaité et qu'il reste encore un grand travail à faire. Notre travail a l'encontre des travaux précédemment cités donne une approche automatique basée sur des standards et normes tel que MDA et ODM pour la création d'une ontologie de domaine ISO/IEC 24744 dans un langage formelle à savoir OWL. Notre contribution peut aider à l'amélioration des différents travaux qui sont encore en cours.

Nous avons comparé notre travail avec le travail de (Líška and Návrat 2010). Ce dernier vise à créer une ontologie pour la méthode SPEM 2.0. Notre intention n'était pas de concurrencer ces travaux pour la simple raison qu'il traites de méthodes différentes mais de réutiliser leurs contributions pour améliorer notre approche. Notre travail est assez semblable du point de vue architectural au travail de Liska vue que les deux approches ont utilisé l'architecture de Djuric pour la création d'ontologies. Les deux approches proposent la transformation d'une norme de MDA vers le Web sémantique avec un mapping entre le profil d'UML Ontology et un profil arbitraire d'UML. L'existence de similitude entre les deux modèles « SPEM et ISO/IEC 24744 » nous a motivé à faire ce choix.

Notre approche a permit de créer un profil UML pour la norme tandis que le travail de (Líška and Návrat 2010) avait l'avantage d'avoir le SPEM UML profil avec la spécification formel OMG. Bien que le travail de Liska ait cet avantage vu que SPEM est natif d'MDA avec une spécification en XMI, lui permettant d'utiliser directement des outils de modélisation tel qu'Eclipse, ATL et Ecore, l'auteur a préféré écrire des règles de mapping en Adhoc. Cependant, dans notre approche, nous utilisons les règles de passage avec le langage ATL pour générer les mapping de façon automatisée. De plus, nous avons préféré la modularité en utilisant les trois profils Endeavour, Template et Resource pour faciliter la maintenance et l'évaluation chose qui n'a pas été faite dans les autres travaux. Le travail de Liska a été une

référence pour nous et une grande source d'inspiration bien que les objectifs ne soient pas spécifiquement les mêmes, mais le but global était de créer une ontologie de domaine pour le génie logiciel.

7.3 Perspectives

Ce travail de recherche ouvre la voie vers différentes perspectives et peut être poursuivi dans Plusieurs directions. Parmi ces directions, nous suggérons celle du perfectionnement de la méthode. Ensuite, il y a la direction de l'expérimentation et de l'évaluation de l'ontologie et enfin la direction de l'opérationnalisation de l'ontologie.

7.3.1 Perfectionnement de la méthode

La méthode proposée dans cette thèse s'est concentrée surtout sur la mise en place d'un processus qui permet a partir d'un metamodèle de la norme ISO/IEC 24744 d'arriver a une ontologie. Cette ontologie a pour objectif d'être une référence sémantique qui définit les concepts du domaine et les différentes relations qui existent entre eux. Donc la première perspective a notre avis est de compléter et d'affiner la description des éléments du metamodéle et de faire une révision approfondie du profil UML défini.

7.3.2 L'expérimentation et de l'évaluation de l'ontologie

Des expérimentations supplémentaires sont nécessaires pour bien évaluer la pertinence de notre ontologie. Dans cette direction nous suggérons de mener avec des partenaires industriels des études expérimentales mettant en œuvre cette ontologie. Ces expérimentations doivent être menées de façon à garantir l'objectivité et l'impartialité des résultats.

7.3.3 L'opérationnalisation de l'ontologie

C'est la principale perspective de notre travail. L'opérationnalisation de l'ontologie a pour objectif de faire passer notre ontologie d'une simple définition de concepts et relations à un outil capable de valider ou vérifier un projet logiciel. Nous visons à atteindre cet objectif en ajoutant des mécanismes d'inférences sur l'ontologie pour pouvoir l'utiliser.

8 Bibliographie

- AMBRIOLA, Vincenzo, CONRADI, Reidar, et FUGGETTA, Alfonso. Assessing process-centered software engineering environments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1997, vol. 6, no 3, p. 283-328.

- AOUSSAT, Fadila, OUSSALAH, Mourad, et NACER, Mohamed Ahmed. SPEM Extension with software process architectural concepts. In :*Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*. IEEE, 2011. p. 215-223.

- ARMENISE, P., BANDINELLI, S., GHEZZI, C., *et al.Survey and assessment the process representing formalisms*. GOODSTEP Technical Report No, 1993.

- ARPÍREZ, J. C., CORCHO, O., FERNÁNDEZ-LÓPEZ, M., *et al.*WebODE: a scalable ontological engineering workbench. In :*First International Conference on Knowledge Capture (K-CAP 2001). Victoria, Canada*. 2001.

- ARPIREZ, J., GÓMEZ-PÉREZ, Asuncion, LOZANO, Adolfo, *et al.* 2Agent: An ontology-based WWW broker to select ontologies. In :*Workshop on Applications of Ontologies and PSMs, Brighton, England*. 1998.

- ATKINSON, Colin. Meta-modelling for distributed object environments. In :*Enterprise Distributed Object Computing Workshop [1997]. EDOC'97. Proceedings. First International*. IEEE, 1997. p. 90-101.

- ATKINSON, Colin et KÜHNE, Thomas. The essence of multilevel metamodeling. In :*International Conference on the Unified Modeling Language*. Springer Berlin Heidelberg, 2001. p. 19-33.

- ATKINSON, Darren C., WEEKS, Daniel C., et NOLL, John. The design of evolutionary process modeling languages. In :*Software Engineering Conference, 2004. 11th Asia-Pacific*. IEEE, 2004. p. 73-82.

- ATL,PROJET ATLAS, INRIA. ATL: Atlas Transformation Language,http://www.eclipse.org/atl/. 2005.

- AVGERIOU, Paris, GUELFI, Nicolas, et MEDVIDOVIC, Nenad. Software architecture description and UML. In :*International Conference on the Unified Modeling Language*. Springer Berlin Heidelberg, 2004. p. 23-32.

- BACHIMONT, Bruno. Engagement sémantique et engagement ontologique: conception et réalisation d'ontologies en ingénierie des connaissances. *Ingénierie des connaissances: évolutions récentes et nouveaux défis*, 2000, p. 305-323.

- BACHIMONT, Bruno, ISAAC, Antoine, et TRONCY, Raphaël. Semantic commitment for designing ontologies: a proposal. In :*International Conference on Knowledge Engineering and Knowledge Management*. Springer Berlin Heidelberg, 2002. p. 114-121.

- BECHHOFER, S. Horrocks, GOBLE, I., STEVENS, C., *et al.* A Reason-able Ontology Editor for the Semantic Web. Joint German. In :*Austrian Conference on AI: Advances in Artificial Intelligence, LNCS*. 2001.

- BECK, Kent et FOWLER, Martin. *Planning extreme programming*. Addison-Wesley Professional, 2001.

- BECKER, Pablo, PAPA, Fernanda, et OLSINA, Luis. Process ontology specification for enhancing the process compliance of a measurement and evaluation strategy. *CLEI Electronic Journal*, 2015, vol. 18, no 1, p. 3-3.

- BENDRAOU, Reda, GERVAIS, Marie-Pierre, BLANC, Xavier, *et al.* Vers l'Exécutabilité des Modèles de Procédés Logiciels. In :*LangageModèles et Objets LMO'08*. 2008.

- BEZIVIN, Jean. Model Engineering for Software Modernization. In :*WCRE*. 2004. p. 4.

- BÉZIVIN, Jean et BRIOT, Jean-Pierre. Sur les principes de base de l'ingénierie des modèles. *L'OBJET*, 2004, vol. 10, no 4, p. 145-157.

- BÉZIVIN, Jean. On the unification power of models. *Software and systems modeling*, 2005, vol. 4, no 2, p. 171-188.

- BÉZIVIN, Jean et GERBÉ, Olivier. Towards a precise definition of the OMG/MDA framework. In :*Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*. IEEE, 2001. p. 273-280.

- BLANC, Xavier et SALVATORI, Olivier. *MDA en action: Ingénierie logicielle guidée par les modèles*. EditionsEyrolles, 2011.

- BLÁZQUEZ, Juan, FERNÁNDEZ, Mariano, GARCÍA-PINAR, Juan M., *et al.*Building ontologies at the knowledge level using the ontology design environment. 1998.

- BOHLEN, M., *et al.*AndroMDA Model Driven Architecture framework. 2007.

- BOOCH, Grady. Objectifying information technology. *Object Magazine*, 1993, vol. 3, no 3, p. 24-28.

- BORGO, Stefano, GUARINO, Nicola, et MASOLO, Claudio. Stratified ontologies: the case of physical objects. In :*Proceedings of ECAI-96 Workshop on Ontological Engineering*. 1996.

- BORST, Willem Nico. *Construction of engineering ontologies for knowledge sharing and reuse*. UniversiteitTwente, 1997.

- BOURQUE, Pierre, FAIRLEY, Richard E., *et al.Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

- BURTON-JONES, Andrew, STOREY, Veda C., SUGUMARAN, Vijayan, *et al.*A semiotic metrics suite for assessing the quality of ontologies. *Data & Knowledge Engineering*, 2005, vol. 55, no 1, p. 84-102.

- CAMARGO, Diana. *Implementation of Six Sigma's DMAIC and Lean Manufacturing Process Improvement Methodologies in a Software Development Environment: A Case Study*. 2006. Thèse de doctorat. Master Thesis.

- CERAVOLO, Paolo, DAMIANI, Ernesto, MARCHESI, Michele, *et al.*A ontology-based process modelling for XP. In :*Software Engineering Conference, 2003. Tenth Asia-Pacific*. IEEE, 2003. p. 236-242.

- CHANDRASEKARAN, Balakrishnan, JOSEPHSON, John R., et BENJAMINS, V. Richard. What are ontologies, and why do we need them?.*IEEE Intelligent Systems and their applications*, 1999, vol. 14, no 1, p. 20-26.

- CHARLET, Jean, LAUBLET, Philippe, et REYNAUD, Chantal. Web sémantique. Rapport final de l'action spécifique 32, CNRS. *STIC (version 3 de décembre 2003), publié chez Cépadues (Hors-série de la collection Information interaction intelligence)*, 2003.
- CHARLOTTE, H. U. G. *Méthode, modèles et outil pour la méta-modélisation des processus d'ingénierie de systèmes d'information*. 2009. Thèse de doctorat. UNIVERSITÉ JOSEPH FOURIER-GRENOBLE.

- CURTIS, Bill, KELLNER, Marc I., et OVER, Jim. Process modeling. *Communications of the ACM*, 1992, vol. 35, no 9, p. 75-90.

- CZARNECKI, Krzysztof et HELSEN, Simon. Classification of model transformation approaches. In :*Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*. 2003. p. 1-17.

- DE ALMEIDA FALBO, Ricardo, BARCELLOS, MonalessaPerini, NARDI, Julio Cesar, *et al.*Organizing ontology design patterns as ontology pattern languages. In :*Extended Semantic Web Conference*. Springer Berlin Heidelberg, 2013. p. 61-75.

- MCGUINNESS, Deborah L., VAN HARMELEN, Frank, *et al.* OWL web ontology language overview. *W3C recommendation*, 2004, vol. 10, no 10, p. 2004.

- DERNIAME, Jean-Claude, KABA, Badara A., et WASTELL, David (ed.). *Software process: principles, methodology, and technology*. Springer, 2006.

- DJURIĆ, Dragan, GAŠEVIĆ, Dragan, et DEVEDŽIĆ, Vladan. Ontology modeling and MDA. *Journal of Object technology*, 2005, vol. 4, no 1, p. 109-128.

- EBERT, Jürgen et ENGELS, Gregor. Structural and behavioural views on OMT-classes. In :*Object-Oriented Methodologies and Systems*. Springer Berlin Heidelberg, 1994. p. 142-157.

- E. M. F ,ECLIPSE. Eclipse modeling framework. *URL http://www. eclipse. org/modeling/emf*, 2006.

- ESTUBLIER, Jacky. *Sonia JAMAL (épouse SANLAVILLE)*." Environnement de procédé extensible pour l'orchestration Application aux services web ". 2005. Thèse de doctorat. Xerox Research Centre Europe.

- FALBO, Ricardo De Almeida et BERTOLLO, Gleidson. A software process ontology as a common vocabulary about software processes. *International Journal of Business Process Integration and Management*, 2009, vol. 4, no 4, p. 239-250.

- FARQUHAR, Adam, FIKES, Richard, PRATT, Wanda, *et al.Collaborative ontology construction for information integration*. Technical Report KSL-95-63, Stanford University Knowledge Systems Laboratory, 1995.

- FAVRE, Jean-Marie. Foundations of meta-pyramids: Languages vs. metamodels--episode ii: Story of thotus the baboon1. In :*Dagstuhl Seminar Proceedings*. SchlossDagstuhl-Leibniz-ZentrumfürInformatik, 2005.

- FEILER, Peter H. et HUMPHREY, Watts S. Software process development and enactment: Concepts and definitions. In :*Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the*. IEEE, 1993. p. 28-40.

- FERNÁNDEZ-LÓPEZ, Mariano, GÓMEZ-PÉREZ, Asunción, et JURISTO, Natalia. Methontology: from ontological art towards ontological engineering. AAAI Technical Report SS-97-06 1997.

- FINKELSTEIIN, Anthony, KRAMER, Jeff, et NUSEIBEH, Bashar. *Software process modelling and technology*. John Wiley & Sons, Inc., 1994.

- FUJITA, H. et ZUALKERNAN, I. A. An ontology-driven approach for generating assessments for the scrum software process. *Proceedings of the seventh SoMeT_08. IOS Press, The Netherlands*, 2008, p. 190-205.

- FÜRST, Frédéric. L'ingénierie ontologique, rapport technique. *Institut de recherche en Informatique de Nantes*, 2002.

- FÜRST, Frédéric. *Contribution à l'ingénierie des ontologies: une méthode et un outil d'opérationalisation*. 2004. Thèse de doctorat. Nantes.

- Garcia Camargo, S. *Ingénierie concurrente en génie logiciel: Céline*. 2006. Thèse de doctorat. Université Joseph-Fourier-Grenoble I.

- GAŠEVIĆ, Dragan, KAVIANI, Nima, et MILANOVIĆ, Milan. Ontologies and software engineering. In :*Handbook on Ontologies*. Springer Berlin Heidelberg, 2009. p. 593-615.

- GAZEL, Sema, SEZER, EbruAkçapinar, et TARHAN, Ayca. An ontology based infrastructure to support CMMI-based software process assessment. *Gazi University Journal of Science*, 2012, vol. 25, no 1, p. 155-164.

- GEIGER, Leif et ZÜNDORF, Albert. TOOL modeling with Fujaba. *Electronic Notes in Theoretical Computer Science*, 2006, vol. 148, no 1, p. 173-186.

- GIARETTA, Pierdanieleet GUARINO, N. Ontologies and knowledge bases towards a terminological clarification. *Towards very large knowledge bases: knowledge building & knowledge sharing*, 1995, vol. 25, p. 32.

- GIESE, Holger et WAGNER, Robert. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, 2009, vol. 8, no 1, p. 21-43.

- GÓMEZ-PÉREZ, Asunción. Evaluation of taxonomic knowledge in ontologies and knowledge bases.*Proceedings of the Banff Knowledge Acquisition for Knowledge-Based SystemsWorkshop. KAW'99 Volumen: II Editor: Gaines, B.R.; Musen, M.*,1999.

- GÓMEZ-PÉREZ, Asuncion, FERNÁNDEZ-LÓPEZ, Marianno, et CORCHO, Oscar. Ontological Engineering. Advanced Information and Knowlege Processing. 2003.

- GÓMEZ-PÉREZ, Asunción, RAMÍREZ, Jaime, et VILLAZÓN-TERRAZAS, Boris. An ontology for modelling human resources management based on standards. In :*Knowledge-Based Intelligent Information and Engineering Systems*. Springer Berlin/Heidelberg, 2007. p. 534-541.

- GONZALEZ-PEREZ, Cesar. Supporting situational method engineering with ISO/IEC 24744 and the work product pool approach. In :*Situational Method Engineering: Fundamentals and Experiences*. Springer US, 2007. p. 7-18.

- GONZALEZ-PEREZ, C., HENDERSON-SELLERS, B., MCBRIDE, T., *et al.* An Ontology for ISO software engineering standards: 2) Proof of concept and application. *Computer Standards & Interfaces, 2016, vol. 48, p. 112-123.*

- GRUBER, Thomas R., *et al.* A translation approach to portable ontology specifications. *Knowledge acquisition*, 1993, vol. 5, no 2, p. 199-220.

- GRÜNINGER, Michael et FOX, Mark S. Methodology for the Design and Evaluation of Ontologies.Workshop on Basic Ontological Issues in Knowledge Sharing, 1995.

- GUARINO, N. et WELTY, C. A formal ontology of properties knowledge engineering and knowledge management: Methods models and tools. In :*12th International Conference (EKAW 2000)*. p. 97-112.

- GUERRA, Esther et DE LARA, Juan. Event-driven grammars: Towards the integration of meta-modelling and graph transformation. In :*International Conference on Graph Transformation*. Springer Berlin Heidelberg, 2004. p. 54-69.

- GUIZZARDI, Giancarlo, DE ALMEIDA FALBO, Ricardo, et GUIZZARDI, Renata SS. Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology. In :*CIbSE*. 2008. p. 127-140.

- HAMRI, Mehdi Mohamed et BENSLIMANE, Sidi Mohamed. Building an Ontology for the Metamodel ISO/IEC24744 using MDA Process. *International Journal of Modern Education and Computer Science*, 2015, vol. 7, no 8, p. 48.

- HAMRI, Salah. *Interopérabilité des Modèles de Workflow.2012* Thèse de doctorat. UNIVERSITÉMENTOURI DE CONSTANTINE.

- HAPPEL, Hans-Jörg et SEEDORF, Stefan. Applications of ontologies in software engineering. In :*International workshop on semantic web enabled software engineering (SWESE'06)*. 2006. p. 1-14.

- HENDERSON-SELLERS, Brian. Towards the formalization of relationships for object modelling. In :*Technology of Object-Oriented Languages and Systems, 1997. TOOLS 25, Proceedings*. IEEE, 1997. p. 267-283.

- HENDERSON-SELLERS, Brian et GONZALEZ-PEREZ, Cesar. A comparison of four process metamodels and the creation of a new generic standard. *Information and software technology*, 2005, vol. 47, no 1, p. 49-65.

- HENDERSON-SELLERS, Brian et GONZALEZ-PEREZ, Cesar. On the ease of extending a powertype-based methodology metamodel. *Meta-Modelling and. WoMM*, 2006, vol. 2006, p. 11-25.

- HENDERSON-SELLERS, Brian, GONZALEZ-PEREZ, Cesar, MCBRIDE, Tom, *et al.* An ontology for ISO software engineering standards: 1) Creating the infrastructure. *Computer Standards & Interfaces*, 2014, vol. 36, no 3, p. 563-576.

- HENDERSON-SELLERS, Brian, MCBRIDE, Tom, LOW, Graham, *et al.* Ontologies for international standards for software engineering. In :*International Conference on Conceptual Modeling*. Springer Berlin Heidelberg, 2013. p. 479-486.

- HNETYNKA, Petr et PISE, Michal. Hand-written vs. MOF-based metadata repositories: the SOFA experience. In :*Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*. IEEE, 2004. p. 329-336.

- HUFF, Karen. Software process modeling. *Software Process*, 1996, vol. 5, p. 1-24.
- ISO/IEC ((2007)). "ISO/IEC 24744. Software Engineering – Metamodel for Development Methodologies." ISO, Geneva (2007).

- JACOBSON, Ivar, CHRISTERSON, Magnus, et CONSTANTINE, Larry L. The OOSE method: a use—case-driven approach. In : *Object developmentmethods*. SIGS Publications, Inc., 1994. p. 247-270.

- JÉZÉQUEL, Jean-Marc, DEFOUR, Olivier, et PLOUZEAU, Noël. An MDA approach to tame component based software development. In :*International Symposium on Formal Methods for Components and Objects*. Springer Berlin Heidelberg, 2003. p. 260-275.

- JOUAULT, Frédéric. *Contribution à l'étude des langages de transformation de modèles*. 2006. Thèse de doctorat. Nantes.

- JOUAULT, Frédéric, ALLILAIRE, Freddy, BÉZIVIN, Jean, *et al.*ATL: A model transformation tool. *Science of computer programming*, 2008, vol. 72, no 1, p. 31-39.

- KABBAJ, Mohammed Issam. État de l'art sur l'adaptation dynamique des Procédés de développement de logiciels. Rapport IRIT Toulouse, 2009.

- KASSEL, Gilles. OntoSpec: une méthode de spécification semi-informelle d'ontologies. *Actes des 13e journées francophones d'Ingénierie des Connaissances*, 2002, p. 75-87.

- KAY, Michael. *XSLT programmer's reference*. Wrox Press Ltd., 2000.

- SCHWABER, K. Agile Project Management With Scrum: Microsoft. 2004.

- KLEPPE, Anneke G., WARMER, Jos, BAST, Wim, *et al.* The model driven architecture: practice and promise. 2003.

- KURTEV, Ivan. State of the art of QVT: A model transformation language standard. In :*International Symposium on Applications of Graph Transformations with Industrial Relevance*. Springer Berlin Heidelberg, 2007. p. 377-393.

- Kassel, Germany*, A Case Study for Graph Transformation Tools, Applications of Graph October 10-12, 2007, Revised Selected and Invited Papers, Springer-Verlag, Berlin, Heidelberg.*

- LEE, Chang-Shinget WANG, Mei-Hui. Ontology-based computational intelligent multi-agent and its application to CMMI assessment. *Applied Intelligence*, 2009, vol. 30, no 3, p. 203-219.

- LEE, Jae Kyu et MIZOGUCHI, Riichiro. Guest editors' note Research frontier in the Far East. *Expert Systems with Applications*, 1995, vol. 9, no 1, p. 1.

- LIAO, L. Yuzhong Q. Leung HKN: Software process ontology and its application. In :*4th International Semantic Web Conference Galway*. 2005.

- LIN, Yishuai, HILAIRE, Vincent, GAUD, Nicolas, *et al.*Scrum conceptualization using K-CRIO ontology. In :*International Symposium on Data-Driven Process Discovery and Analysis*. Springer Berlin Heidelberg, 2011. p. 189-211.

- LÍŠKA, Miroslav et NAVRAT, Pavol. An approach to project planning employing software and systems engineering meta-model represented by an ontology. *Computer Science and Information Systems*, 2010, vol. 7, no 4, p. 721-736.

- LÍŠKA, Miroslav et NÁVRAT, Pavol. An ontology driven approach to software project enactment with a supplier. In :*East European Conference on Advances in Databases and Information Systems*. Springer Berlin Heidelberg, 2010. p. 378-391.

- LORSZ, MARTA. The software process: Modeling, evaluation and improvement. *Handbook of Software Engineering and Knowledge Engineering*, 2001, vol. 1, p. 193.

- MAUW, Sjouke, WIERSMA, Wouter T., et WILLEMSE, Tim AC. Language-driven system design. In :*System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*. IEEE, 2002. p. 3637-3646.

- MENDES, Olavo et ABRAN, Alain. Issues in the Development of an Ontology for a Emerging Engineering Discipline. In :*SEKE*. 2005. p. 139-144.

- MONTANGERO, Carlo. The software process: Modelling and technology. *Software process: principles, methodology, and Technology*, 1999, p. 1-13.

- MOSAIC, X. M. F. The Xactium XMF Mosaic. 2007.

- NECHES, Robert, FIKES, Richard E., FININ, Tim, *et al.* Enabling technology for knowledge sharing. *AI magazine*, 1991, vol. 12, no 3, p. 36.

- NOY, Natalya Fridman, MUSEN, Mark A., *et al.*Algorithm and tool for automated ontology merging and alignment. In :*Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00). Available as SMI technical report SMI-2000-0831*. 2000.

- NOY, Natalya Fridman, MUSEN, Mark A., *et al.*Promptdiff: A fixed-point algorithm for comparing ontology versions. *AAAI/IAAI*, 2002, vol. 2002, p. 744-750.

- ODELL, James J. Power types. *Journal of Object-Oriented Programming*, 1994, vol. 7, no 2, p. 8-&.

- OMG(2003), MDA. Guide v1. 0.1. *Disponible à l'adresse http://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf,*June 2003.

- OMG, (2008). Software & Systems Process Engineering Metamodel™ Specification (SPEM™) Version 2.0. *Disponible à l'adresse*http://www.omg.org/spec/SPEM/2.0/,April 2008

- OMG. (2011a).UML, OMG. *2.4. 1 superstructure specification*. document formal/2011-08-06. Technical report, OMG, 2011.

- OMG. (2011b). QVT, OMG MOF 2.0 Query/View/Transformation Specification (OMG QVT) Version 1.1. *Object Management Group. http://www. omg. org/spec/QVT/1.1*, 2011.

- OMG. (2013). "MOF 2.4.1 Meta Object Facility™.*http://www.omg.org/spec/MOF/2.4.1/.2013*

- OMG. (2014). Documents Associated With Ontology Definition Metamodel™ (ODM™)Version 1.1.*http://www.omg.org/spec/ODM/1.1/.2014*

- OSTERWEIL, Leon. Software processes are software too. In :*Proceedings of the 9th international conference on Software Engineering*. IEEE Computer Society Press, 1987. p. 2-13.

- PALMER, Steve R. et FELSING, Mac. *A practical guide to feature-driven development*. Pearson Education, 2001.

- Papyrus "http://eclipse.org/papyrus/."

- PARDO, César, PINO, Francisco J., GARCÍA, Félix, *et al.*An ontology for the harmonization of multiple standards and models. *Computer Standards & Interfaces*, 2012, vol. 34, no 1, p. 48-59.

- PARSON, David. Agile software development methodology, an ontological analysis. 2011.

- PARSONS, David. An ontology of agile aspect oriented software development. 2011.*Inf. Math. Sci., 2011, Vol. 15, pp. 1–11*

- POOLE, John, CHANG, Dan, TOLBERT, Douglas, *et al.Common warehouse metamodel developer's guide*. John Wiley& Sons, 2003.

- PSYCHÉ, V. État de l'art sur l'ontologie-application au téléapprentissage. *Note de recherche. Montréal, Canada, Centre de recherche LICEF, Télé-université*, 2003.

- REVAULT, Nicolas, SAHRAOUI, Houari A., BLAIN, Gilles, *et al.*A Metamodeling technique: The METAGEN system. In : *Proceedings of TOOLS*. 1995. p. 127-139.

- RICHTERS, Mark et GOGOLLA, Martin. OCL: Syntax, semantics, and tools. In :*Object Modeling with the OCL*. Springer Berlin Heidelberg, 2002. p. 42-68.

- ROBIE, Jonathan. XML Processing and data integration with XQuery. *IEEE Internet Computing*, 2007, vol. 11, no 4.

- RODRÍGUEZ, Daniel et SICILIA, Miguel-Ángel. Defining spem 2 process constraints with semantic rules using swrl. *Proceedings of ONTOSE*, 2009, p. 96.

- ROLLAND, Colette. A comprehensive view of process engineering. In :*Advanced information systems engineering*. Springer Berlin/Heidelberg, 1998. p. 1-24.

- RUIZ, Francisco et HILERA, José R. Using ontologies in software engineering and technology. In :*Ontologies for software engineering and software technology*. Springer Berlin Heidelberg, 2006. p. 49-102.

- RUY, Fabiano Borges, DE ALMEIDA FALBO, Ricardo, BARCELLOS, MonalessaPerini, *et al.*An Ontological Analysis of the ISO/IEC 24744 Metamodel. In :*FOIS*. 2014. p. 330-343.

- RUY, Fabiano B., FALBO, Ricardo A., BARCELLOS, Monalessa P., *et al.* An ISO-based software process ontology pattern language and its application for harmonizing standards. *ACM SIGAPP Applied Computing Review*, 2015, vol. 15, no 2, p. 27-40.

- SANTANA, A., LÓSCIO, B. F., et FREITAS, F. A Scrum Application Ontology for Support Historical Data Management of Projects. *Centro de Informática—Universidade Federal de Pernambuco, Recife*, 2013.

- SEIDEWITZ, Edwin. What models mean. *IEEE software*, 2003, vol. 20, no 5, p. 26-32.

- SHARIFLOO, Amir Azim, SHAMSFARD, Mehrnoush, MOTAZEDI, Yasaman, *et al.* An ontology for cmmi-acq model. In :*Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*. IEEE, 2008. p. 1-6.

- SICILIA, M. A., CUADRADO, Juan-José, GARCÍA, Elena, *et al.*The evaluation of ontological representation of the SWEBOK as a revision tool. In :*29th Annual International Computer Software and Application Conference (COMPSAC), Edinburgh, UK*. 2005. p. 26-28.

- SIDDIQUI, Farheen et ALAM, M. Afshar. Ontology Based Feature Driven Development Life Cycle. *arXiv preprint arXiv:1307.4174*, 2013.

- SILAGHI, Raul, FONDEMENT, Frédéric, et STROHMEIER, Alfred. “Weaving” MTL model transformations. In :*Model Driven Architecture*. Springer Berlin Heidelberg, 2005. p. 123-138.

- SMITH, Barry et ONTOLOGY, L. Floridi. Blackwell guide to the philosophy of computing and information. *Blackwell Guide to the Philosophy of Computing and Information*, 2003.

- SOLEY, Richard, *et al.* Model driven architecture. *OMG white paper*, 2000, vol. 308, no 308, p. 5.

- SOYDAN, GokhanHalitet KOKAR, M. An OWL ontology for representing the CMMI-SW model. In :*Workshop on Semantic Web Enabled Software Engineering (SWESE)*. 2006.

- STUDER, Rudi, BENJAMINS, V. Richard, et FENSEL, Dieter. Knowledge engineering: principles and methods. *Data &knowledge engineering*, 1998, vol. 25, no 1-2, p. 161-197.

- SURE, York, ERDMANN, Michael, ANGELE, Jürgen, *et al.*OntoEdit: Collaborative ontology development for the semantic web. In: *International Semantic Web Conference*. Springer Berlin Heidelberg, 2002. p. 221-235.

- TANKOANO, Joachim, DERNIAME, Jean-Claude, et KABA, Ali. Software process design based on products and the object oriented paradigm. *Software Process Technology*, 1994, p. 177-185.

- TEAM, CMMI Product. CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Continuous Representation. *CMU/SEI*, 2002.

- TOLVANEN, Juha-Pekka et ROSSI, Matti. MetaEdit+: defining and using domain-specific modeling languages and code generators. In :*Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 2003. p. 92-93.

- USCHOLD, Mike et GRUNINGER, Michael. Ontologies: Principles, methods and applications. *The knowledge engineering review*, 1996, vol. 11, no 02, p. 93-136.

- USCHOLD, Michael et KING, Martin. *Towards a methodology for building ontologies*. Edinburgh : Artificial Intelligence Applications Institute, University of Edinburgh, 1995.

- VALASKI, Joselaine, MALUCELLI, Andreia, REINEHR, Sheila S., *et al.*Ontology to Classify Learning Material in Software Engineering Knowledge Domain. In :*ONTOBRAS-MOST*. 2011. p. 37-47.

- VAN HEIJST, Gertjan, SCHREIBER, A. Th, et WIELINGA, Bob J. Using explicit ontologies in KBS development. *International journal of human-computer studies*, 1997, vol. 46, no 2-3, p. 183-292.

- VIZCAÍNO, Aurora, GARCÍA, Felix, PIATTINI, Mario, *et al.* A validated ontology for global software development. *Computer Standards & Interfaces, 2016, vol. 46, p. 66-78.*

- WANG, Shengjun, JIN, Longfei, et JIN, Chengzhi. Represent software process engineering metamodel in description logic. *self*, 2006, vol. 10, p. 16.

- WILLE, Cornelius, ABRAN, Alain, DESHARNAIS, Jean Marc, *et al.*The Quality concepts and sub concepts in SWEBOK: An ontology challenge. In :*International Workshop on Software Measurement (IWSM), Montreal*. 2003. p. 18.

- WILLE, Cornelius, DUMKE, Reiner R., ABRAN, Alain, *et al.* E-learning infrastructure for software engineering education: Steps in ontology modeling for SWEBOK. In :*IASTED Conf. on Software Engineering*. 2004. p. 520-525.

- ZAMLI, Kamal Zuhairi. Process modeling languages: A literature review. *Malaysian Journal of Computer Science*, 2001, vol. 14, no 2, p. 26-37.

- ZAMLI, Kamal Zuhairi, ISA, Mat, et ASHIDI, Nor. A survey and analysis of process modeling languages. *MalaysiaN Journal of Science*, 2004, vol. 17, no 2, p. 68-89.

- ZUALKERNAN, I et FUJITA, H. . An ontology-driven approach for generating assessments for the scrum software process. *Proceedings of the seventh SoMeT_08. IOS Press, The Netherlands*, 2008, p. 190-205.

9 Annexe

Nous présentons dans cette annexe, un extrait du code OWL de l'ontologie ISO/IEC 24744 que nous avons généré.

```
<rdf:RDF xmlns="IsoOntology#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-
ns#" xmlns:owl="http://www.w3.org/2002/07/owl#"xmlns:xsd="http://www.w3.org
/2001/XMLSchema#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#" xml:base="IsoOntology">
<owl:Ontology rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Iso
Ontology"/>
<!--


///////////////////////////////////////////////////////////////////////
////////////
    //
    // Object Properties
    //

///////////////////////////////////////////////////////////////////////
////////////

-->
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Actsup
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Actsup">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#action"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#workproduct"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Agent
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Agent">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#producer"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#workPerformance"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Causes
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Causes">
<owl:inverseOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier
/IsoOntology#Effect"/>
```

```
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#action"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#task"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Child
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Child">
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#process"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#process"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Compostion
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Compostion">
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#task"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#taskTechniqueMapping"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#technique"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Context
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Context">
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#task"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#workunit"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Depicts
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Depicts">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#document"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#workproduct"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Effect
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Effect">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#action"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#task"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Job
-->
```

```
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Job">
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#workPerformance"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#workunit"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#KindOf
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#KindOf">
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#ActionKind"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#RoleKind"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#TeamKind"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#ToolKind"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#action"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#person"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#producer"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#producerKind"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#role"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#team"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#tool"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Parent
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Parent">
<owl:inverseOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier
/IsoOntology#Child"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#process"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#process"/>
</owl:ObjectProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Plays
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Plays">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#producer"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#role"/>
</owl:ObjectProperty>
<!--

file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Tomporelle_conte
xt
```

```
-->
<owl:ObjectProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Tomporelle_context">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#process"/>
<rdfs:range rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/Is
oOntology#stageWithDuration"/>
</owl:ObjectProperty>
<!--


///////////////////////////////////////////////////////////////////////
////////////
    //
    // Data properties
    //

///////////////////////////////////////////////////////////////////////
////////////

-->
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#EndTime
-->
<owl:DatatypeProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dos
sier/IsoOntology#EndTime">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#stageWithDuration"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#workunit"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Justification
-->
<owl:DatatypeProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dos
sier/IsoOntology#Justification">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#workPerformance"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#StartTime
-->
<owl:DatatypeProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dos
sier/IsoOntology#StartTime">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#stageWithDuration"/>
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#workunit"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Time-reached
-->
<owl:DatatypeProperty rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dos
sier/IsoOntology#Time-reached">
<rdfs:domain rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/I
soOntology#instantaneouStage"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
```

```
<!--


///////////////////////////////////////////////////////////////////////
////////////
    //
    // Classes
    //

///////////////////////////////////////////////////////////////////////
////////////

-->
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#ActionKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#ActionKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Template"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Constraint
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#Constraint">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#resource"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#ConstraintKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#ConstraintKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Template"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#DocumentKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#DocumentKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workProductKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Endeavour
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#Endeavour"/>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Guideline
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#Guideline">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#resource"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Language
-->
```

```
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#Language">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#resource"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#ModelKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#ModelKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workProductKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Notation
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#Notation">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#resource"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Outcome
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#Outcome">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Template"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Postcondition
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#Postcondition">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#ConstraintKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Precondition
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#Precondition">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#ConstraintKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#ProcessKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#ProcessKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workunitKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#RoleKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#RoleKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#producerKind"/>
</owl:Class>
```

```
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#TaskKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#TaskKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workunitKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#TeamKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#TeamKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#producerKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#TechniqueKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#TechniqueKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workunitKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#Template
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#Template"/>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#ToolKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#ToolKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#producerKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#action
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#action">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Endeavour"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#buildKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#buildKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#stageKind"/>
</owl:Class>
<!--

file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#compositeWorkPro
duct
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#compositeWorkProduct">
```

```
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workproduct"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#document
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#document">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workproduct"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#hardwareItem
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#hardwareItem">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workproduct"/>
</owl:Class>
<!--

file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#instantaneouStag
e
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#instantaneouStage">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#stage"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#model
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#model">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workproduct"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#person
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#person">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#producer"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#phaseKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#phaseKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#stageKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#process
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#process">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workunit"/>
</owl:Class>
```

```
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#producer
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#producer">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Endeavour"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#producerKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#producerKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Template"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#resource
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#resource"/>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#role
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#role">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#producer"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#softwareItem
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#softwareItem">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workproduct"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#stage
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#stage">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Endeavour"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#stageKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#stageKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Template"/>
</owl:Class>
<!--

file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#stageWithDuratio
n
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#stageWithDuration">
```

```
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#stage"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#task
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#task">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workunit"/>
</owl:Class>
<!--

file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#taskTechniqueMap
ping
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#taskTechniqueMapping">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Endeavour"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#team
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#team">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#producer"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#technique
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#technique">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#workunit"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#timeCycleKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#timeCycleKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#stageKind"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#tool
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#tool">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#producer"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#workPerformance
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#workPerformance">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Endeavour"/>
</owl:Class>
```

```
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#workProductKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#workProductKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Template"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#workproduct
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#workproduct">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Endeavour"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#workunit
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#workunit">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Endeavour"/>
</owl:Class>
<!--
 file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOntology#workunitKind
-->
<owl:Class rdf:about="file:/D:/Users/mehdi/Desktop/Nouveau%20dossier/IsoOnt
ology#workunitKind">
<rdfs:subClassOf rdf:resource="file:/D:/Users/mehdi/Desktop/Nouveau%20dossi
er/IsoOntology#Template"/>
</owl:Class>
</rdf:RDF>
<!--
 Generated by the OWL API (version 3.5.0) http://owlapi.sourceforge.net
-->
```