Gestion de modèles : définitions, besoins et revue de littérature

Dinh Thi-Lan-anh[1] , Gerbé Olivier[2] , Sahraoui Houari[1]

[1] Université de Montréal, DIRO, CP 6128 succ CV, Montréal, QC Canada, H3C 3J7
[2] HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Québec, Canada H3T 2A7

Résumé

La gestion de modèles intéresse de nombreuses communautés de recherche. L'Object Management Group) travaille sur MDA (Model Driven Architecture). Les communautés travaillant sur la gestion des connaissances, la gestion de méta-données, les ontologies, la qualité de service et le génie logiciel s'intéresse à la définition et l'utilisation des modèles. Nous faisons dans cet article, le point sur les définitions et besoins pour la gestion de modèles et passons en revue divers formalismes de modélisation : sNets, graphes conceptuels, UML (Unified Modeling Language) et MOF (Meta Object Facility), XML (eXtensible Markup Language) et XML Schema, RDF (Resource description Framework) et RDF Schema, et OWL (Web Ontology Language).

1. Introduction

La gestion de modèles intéresse de nombreuses communautés de recherche. L'Object Management Group) travaille sur MDA (Model Driven Architecture). Les communautés travaillant dans les domaines du génie logiciel, de la gestion des connaissances, de la gestion de méta-données, des ontologies et de la qualité de service s'intéressent à la définition et l'utilisation des modèles.

La notion de modèle est utilisé pour représenter quelques aspects de la réalité. Il est codé comme un ensemble d'éléments et de relations entre eux. La gestion de modèles concerne les mécanismes permettant de représenter, créer, stocker et manipuler les modèles.

Dans le domaine du génie logiciel, l'OMG a lancé MDA dont les concepts sont orientés-modèles plutôt que orientés-objets. MDA [27] offre le pouvoir d'abstraction, de raffinement et de vues différentes sur les modèles. Et surtout, elle donne la possibilité de concevoir des modèles indépendants des plate-formes et de l'environnement d'implémentation.

Dans le domaine de la gestion des connaissances, ces dernières doivent être modélisées afin d'être gérées. La gestion des connaissances dans l'entreprise, la mémoire d'entreprise, pose un problème de quantité, de complexité et de diversité [10]. Ceci implique un challenge pour la représentation et la modélisation de cette mémoire.

Dans le domaine des ontologies, la représentation et la modélisation des ontologies sont des questions évidemment primordiales.

Dans le domaine des bases de données, la gestion des méta-données touche la manipulation de la structure de données plutôt que les données elle-mêmes. L'approche basée sur la gestion de modèles est une solution possible aux problèmes d'intégration et de transformation de données [1, 3, 23].

Enfin dans le domaine de la qualité de service (QoS), afin d'assurer le fonctionnement des applications dans un environnement complexe tel que les systèmes multimédias répartis, l'intégration des informations de gestion devient fondamental [19]. Une solution possible est de développer des modèles génériques pour les systèmes, les applications et les utilisateurs et d'ensuite développer des mécanismes d'intégration [15].

Cet article présente les principales définitions, les principaux besoins et une revue de littérature pour la représentation de modèles, un axe de recherche essentiel dans la gestion de modèles. Le reste de l'article est organisé comme suit. La section 2 décrit l'architecture des niveaux de modélisations. Les notions de base pour la compréhension de l'article sont présentées dans la section 3. La section 4 expose les besoins relatifs à la représentation. La section 5 passent en revue les principaux formalismes de modélisation et métamodélisation et les évaluent par rapport aux besoins. Enfin la dernière section conclut et présente les travaux futurs.

2. Architecture de modélisation

La modélisation est une activité qui consiste à transformer des descriptions informelles de la réalité en des descriptions formelles, appelées modèles, afin de pouvoir opérationnaliser les connaissances. La métamodélisation est une activité qui consiste à définir le vocabulaire et la grammaire, appelés métamodèles, permettant la réalisation de modèles. Enfin la méta-métamodélisation est une activité qui consiste à définir le vocabulaire et la grammaire, appelés méta-métamodèles, permettant la réalisation de métamodèles. Ces trois définitions correspondent aux trois niveaux de modélisation appelés : M1, M2 et M3. À ces trois niveaux de modélisation, on ajoute le monde réel que l'on cherche à modéliser, c'est le niveau M0.

Aujourd'hui semble émerger un consensus sur cette architecture à quatre niveaux. La figure 1 présente cette architecture et nous détaillons ci-dessous chacun des niveaux :

- M3 (méta-métamodèle) est le niveau le plus abstrait de cette architecture. Il définit les notions de base permettant la représentation des niveaux inférieurs ainsi que lui-même.
- [•] M2 est le niveau métamodèle. En utilisant le vocabulaire et la grammaire spécifiée au niveau M3, ce niveau définit le vocabulaire et la grammaire pour représenter des modèles au niveau M1.
- M1 est le niveau modèle. Ce niveau définit l'ontologie utilisée pour décrire l'application du le monde réel. Chaque modèle au niveau M1 respecte le vocabulaire et la grammaire spécifiée par son métamodèle au niveau M2.
- M0 est le monde réel décrit au niveau M1.

Il est à noter une évolution dans les architectures de modélisation. Si le nombre de niveaux est inchangé, le rôle des niveaux a été modifié. Dans VODAK [9], dans IRDS [18], dans Telos [26], dans Modèle uniforme [14], UML1.x, etc. le niveau M0 représentait les instances et le monde réel n'était pas pris en compte. La clarification des activités de modélisation a permis de préciser : (i) seulement les trois derniers niveaux (M3, M2, M1) appartiennent au monde de la modélisation tandis que le

niveau M0 présentant le monde réel n'y appartient pas ; (ii) les types et les instances, contrairement à ce qui est souvent perçu, sont au même niveau M1 [22, 5]. Dans la figure 1, Mary est une Instance dans le contexte global (qui est spécifié par la relation meta sur l'axe vertical), et aussi une instance de Person dans le contexte local (qui est spécifié par la relation InstOf sur l'axe horizontal). Cela résout le problème de la double instanciation [5, 11].

Cependant certaines questions importantes restent encore à discuter : Quel est le nombre de niveaux d'abstraction nécessaires pour la représentation des connaissances ? Quels sont les critères sémantiques permettant caractériser la frontière entre les niveaux de modélisation ? Quels sont les éléments dans chaque niveau de modélisation ?

Notons que la réflexivité du niveau M3 dans l'architecture de modélisation possède des avantages multiples. Elle permet de limiter le nombre de niveaux d'abstraction. Ce niveau réflexif se valide lui-même. Les outils et algorithmes applicables au niveau métamodèle sont applicables aussi à ce niveau [22]. Ceci facilite l'adaptation aux extensions ou modifications futures.

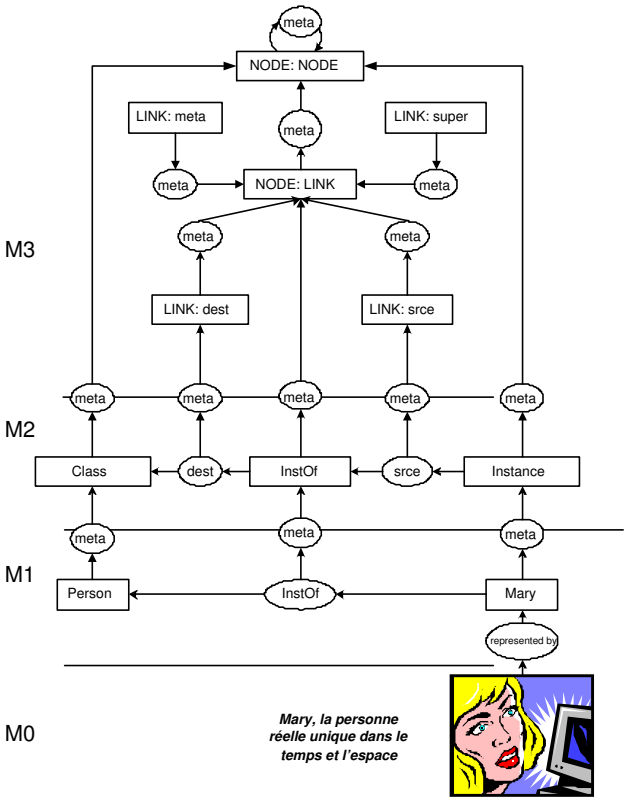

Figure 1. *Exemple d'architecture de modélisation.*

3. Notions de base

Nous présentons dans cette section les définitions des notions de base que nous utiliserons dans la suite de cet article. D'abords nous précisons la notion de modèle puis nous précisons les définitions des éléments associés à cette notion de modèle.

3.1. Modèle

Au niveau conceptuel, la notion de modèle est proche de celle de contexte visant la contextualisation des connaissances, mais la notion de contexte est encore discutée [34, 36, 38]. Dans le domaine de la modélisation, la notion de contexte est différente, on parle de : space dans [4], Universe dans sNets [22], SubjectArea dans CDIF [13], Package dans UML et dans MOF.

Dans cet article, nous appellerons modèles tous ce que nous pouvons retrouver dans les différents niveaux de modélisation. Nous citons quelques exemples pour chaque niveau. Au niveau M1, un modèle peut être un schéma de données, une représentation d'une fonction ou d'un comportement d'un système, une ontologie, un modèle de l'information de la qualité de service, un document XML ou encore un modèle RDF. Au niveau M2, un modèle peut être le métamodèle UML, le métamodèle Entité-Association, RDF-S. Au niveau M3 un modèle peut être la spécification d'un formalisme réflexif tel que MOF, CDIF ou sNets.

Définition 1 *Un modèle, qui est un codage d'une représentation de quelques aspects de l'univers du discours, contient un ensemble d'éléments et de relations entre ces éléments.*

Parmi les modèles nous distinguons les métamodèles et méta-métamodèles en utilisant les règles suivantes [22] : (i) un modèle appartient au M3 s'il est aussi son propre métamodèle ; (ii) un modèle appartient au M2 si son métamodèle est un modèle de niveau M3 ; (iii) un modèle appartient à M1 si son métamodèle est un modèle de niveau M2.

3.2. Autres notions

Voici certains autres éléments fondamentaux qui permettent de représenter la structure des connaissances.

Définition 2 *Catégorie (ou classe, type) : un ensemble de choses ayant les mêmes propriétés (attributs, relations et comportement).*

Définition 3 *Instance d'une catégorie : une chose qui se conforme à la définition de la catégorie.*

Définition 4 *Type relationnel (ou association) : un ensemble de relations ayant les mêmes propriétés entre instances et/ou catégories.*

Définition 5 *Lien : une relation binaire entre instances et/ou catégories qui se conforme à la définition d'un type relationnel.*

Définition 6 *Distinction entre relations instanciées d'un même type : soient deux relations d'un même type : r1 reliant e1, e2, ... eN ; et r2 reliant e'1, e'2, ... e'N. Si les deux séquence (e1, e2, ... eN) et (e'1, e'2, ... e'N) sont différentes, alors r1 et r2 sont distinctes ; sinon, r1 et r2 sont considérées comme identiques.*

Définition 7 *Spécialisation (ou héritage) : une relation de type est-une-sorte-de entre deux catégories spécifie qu'une catégorie hérite des propriétés de la catégorie supérieure.*

Définition 8 *Classification : une hiérarchie des catégories construites à partir des propriétés héritées. La spécialisation forme une classification des catégories.*

Définition 9 *Classification dynamique et Classification statique : pour une classe, la classification dynamique divise exhaustivement l'espace d'états de ses instances en sous-espaces disjoints, et la classification statique divise exhaustivement l'espace d'objets de ses instances en sous-espaces disjoints [40].*

4. Les besoins

4.1. Besoins pour le niveau M2

Nous avons répertoriés les besoins que les métamodèles de niveau M2 devraient supporter pour représenter des connaissances/modèles au niveau M1.

Besoin 1 *Relation n-aire. Une relation n-aire relie n éléments où n est un entier positif. Nous avons donc des relations unaires (n=1), binaires (n=2), ternaire (n=3), etc.*

Besoin 2 *Héritage entre types et héritage entre types relationnels. Parce qu'un type relationnel peut avoir ses propres attributs, comportements et relations avec d'autres éléments, l'héritage entre types relationnels est aussi important que celui entre types non-relationnels.*

Besoin 3 *Multi-classification et connaissance partielle pour éviter l'explosion combinatoire liée à l'héritage multiple comme les exemples de classifier des automobiles dans [10, 14, 40].*

Besoin 4 *Catégorie ou Instance [10, 14]. Un élément peut être traité comme instance à un niveau mais comme une catégorie à un niveau inférieur. Par exemple dans la figure 1 Person est vu comme une instance de Class mais aussi comme la catégorie de Mary.*

Besoin 5 *Distinction entre l'instanciation globale (traversant deux niveaux de modélisation adjacents) de l'instanciation locale (existant entre types et instances au niveau M1).*

Besoin 6 *Relation entre catégories et/ou instances [10, 14].*

Besoin 7 *Contraintes de l'arité maximale/minimale pour un type relationnel. Ceci capture le nombre des éléments participant ensemble à un type de relations.*

Besoin 8 *Contraintes de cardinalité maximale/minimale pour un type relationnel. Ceci capture le nombre des instances de ce type relationnel.*

Besoin 9 *Contraintes de cardinalité maximale/minimale pour une méta-relation. Ceci capture le nombre de relations ayant cette méta-relation comme méta-relation (cas particulier du besoin 8). Par exemple, un élément est défini dans un seul modèle, et peut être utilisé dans plusieurs modèles.*

Besoin 10 *Contraintes de l'itération maximale/minimale pour un type relationnel. Ceci capture le nombre de répétitions pour une relation (ref. Définition6) instanciée de ce type relationnel.*

Besoin 11 *Relations essentielles pour les modèles : relation de contenant entre un modèle et ses éléments ; relation d'accès entre modèles pour réutiliser dans un modèle des éléments d'un autre ; relation du respect sémantique entre un modèle et son métamodèle.*

Besoin 12 *Relations entre modèles de M1 : relation d'équivalence entre modèles, relation de transformation de modèles, relation entre un modèle et ses différentes vues , etc.*

Besoin 13 *Distinction explicite entre la classification dynamique et la classification statique [40].*

Les besoins 7, 8 et 10 peuvent être combinées afin d'augmenter le pouvoir d'expression. Dans le problème ci-dessous, Les contraintes 1 à 4 sont des contraintes d'arité, les contraintes 5 et 6 sont des contraintes de cardinalité et la contrainte 7 est une contrainte d'itération. A notre connaissance, aucun des formalismes actuels de niveau M2 ne distingue ni ne supporte ces trois besoins afin de pourvoir modéliser des situations comme celle présentées dans le problème.

Problème. Comment peut-on modéliser la situation suivante ?

Soit au niveau M1, l'association signer-des-contrats (SignContracts) entre des participants. Cette association implique x types de fournisseurs (Supplier), y types de clients (Customer), z types de témoins (Witness) où : (contrainte1) $1 \leq x \leq 3$, (contrainte 2) $1 \leq y \leq 2$, (contrainte 3) $1 \leq z \leq 2$.

Chaque relation instanciée de SignContracts va relier les $(x+y+z)$ éléments suivants : x fournisseurs du type Supplier, y clients de type Customer et z témoins de type Witness. L'arité de SignContracts est en fait variée. De plus, Un fournisseur ou client peut être une organisation d'état (GouvernmentOrganization), une entreprise privée (PrivateEnterprise), ou une personne (Person). Un témoin peut être un avocat (Lawyer) ou une organisation (Organization).

Nous pouvons ajouter des contraintes sur les participants comme par exemple : (contrainte 4) parmi les fournisseurs pour une instance de SignContracts, il peut y avoir 1 ou 2 organisations d'états mais une personne au maximum. S'il y a possibilité de 1 ou 2 fournisseurs différents dans chacun des SignContracts avec les mêmes participants restants ($(x-1)$ fournisseurs, y clients et z témoins), alors (contrainte 5) la cardinalité sur Supplier pour SignContracts est de 1 à 2. Et s'il y a au maximum 2 entreprises privées différentes dont chacune participe à SignContracts sous le rôle de fournisseur avec les mêmes rôles restants ($(x-1)$ fournisseurs, y clients et z témoins) et mêmes joueurs de ces rôles, alors (contrainte 6) la cardinalité sur PrivateEnterprise sous le rôle Supplier pour SignContracts est de 0 à 2. Similairement, nous pouvons avoir pour SignContracts des autres contraintes de cardinalité sur des types de rôles et joueurs restants. Si des mêmes participants (x fournisseurs, y clients et z témoins) peuvent signer-des-contrats ensemble de 2 à 4 fois au maximum, alors (contrainte 7) le nombre d'itérations de SignContracts est de 2 à 4.

4.2. Besoins pour M3

Selon les caractéristiques du niveau M3 mentionnés dans la section 2, les éléments de M3 sont des instances d'éléments de M3 et les éléments de M2 sont des instances des éléments de M3. M3 doit supporter les huit premières définitions (section 3.2), des relations binaires (besoin 1 où n = 2) ainsi que les besoins 2, 4, 11 et 9.

5. Formalismes pour la représentation de modèles

Plusieurs formalismes ont été proposés pour modéliser le côté structurel des connaissances. Dans [14], en réponse aux trois besoins 3, 4 et 6, Gerbé a prouvé que les graphes conceptuels (GC) étaient le meilleur formalisme pour modéliser une mémoire d'entreprise parmi : le modèle relationnel, le modèle Entité-Association [6], Classic, UML1.x et les GC. Il a proposé un modèle uniforme des GC pour la modélisation et la métamodélisation des mémoires d'entreprise.

Pour l'ingénierie des modèles, Lemesle a analysé dans [22] le formalisme des réseaux sémantiques (RS), les GC, CDIF [13], XML, MOF1.3 et XMI, puis a proposé sNets basé sur les réseaux sémantiques.

Attachés aux applications Web, particulièrement au Web sémantique et ontologies, les formalismes tels que XML-XML Schema, RDF-RDFS, DAML+OIL [8], OWL sont en cours de standardisation auprès du W3C. OWL est une révision de DAML+OIL incorporant des expériences apprises des conceptions et applications de DAML+OIL.

Très récemment, de nouvelles versions de UML et MOF (UML2.0 et MOF2.0) ont été soumises à l'Object Management Group.

Nous avons étudié un certain nombre de formalismes vis-à-vis des besoins recensés pour la gestion de modèles. Nous avons choisi des formalismes utilisés dans chacun des domaines qui s'intéressent plus particulièrement à la représentation de modèles : le domaine de l'intelligence artificielle, le domaine du génie logiciel, et le domaine du Web sémantique.

- Intelligence artificielle : nous avons retenu les réseaux sémantiques incluant le formalisme sNets et les graphes conceptuels.
- Génie logiciel : nous avons choisi UML et MOF comme représentants de l'approche Orienté-Objet.
- Web sémantique : nous avons retenu XML-XML Schema, RDF-RDFS et OWL.

Il existe de nombreux autres formalismes. Notre choix s'est restreint à ceux qui nous semblaient les plus connus et les plus intéressants dans ces domaines.

5.1. Réseaux Sémantiques et sNets

Les réseaux sémantiques (RS) [35] sont un formalisme pour représenter des concepts et leurs relations par des graphes en donnant une sémantique aux noeuds et aux liens. Les noeuds sont les différents types d'information. Les types d'information sont les propositions, énoncés et propriétés. L'étiquette associée au lien indique le type de relation entre deux noeuds. Les RS permettent la représentation de toute sorte d'information mais il manque la modularité et la structuration de la sémantique de ces informations [22].

sNets [22] est dérivé des RS, sNets intègre la modularité et le typage. La figure 2 présente une partie du noyau réflexif de sNets. Dans sNets, chaque noeud (Entity) est identifié par son nom, est lié par un lien meta à son méta-entité (EntityType) et par un partOf à son modèle (Universe). Un lien entre deux entités doit se conforme à sa méta-relation de même nom définie au niveau méta. Chaque modèle (Universe) est en relation sémantique (sem) avec son méta-modèle (SemanticUniverse). La méta-relation extends est définie pour les besoins d'extension des modèles.

sNets est un formalisme simple et relativement facile à implémenter avec un noyau minimal. La version présentée dans [22] développe uniquement le méta-métamodèle de sNets avec les fonctionnalités

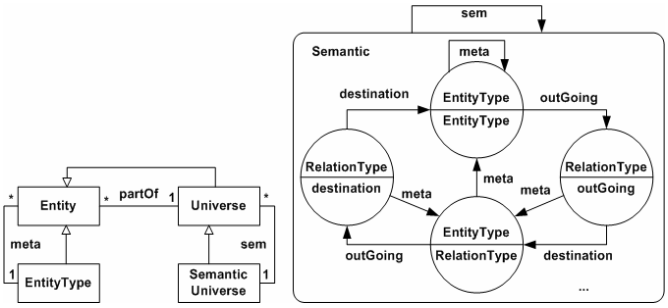

Figure 2. *Méta-métamodèle de sNets (figures 64 85-[22]).*

suivantes : la représentation des entités, méta-entités, relations et méta-relations binaires, l'héritage entre méta-entités mais pas entre méta-relations, les contraintes de cardinalité maximale d'une méta-relation, l'inversion entre méta-relations et les besoins 4 et 11.

Pour une utilisation de sNets, nous devrions (i) étendre son méta-métamodèle actuel et (ii) développer un métamodèle au niveau M2 de sNets pour qu'ils répondent à tous les besoins exposés dans la section 4.

5.2. Graphes Conceptuels (GC) et Modèle uniforme

La théorie des GC, qui modélise le monde réel par des concepts et relations conceptuelles, a été introduite par J. Sowa en 1984 [34], puis a été rapidement reconnue comme un formalisme de représentation des connaissances très puissant et proche de la langue naturelle [2, 34, 14]. Un modèle uniforme des GC [14] a été proposé pour la modélisation et la méta-modélisation des mémoires d'entreprise. La théorie des GC a également été appliquée à la gestion de modèles [16]. La figure 3 présente une partie de l'ontologie permettant la représentation des GC. La hiérarchie des types de concepts du langage se divise en deux branches : les éléments externes (ExternalElt) et les internes (InternalElt). Les éléments externes représentent les objets de l'univers du discours qui peuvent être référés par les éléments internes. Les éléments internes se regroupent en six types de base : Referent, Graph, Context, Type, CoreLink, GraphElt. Les référents (Referent) sont les représentants des objets. Les graphes (Graph) sont les phrases écrites dans le langage. Les contextes (Context) visent à regrouper les graphes. Les types (Type) catégorisent les référents. Les liens de co-référence (CoreLink) lient les concepts représentant les mêmes éléments. Les éléments de graphe (GraphElt) sont les éléments : arcs (source ou destination), relations, concepts (individu ou générique). Les graphes de définition (DefinitionGraph) servent à définir les types de concept ou de relation, tandis que les graphes de restriction (RestrictionGraph) contraignent les définitions des types de concepts. Avec ce langage, nous pouvons représenter les modèles aux trois niveaux de modélisation ainsi que d'autres formalismes comme UML, RDF-RDFS [16].

Cependant, le modèle uniforme [14] ne supporte pas tous les besoins (besoins 5, et 11 à 10) et il n'existe pas encore, à notre connaissance, un modèle des GC supportant tous nos besoins. De plus, l'implémentation des graphes conceptuels est difficile. Divers problèmes complexes sont encore ouverts. Par exemple : l'interprétation sémantique [20, 12] ; l'inférence dans les GC emboîtés complexes contenant des opérations de négation ou marqueurs génériques [25, 33] ; les projections et le «matching» dans les GC sont connus comme NP-complet [24, 32, 41].

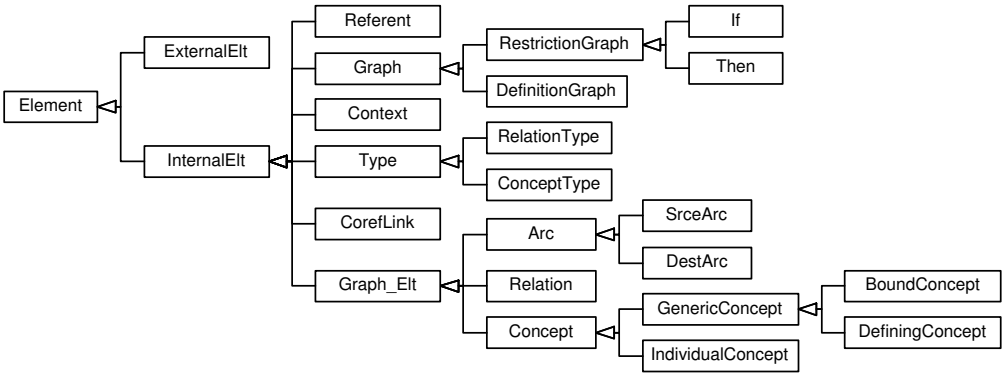

Figure 3. *Hiérarchie des types de concepts (figure 3.8-[14]).*

5.3. MOF et UML

MOF (Meta Object Facility) a pour but de définir un langage de modélisation unique pour représenter des modèles et métamodèles. MOF est utilisé pour sa propre définition, et aussi utilisé dans plusieurs technologies standardisées par OMG comme UML, CWM, XMI. Contrairement aux versions précédentes, MOF2.0 [28] suit MDA. La collaboration entre «UML2.0 Infrastructure» et MOF2.0 vise à une meilleure réutilisation et à l'intégration des concepts de modélisation afin de fournir une fondation pour MDA.

La figure 4 montre une partie du noyau réflexible de MOF2.0. Chaque objet (Object) est instancié d'une méta-classe (Class) qui décrit ses propriétés et ses opérations et qui est retournée par l'opération getMetaClass() de l'objet. Object est sous-type de Abstraction : :Elements : :Element. Tous les éléments de modèles qui spécialisent Reflection : :Object (ex. : tous les éléments dans «UML2.0 Infrastructure») héritent de cet objet la réflexivité. Chaque élément du type Factory crée des instances des types dans un paquetage (Package) auquel il est attaché.

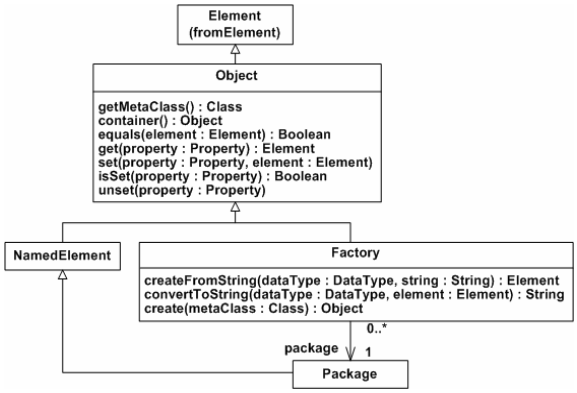

Figure 4. *Paquetage «Reflection» (figure 2 [28]).*

MOF2.0 définit aussi des collections et des séquences réflexives. Une collection réflexive (ReflectiveCollection) est une classe réflexive utilisée pour accéder aux propriétés qui peuvent avoir plusieurs valeurs possibles. Et une séquence réflexive (ReflectiveSequence) est une collection réflexive utilisée pour accéder aux propriétés ordonnées. Le paquetage Identity sert à identifier des objets des métamo-

dèles indépendamment des données de modèles qui peuvent être changées. Et le paquetage Extension propose un mécanisme pour l'extension des éléments de modèles avec des paires nom-valeur.

Un type relationnel (Association) dans MOF2.0 peut relier n éléments. Mais au niveau de l'implémentation, seulement les relations binaires sont permises. Un lien (Link) qui est une instance d'une association (Association) ne relie que deux objets (Object). MOF2.0 répond alors au besoin 2. Il répond aussi aux besoins 4 et 9. Concernant la modularité, MOF2.0 supporte le mécanisme de paquetages et les relations d'extension des paquetages comme PackageImport, PackageMerge qui sont représentés par les stéréotypes import, merge, combine. Pourtant, comme Lemesle l'avait remarqué dans [22] pour MOF1.3, la relation sémantique entre un modèle et son métamodèle n'est pas spécifiée dans MOF2.0.

UML (Unified Modelling Language) est reconnu comme un standard de facto parmi les formalismes Orienté-Objet au niveau M2. Ses versions précédentes (UML1.x) ne supportaient pas MDA (Model Driven Architecture) non plus que certains de nos besoins pour M2. Par exemple, comme elles traitaient de façon indépendante le niveau des classes et celui des instances, elles ne supportaient pas les besoins 5 et 6. La nouvelle version, UML2.0 [29, 30], apporte des points nouveaux et suit l'approche de méta-modélisation supportant MDA.

Le métamodèle de UML2.0 vise à modéliser les structures et les comportements des connaissances dans le champ du développement de logiciels. Il permet à l'usager de spécifier des modèles au niveau M1. Par exemple, des éléments définis au niveau M1 tels que des classes, classes relationnelles et instances sont instanciées des méta-classes concrètes Class, Association, AssociationClass, Property, InstanceSpecification et Package. UML2.0 remplit assurément nos cinq premiers besoins pour M2.

Dans UML2.0, les relations sont définies au niveau des classes et applicables au niveau des instances. Nous trouvons deux moyens permettant de supporter certains cas du besoin 6 : (i) des valeurs par défaut pour des attributs d'une classe ; (ii) le power-type (PowerTypes). L'usager peut spécifier des valeurs concrètes à certains attributs d'une classe lors du raffinement de celle-ci. Cependant uniquement les relations binaires sont représentables comme attributs d'une classe. Rappelons qu'un power-type est une classe dont les instances permettent catégoriser les instances en sous-classes. Toutes les instances ayant la même valeur du power-type forment un sous-classe. Donc, une classe est divisible en sous-classes plus spécifiques, chaque partition correspondant à une valeur du power-type. Cependant, ce mécanisme est incommode pour les cas où les partitions d'une classe se font pour divers power-types combinés. Le besoin 6 n'est pas complètement satisfait dans UML.

Concernant les mécanismes de modularité, UML supporte plusieurs notions comme les espaces de noms (Namespace), les paquetages (Package), les composants (Component) et les modèles (Model). Le métamodèle UML lui-même est divisé en différents paquetages afin de faciliter l'extension et la réutilisation, d'augmenter la cohésion et de diminuer le couplage entre paquetages. Des paquetages métamodèles peuvent toucher des sujets bien différents et peuvent être considérés comme des métamodèles séparés et sémantiquement différents. Alors, il est nécessaire d'indiquer clairement par lequel de ces métamodèles un modèle est spécifié. Autrement dit, il faut clarifier la relation sémantique qui existe entre un modèle et son métamodèle. Les relations entre paquetages comme import et merge servent pour la réutilisation des éléments ainsi que pour l'extension des paquetages. Toutes les dépendances entre modèles, ou plutôt les dépendances entre éléments que UML mentionne, servent au développement de logiciels et font partie du besoin 12. De plus, la relation entre un modèle et son métamodèle n'est pas identique à la relation instanceOf entre un élément et son méta-élément. C'est pourquoi, selon nous, cette relation la relation sémantique entre un modèle et son métamodèle n'est pas représentée formellement dans UML.

UML distingue entre les rôles statiques et les rôles dynamiques. Les rôles statiques («association ends») spécifient les fonctions des participants pour une relation donnée. Ils ressemblent aux rôles dans les modèles Entité-Association. Les rôles dynamiques spécifient les fonctions des participants dans une collaboration. Les collaborations décrivent les interactions pertinentes entre instances en identifiant les rôles spécifiques que ces instances joueront afin d'achever collectivement des fonctionnalités/comportements d'un système. Les interfaces permettent aux propriétés observables extérieurement d'une instance d'être spécifiées sans déterminer la catégorie («classifier») de cette instance. Ainsi, les rôles dans une collaboration seront souvent typés par des interfaces. Ils spécifieront les propriétés que les instances participantes devront exposer mais ne détermineront pas les classes précises de ces instances. Des instances de classes différentes peuvent jouer un même rôle défini par une interface donnée, et plusieurs interfaces peuvent être réalisées par une même classe. Voir [30] pour avoir plus détails sur ce sujet dans UML.

UML supporte le besoin 8 mais pas les besoins 7 et 10. Une situation telle que celle exposée dans le problème ne peut pas être modélisée dans UML.

5.4. XML-XML Schema

XML (Extensible Markup Language) [39] est un langage de balisage (markup) largement utilisé sur le Web. Il permet la représentation structurée d'informations dans un format texte et l'information représentée est encadrée par des balises. En XML, les balises sont redéfinissables. La définition des balises et leurs relations sont spécifiées dans une DTD (Document Type Description) ou dans un XML Schema. Concernant la représentation de modèles/schémas, XML Schema est plus expressif et utile que les DTD.

Pour la modularité, XML Schema [39] supporte les notions de schéma (schema), de type complexe (complexType), de type simple (simpleType), d'élément (element) et d'attribut (attribute). Un type complexe peut avoir des attributs et/ou contenir d'autres éléments, alors qu'un type simple ne peut pas se composer d'autres éléments et ne peut pas avoir d'attribut. Un élément peut appartenir à un type complexe ou simple mais un attribut ne peut qu'appartenir à un type simple. L'attribut fixed est utilisé dans la déclaration d'attributs et éléments pour affecter les valeurs particulières à ces éléments et attributs. Ce mécanisme est le moyen que nous trouvons dans XML Schema pour la possibilité de supporter le besoin 6. La relation d'instanciation entre un type et un élément est indiquée par l'attribut type, le besoin 5 n'est donc pas supporté. XML Schema ne permet pas le besoin 3 (multi-classification). Il supporte les mécanismes pour la redéfinition (redefine) et l'extension (extension) d'un schéma, mais ne supporte pas la relation entre modèle et métamodèle ni les relations citées dans le besoin 12. XML Schema supporte le besoin 8 mais pas le besoin 7 ni le 10. Nous ne pouvons donc pas modéliser des situations comme celle du problème.

Il y a plusieurs façons dans XML-XML Schema pour représenter une même information. Par exemple, le type relationnel works-for entre Person et Organization peut être codé de deux façons différentes : (i) Person et Organization sont les éléments de works-for ; (ii) works-for est un élément de Person, et Organization est un élément de works-for. Ceci montre la souplesse de XML XML Schema mais signale également l'ambiguïté sémantique dans la représentation et rend difficile le traitement sémantique des informations.

5.5. RDF-RDFS

RDF (Resource Description Framework) et RDFS (RDF Schema) [39] sont des langages pour la représentation sémantique d'informations sur le Web.

RDF définit trois types d'objets : ressources (resource), propriétés (property) et valeurs (value). Une ressource est un objet décrit par RDF et identifié par son URI (Uniform Resource Identifier). Une propriété est un attribut, un aspect, une caractéristique qui s'applique à une ressource. Il peut également s'agir d'une relation binaire d'une ressource avec une autre ressource. Les valeurs sont les valeurs particulières affectées aux propriétés. Une valeur peut être une ressource ou un littéral. Un littéral est un type primitif tel que un entier ou une chaîne de caractères. Une ressource est identifiée par un URI tandis qu'un littéral ne l'est pas. Une assertion RDF sur une ressource est représentée comme un triplet (ressource, propriété, valeur) ou bien (sujet, prédicat, objet). Il existe une représentation textuelle et une représentation graphique. Dans la représentation graphique, la ressource (sujet) et la valeur (objet) correspondent aux noeuds du graphe, et la propriété (prédicat) correspond à l'arc reliant le noeud ressource (sujet) au noeud valeur (objet).

RDFS spécifie le vocabulaire et la grammaire de RDF. RDFS supporte les notions primitives suivantes : classes et propriétés. Les ressources sont divisibles en groupes/classes (Class). Une classe est aussi une ressource identifiée d'une manière unique et décrit par des propriétés. La propriété rdf :type est utilisée pour indiquer qu'une ressource est une instance d'une classe. Une propriété est décrite comme une relation binaire entre une ressource sujet avec une ressource objet. Les propriétés rdfs :domain et rdfs :range restreignent le domaine d'application et celui de variation pour une propriété donnée. La relation d'héritage entre classes et celle entre propriétés sont définies respectivement par les propriétés rdfs :subClassOf et rdfs :subPropertyOf. Les besoins 2 et 3 sont supportés. Les cas de relations binaires (besoin 6) sont représentables dans RDF/RDFS mais leur interprétation sémantique n'est pas bien spécifiée. La figure 5 montre que EmployeeTeximus est une sous-classe de Employee et reliée par works-for à une organisation particulière Teximus. Donc, si Employee est prédéfini comme le domaine d'application de works-for, alors EmployeeTeximus étant sous-classe de Employee va être vu comme instance de Employee. Ceci implique une contradiction. RDF/RDFS ne supportent pas les relations autres que binaires (besoin 1) ni la définition d'attributs pour les relations. Comme RDF/RDFS ne supportent aucun des besoins 7, 8 et 10, RDF/RDFS ne peuvent pas modéliser des situations comme celle du problème. Comme XML/XML Schema et RDF/RDFS ne supportent pas le besoin 5, ni la relation entre modèle et métamodèle, ni les relations citées dans le besoin 12. De plus, RDF/RDFS laissent aux applications l'interprétation de certaines déclarations dans RDFS. Ceci rend plus difficile la construction d'un mécanisme d'inférence pour RDF.

```
...
<rdfs:Class rdf:ID="Employee"/>
<rdfs:Class rdf:ID="Organization"/>

<rdf:Property rdf:ID="works-for">
    <rdfs:range rdf:resource="#Organization"/>
</rdf:Property>
```

```
<rdf:Description rdf:ID="Teximus">
   <rdf:type rdf:resource="#      Organization   "/>
</rdf:Description>

<rdfs:Class rdf:ID="Employee-Teximus">
   <rdfs:subClassOf rdf:resource="#       Employee   "/>
   < works-for   rdf:resource="#   Teximus"/>
</rdfs:Class>
...
```

Figure 5. *Exemple de relations entre catégories et/ou instances.*

5.6. OWL

OWL (Web Ontology Language) [39] standardisé par W3C est un langage d'ontologies pour le Web. Il est basé sur RDF, et inspiré de DAML+OIL.

Il existe trois sortes d'OWL : OWL Lite, OWL DL, OWL Full. OWL Lite supporte les besoins primitifs comme la constitution de taxonomies ou de thesaurus. Il supporte également les contraintes de cardinalité simples où la valeur de cardinalité doit être 0 ou 1 afin de simplifier l'implémentation, de faciliter et d'accélérer la migration des taxonomies et thesaurus. OWL DL (OWL Description Logic) correspond aux logiques de description. Ce langage est expressif, et les procédures d'inférences sont complètes (complétude computationnelle), et effectuables en un temps fini (décidabilité). Dans OWL DL, les trois types : classes, propriétés et individus doivent être disjoints mutuellement. OWL Full offre une expressivité maximale, mais sans garantie quant à la complétude et à la terminaison des procédures d'inférence. Comme dans RDF, une classe dans OWL Full peut être traitée comme une collection d'individus ou comme un individu.

Une ontologie OWL est composée de triplets RDF. Elle peut contenir les éléments suivants : en-têtes optionnels (commentaire, version, importation d'ontologie), des éléments de classe, des éléments de propriétés et des instances.

Une classe OWL est soit anonyme, soit désignée par son URI. owl :Thing et owl :Nothing sont deux classes pré-définies dans OWL. Tout objet est instance de la classe owl :Thing, et owl :Nothing n'a aucune instance. Dans OWL Full, la classe owl :Thing est équivalente à rdf :Ressource, et owl :Classe équivalente à rdf :Classe. Pour définir une classe, OWL supporte entre autres : des opérateurs de relation et de spécialisation entre classes (owl :subClassOf, owl :disjointWith, owl :equivalentClass), des opérateurs booléens (owl :intersectionOf, owl :unionOf, owl :complementOf), l'énumération exhaustive des instances de la classe et la restriction sur une propriété d'une classe préexistante (owl :allValuesFrom, owl :hasValue, owl :someValuesFrom, owl :cardinality, owl :maxCardinality, owl :minCardinality).

OWL distingue deux sortes de propriétés : les propriétés reliant des individus (owl :ObjectProperty), et les propriétés reliant des individus à des types de données (owl :DatatypeProperty). Ces deux sortes de propriétés, owl :ObjectProperty et owl :DatatypeProperty sont bien disjointes dans OWL Lite et OWL DL. Alors que dans OWL Full, owl :DatatypeProperty est une sous-classe de owl :ObjectProperty (c'est-à-dire les valeurs de données font partie du domaine des individus), et owl :ObjectProperty est équivalente à rdf :Property. Pour définir des propriétés, OWL réutilise les éléments de RDFS comme rdfs :subPropertyOf, rdfs :domain, rdfs :range et en définit d'autres comme : propriétés équivalentes (owl :equivalentProperty), propriétés inversées (owl :inverseOf) ; propriétés fonctionnelles (owl :FunctionalProperty), propriétés fonctionnelles et inversées (owl :InverseFunctionalProperty).

Comme RDF/RDF Schema, OWL peut représenter les cas de relations binaires concernant le besoin 6. OWL ne satisfait pas le besoin 4, ni la relation sémantique entre modèle et métamodèle, ni la relation de transformation entre modèles. OWL ne peut pas modéliser le problème.

6. Conclusion et Travail futur

Comme nous l'avons vu, les travaux sur la gestion de modèles sont applicables dans divers domaines : gestion des connaissances, gestion de méta-données (base de données), ontologies, qualité de service

(QoS) et génie logiciel. C'est pourquoi nous avons jugé intéressant de faire une revue de la littérature sur ce sujet. Nous avons proposé quelques définitions et quelques besoins et avons fait une étude sur les différents formalismes susceptibles de répondre aux besoins de la gestion de modèles. Notre étude bien que non exhaustive a pris en compte, nous l'espérons, les formalismes les mieux adaptés à cette problématique.

Aucun des formalismes étudiés ne rencontrent toutes les exigences pour la gestion de modèles. De plus, un aspect de la modélisation n'a pas été abordé dans cet article. Il s'agit de la notion de rôle, un rôle est une représentation des aspects temporaires et dynamiques des objets du monde réel [7, 9, 17, 21, 31, 34, 36, 37, 40]. Par exemple, une personne, durant son existence, passer de différents états comme enfant, adolescent, adulte, et elle peut jouer des rôles comme étudiante, employée, célibataire ou mariée, etc. C'est un point important qui doit être pris en compte dans la gestion de modèles.

Nous travaillons actuellement dans ce sens et cherchons à spécifier un formalisme, ou une extension d'un formalisme existant qui serait susceptible de satisfaire tous les besoins de la gestion de modèles.

7. Bibliographie

[1] S. Alagic and P.A. Bernstein. A Model for Generic Schema Managemen. In *Proceedings of DBPL 2001*, volume 2397/2002, pages 228–246, , October 29-30 2001. Springer-Verlag Heidelberg.

[2] American National Standard. Conceptual Graph Standard. 1999.

[3] P. Bernstein, A. Halevy and R. Pottinger. A Vision for Management of Complex Models. *ACM SIGMOD Record*, 29(4), pages 55–63, 2000.

[4] J. Bézivin. On Different Interoperability Modes in Software Engineering : the Case of Modeling Activities at OMG. In *Proceedings of Software Engineering'98*, Paris, France, 1998.

[5] J. Bézivin and O. Gerbé. Towards a Precise Definition of the OMG/MDA Framework. In *Proceedings of the 16th Conference on Automated Software Engineering*, pages 273–280, San Diego, USA, November 2001. IEEE Computer Society Press.

[6] P. Chen. The Entity-Relationship Model – Towards a Unified View of Data. *ACM Transactions on Database systems*, 1(1), pages 9–36, 1976.

[7] W.W. Chu and G. Zhang. Associations and roles in object-oriented modeling. In *Proceedings of ER'97. Lecture Notes in Computer Science, Vol. 1331.*, pages 257–270, California, USA, 1997. Springer.

[8] D. Connolly, F.v. Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider and L.A. Stein. DAML+OIL (March 2001) Reference Description. March 2001. W3C Note.

[9] M. Dahchour. Integrating Generic Relationships into Object Models Using Metaclasses. 2001. Université catholique de Louvain, Belgium.

[10] T.L.A. Dinh. Métamodèle pour la gestion des modèles. 2003. University of Montreal, Montreal, Canada.

[11] T.L.A. Dinh and O. Gerbé. A Metamodel for Knowledge Management. In *RIVF'04 - International Conference of French-Speaking or Vietnamese Computer Scientists*, pages 107–112, Hanoi, Vietnam, February 2004.

[12] J. Esch and R. Levinson. An Implementation Model for Contexts and Negation in Conceptual Graphs. In *Proceedings of ICCS'95*, pages 247–262, CA, USA, 1995.

[13] R.G. Flatscher. Metamodeling in EIA/CDIF-Meta-Metamodel and Metamodels. *ACM Trans. on Modeling and Computer Simulation (TOMACS)*, 12(4), pages 322–342, 2002.

[14] O. Gerbé. Un modèle uniforme pour la modélisation et la métamodélisation d'une mémoire d'entreprise. 2000. Université de Montréal, Montréal, Canada .

[15] O. Gerbé, B. Kerhervé and U. Srinivasan. Model Operations for Quality-Driven Multimedia Delivery. In *In Contributions to ICCS 2003. Auxiliary Proceedings of the 11th International Conference on Conceptual Structure*, Dresden, Germany, 2003.

[16] O. Gerbé, G. Mineau and R. Keller. La métamodélisation et les graphes conceptuels. 2003. Cahier du GReSI no 03-01, HEC Montréal.

[17] G. Gottlob, M. Schrefl and B. Rock. Extending object-oriented systems with roles. *ACM Trans. Office Inform. Systems*, 14(3), pages 268–296, 1996.

[18] ISO. ISO-IEC 10027 : Information technology - Information Resource Dictionary System (IRDS) - Framework, ISO/IEC International standard. 1990.

[19] B. Kerhervé and O. Gerbé. Model Management for Quality of Service Support. In *Proceedings of the 14th Int. Conf. on Soft. and Syst. Engineering and their Applications, Vol. 1.* , Paris, France, 2001.

[20] P. Kocura. Semantics of Attribute Relations in Conceptual Graphs. In *Proceedings of the ICCS 2000*, pages 235–248, Germany, 2000. Springer.

[21] B.B. Kristensen. Object-Oriented Modeling with Roles. In *Proceedings of the 2nd Int. Conf. on Object-Oriented Information Systems (OOIS'95)*, Ireland, 1995.

[22] R. Lemesle. Techniques de Modélisation et de Méta-modélisation.. 2000. Université de Nantes, France.

[23] S. Melnik. Generic Model Management. Ph.D Thesis. 2004. Springer.

[24] M.L. Mugnier and M. Chein. Polynominal algorithms for projection and matching. In *Proceedings of the 7th annual Workshop on Conceptual Graphs*, pages 239–251, Las Cruces, NM, USA, July 1992. Springer-Verlag.

[25] M.L. Mugnier. Knowledge Representation and Reasonings Based on Graph Homomorphism. 2000. RR-LIRMM 00-098.

[26] J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis. Telos : Representing knowledge about information systems. *ACM Trans. on Inform. Systems*, 8(4), pages 325–362, 1990.

[27] OMG. MDA Guide Version 1.0.1. 2003. Joaquin Miller and Jishnu Mukerji (ed.).

[28] OMG. Meta Object Facility (MOF) 2.0 Core Specification. 2003. OMG Adopted Specification, ptc/03-10-04.

[29] OMG. Unified Modeling Language : Infrastructure. 2003. OMG Adopted Specification, ptc/03-09-15.

[30] OMG. Unified Modeling Language : Superstructure. 2004. OMG Adopted Specification, ptc/04-10-02.

[31] B. Pernici. Objects with Roles. In *Proceedings of ACM-IEEE Conference of Office Information Systems (COIS)*, pages 205–215, Boston, USA, April 1990.

[32] J. Poole and J.A. Campbell. A Novel Algorithm for Matching Conceptual and Related Graphs. In *Proceedings of the 3nd Int. Conf. on Conceptual Structures, ICCS'95*, pages 293–307, , 1995. Springer.

[33] S. Prediger. Nested Concept Graphs and Triadic Power Context Families : A Situation-Based Contextual Approach. In *Proceedings of the ICCS 2000*, pages 249–262, Germany, 2000. Springer.

[34] J.F. Sowa. Conceptual Structures - Information processing in mind and machine. 1984. Addison wesley 14472.

[35] J.F. Sowa and A. Borgida. Principles of Semantic Networks : Explorations in the Representa-tion of Knowledge. 1991. Morgan Kaufmann Publishers, San Mateo, CA.

[36] J.F. Sowa. Knowledge Representation : Logical, Philosophical and Computational Foun-dations. 2000. BooksCole.

[37] F. Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data and Knowledge Engineering*, 35pages 83–106, 2000.

[38] M. Theodorakis, A. Analyti, P. Constantopoulos and N. Spyratos. A theory of contexts in information bases. *Information Systems*, 27(3), pages 151–191, 2002.

[39] W3C. http ://www.w3.org/. site web du W3C .

[40] R. Wieringa, W. De Jonge and P. Spruit. Using dynamic classes and role classes to model object migration. *Theory and Practice of Object Systems*, 1(1), pages 61–83, 1995.

[41] M. Willems. Projection and Unification for Conceptual Graphs. In *In Conceptual structures : applications, implementation, and theory - Proc. of ICCS'95*, pages 278–292, 1995. Springer.